OpenCV

last modified by Forestier Robin

on 2021/08/23 07:58

Open CV / Reconnaissance visuel

Page :

Multi caméra

Table des matières

J'ai décidé de me spécialiser en traitement d'images durant ma troisième et ma quatrième année d'apprentissage.

Ce XWiki contient une explication de toutes les bases utiles pour le traitement d'images et la reconnaissance d'images.

Gitlab : Reconnaissance visuel (http://172.16.32.230/Forestier/reconnaissance-visuel)

Si vous êtes intéressé par se sujet, je vous recommande de commencer par lire ce livre qui vous apportera toutes les bases nécessaire.

Pratical Python and OpenCV : http://172.16.32.230/Forestier/reconnaissance-visuel/blob/master/1_Documentation/Practical_Python_and_OpenCV__3rd_Edition.pdf

1.0 - Matériel

Voici tout le matériel que j'ai utilisé durant ma spécialisation.

Nom	Utile	Liens	Prix
Raspberry Pi 3B+	✓	Raspberry	40 .-
Raspberry Pi cam V1.3	✓	Raspberry Pi cam (V2)	30 .-
Nvidia jetson TX2	✗	jetson TX2	600 .-
Coral USB accelerator	✗	USB accelerator	60 .-
OpenCV AI kit : OAK-D	✓	OAK-D	300 .-

Liens :

Raspberry : https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
Raspberry Pi cam (V2) : https://www.raspberrypi.org/products/camera-module-v2/
jetson TX2 : https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-tx2/
USB accelerator : https://coral.ai/products/accelerator
OAK-D : https://store.opencv.ai/products/oak-d

1.1 - Nvidia Jetson Tx2

La carte Nvidia Jetson TX2 est une carte de développement pour l'AI (artificial inteligent) et le machin learning.
Elle est, d'après ses caractéristiques, la meilleure carte de développement disponible actuellement sur le marché (2020).

J'ai souhaité l'utiliser dans le cadre de ma spécialisation.
Après plusieurs semaines d'essaies, j'ai décidé d'abandonner cette carte et de revenir sur mon Raspberry Pi 3B+.
Pourquoi ?

Son seul point positif étant :

- Une grande puissance de calcul.

Et ses points négatifs :

- Difficile à mettre en place.
- La carte ne fonctionne que sur son OS, créer par Nvidia.
- Impossible d'accéder à la caméra on board avec OpenCV.
- Prix très élevé.
- Manque de documentation.

1.1.1 - Liens

XWiki : Jetson TX2

Nvidia : https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-tx2/

1.2 - Coral USB accelerator

Voici l'USB accelerator de Coral.

Coral USB Accelerator est un module externe vous ajoutant un coprocesseur Edge TPU, qui vous réalisera l'entièreté de vos calculs en rapport à l'intelligence artificielle à grande vitesse et avec un très petit délai. Ce boitier développé par Coral (Google), fonctionne avec Linux, Mac et Windows. Il est aussi compatible avec Tensorflow lite et disponible au prix de ~60 CHF sur Farnell.

1.2.1 - Liens

Coral : https://coral.ai/products/accelerator

Farnell : https://ch.farnell.com/fr-CH/pi3g/g950-01456-01/accelerateur-usb-raspberry-pi/dp/3583033?st=usb%20accelerator

1.3 - OAK-D

Voici OAK-D, le Kit AI créé par OpenCV.

Avec ses trois, caméras il permet de réaliser une reconnaissance en simultané avec une vision de la profondeur.

Informations des caméras :

- Une caméra centrale de 12MP à 60fps avec un autofocus.
- Deux caméras synchronisées avec 1MP à 120fps.

On la connecte en USB-C directement aux Raspberry Pi et on l'alimente avec l'adaptateur secteur 5V.

1.3.1 - Liens

Xwiki : Opencv multi camera

OpenCV : https://store.opencv.ai/products/oak-d

Uctronics : https://www.uctronics.com/opencv-ai-kit-oak-d-ov9282-2-imx378-intel-movidius-myriad-x-%20oak-d.html

2.0 - Installation

2.1 - Instalation d'OpenCV et de Python

Voici les commandes à entrer dans le terminal.

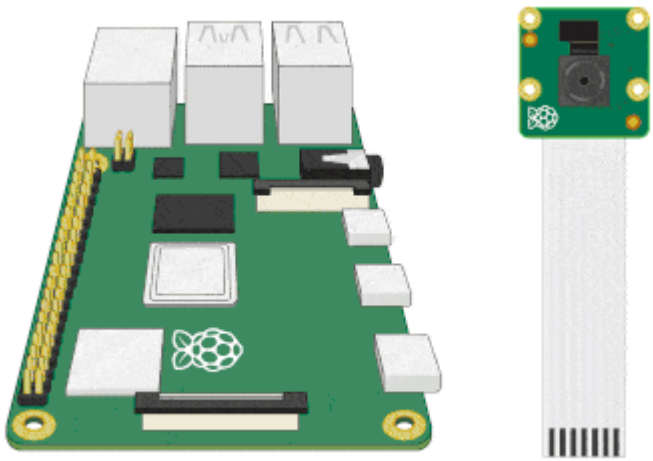
```
sudo apt update
sudo apt install python3
sudo apt install python3-opencv
python3
import cv2
cv2.__version__
exit()
sudo apt update
sudo apt install python3-pip
pip3 install opencv-python
python3
import cv2
cv2.__version__
exit()
```

Vous avez maintenant la dernière version d'Open CV installée.

2.2 - Installation de la caméra

Pour brancher la caméra veillez à :

- Ne pas être charger en électricité statique
- Débrancher l'alimentation de votre Raspberry Pi



- Dans les réglages d'interface du Raspberry Pi, activer la caméra.

2.2.1 - Tester la caméra

Pour tester la caméra taper la commande suivante dans votre terminal.

Pour prendre une photo : raspistill -o photo_01.jpg -t 5000
Pour prendre un vidéo : raspivid -o video_01.h264 - 5000

3.0 - Opencv

3.1 - Présentation

OpenCv est une bibliothèque graphique développée par Intel depuis 2000, elle est disponible sur la majorité des plateformes comme Windows, Mac, Linux, IOS... Elle fonction avec plusieurs langages comme python, java et C++.
OpenCv est sous licence BSD (Berkeley Software Distribution Licence) ce qui permet à n'importe qui de l'utiliser même pour un projet commercialisé.
OpenCv propose plus de 2500 algorithmes pour effectuer différents traitements sur une image comme de la détection de couleur, de l'extraction d'informations etc.

3.2 - Programmes

Vous trouverez lentièreté de mes programmes de test ici.
Pour les tester, utilisez Thonny disponible de base sur Raspberry Pi.
Vous trouverez ci-dessous la documentation officielle d'Open CV ansi qu'un site expliquant l'entièreté des fonctions et le site PyImageSearch avec plein de petits projets.

- OpenCV : https://opencv.org
- LearnOpenCV : https://learnopencv.com
- PyImageSearch : https://www.pyimagesearch.com

3.3 - Test d'OpenCV

Pour tester OpenCV, vous allez réaliser un petit programme en python qui affichera la caméra. Pour cela, ouvrez un IDE python (comme Thonny Python IDE).

```
#Programme test de la caméra
import cv2
cap = cv2.VideoCapture(0)
while(True) :
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Camera', gray)

   #arrêt du programme si la touche "q" est pressée
   if cv2.waitKey(1) & 0xFF = ord('q') :
      break
#décharge de la mémoire
#fermeture de toutes les fenêtres
cap.release()
cv2.destroyAllWindows()
```

En exécutant ce code vous devriez voir apparaître un fenêtre nommée Caméra ou vous, vous voyer en noir et blanc.

4.0 - Lecture et affichage d'une image

Gitlab : 1_affichage_image

Pour travailler avec OpenCv, il nous faudra une image ou une vidéo.
Pour commencer, nous allons simplement sélectionner une image sur notre Raspberry et l'afficher.

- Ouvrez votre IDE python (Thonny)
- Si vous avez cloné mon gitlab, vous pouvez ouvrir l'exemple sous : 5_Programmation/test/1_affichage_image/affiche_image.py

Dans ce cas, nous allons ouvrir l'image minecraft.jpg se trouvant dans notre fichier.

4.0.1 - Code

```
#affiche_image.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import cv2
#charge l'image minecraft.jpg dans my_image
my_image = cv2.imread('minecraft.jpg')
#affiche my_image / titre = mon image
cv2.imshow('mon image', my_image)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

La fonction cv2.imread permet de récupérer une image dans vos fichiers pour l'utiliser dans votre programme.
Si vous ajouter le paramètre cv2.imread('minecraft.jpg' , 0) votre image sera chargée en noir et blanc directement.

La fonction cv2.imshow, permet d'afficher votre image (ou vidéo).
Le premier paramètre représente le nom de votre fenêtre, le deuxième, la variable à afficher.

4.0.2 - Erreur

Si vous obtenez l'erreur suivante :

```
cv2.imshow('mon image', my_image)
error : (-215:Assertion failed) size.with>0 && size.height>0 in function 'imshow'
```

Cela signifie que votre programme n'as pas réussi à ouvrir ou à trouver votre image.
Vérifiez que votre image se trouve bien dans le même dossier que votre programme et que son nom et son format sont identique dans votre programme.

4.1 - Enregistrer une image

Pour enregistrer une image, il faut utiliser la fonction :
cv2.imwrite('image_01.png', img)
Il ne faut pas oublier l'extension du fichier (.jpg/.png/...)

5.0 - Histogramme d'une image

Gitlab : 2_histogramme

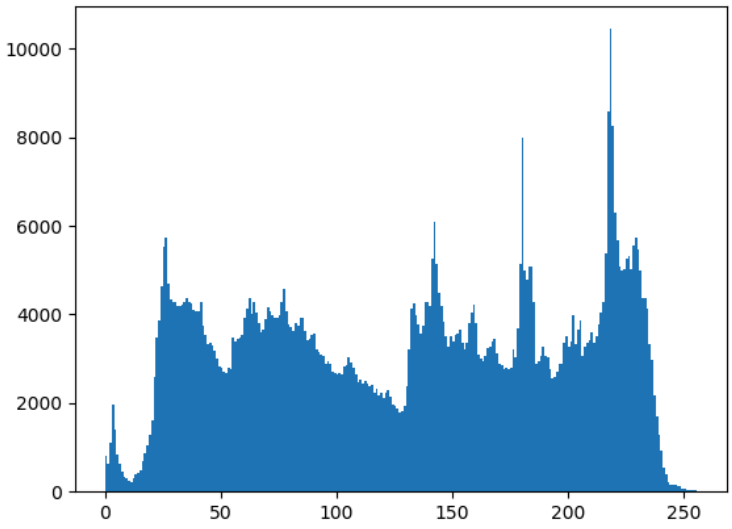
Un histogramme est une représentation du nombre de pixel à une intensité lumineuse précise.

Pour afficher l'histogramme, on utilise la bibliothèque pyplot.
Si vous utilisez une image entièrement blanche, votre histogramme affichera une grande barre tout à droite. En insérant une image noire, la barre se situera tout à gauche.
Nous pourrons utiliser ces valeurs visualisées pour le "seuillage" d'image.

5.1 - Code

```
#hystogramme.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque opencv et matplotlib
import cv2
from matplotlib import pyplot as plt
#charge l'image minecraft.jpg dans img en nuance de gris
img = cv2.imread('minecraft.jpg',0)
#calcule de l'histogramme
plt.hist(img.ravel(),256,[0,256])
#affichage de l'histogramme
plt.show()
```

5.2 - Exemple



6.0 - Lissage d'une image

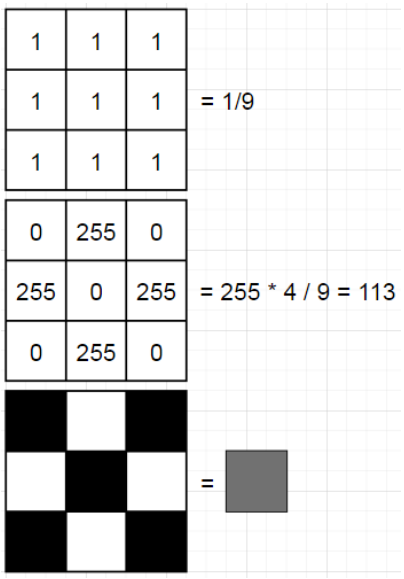
Gitlab : 3_lissage

Le lissage est une fonctionnalité très utile est très utilisée. Elle permet, par exemple, de réduire le bruit.
Pour lisser une image il existe plusieurs algorithmes, le plus souvent ce sont des algorithmes à box.
Imaginez avoir une image de 10x10 pixels de côté, et que nous devons le filtrer avec une boite de 3x3 pixels.

6.1 - Average

Pour le faire, la première méthode est le filtrage Average.

Pour le réaliser, positionner le haut de votre box parallèlement au haut de votre image mais 1px décalé vers la gauche (ce qui permet de ne pas perdre de qualité), et réaliser une moyenne de la valeur de chaque pixel puis placer le résultat aux centres.



Si vous réalisez ce lissage en plaçant votre box sur le coint en haut à gauche de votre image vous perdrez 1px de résolution dans tous les sens (haut, bas, droite, gauche), ce qui vous ferra perdre 2x2 pixel de résolution totale.

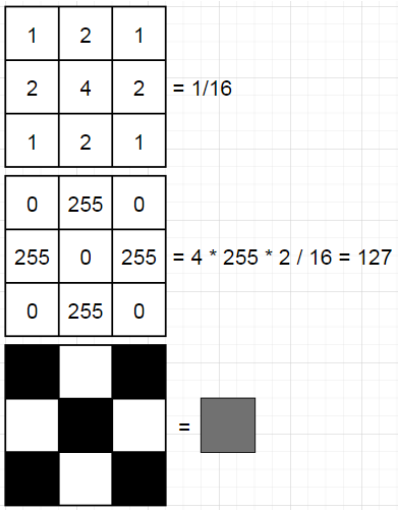
6.1.1 - Code

```
average_image = cv2.blur(img,(3,3))
#affichage
plt.imshow(average_image); plt.show()
```

6.2 - Gaussian

La deuxième méthode s'appelle Gaussian.

Cette méthode fonctionne aussi sur le fonctionnement par box. Mais chaque pixel n'a pas la même valeur :



L'explication du dessus, est une approximation du calcul réellement effectué.

6.2.1 - Code

```
gaussian_image = cv2.GaussianBlur(img,(3,3),0)
#affichage
plt.imshow(gaussian_image); plt.show()
```

6.3 - Median

La troisième fonction est la fonction Médian.

Elle réalise la médiane des valeurs se trouvant dans la box.
Le lissage par médian est très utile dans la suppression de bruit dans une image.

6.3.1 - Code

```
median_image = cv2.medianBlur(img,3)
#affichage
plt.imshow(median_image); plt.show()
```

6.4 - Bilateral

Le dernier filtre est le filtre bilateral.

Le filtre bilatéral est un filtre de lissage non linéaire, préservant les bords et réduisant le bruit dans les images.
Ce filtre est basé sur le « Gaussian Blur ».
Son seul désavantage est qu'il est plus lent que les trois autres filtres.

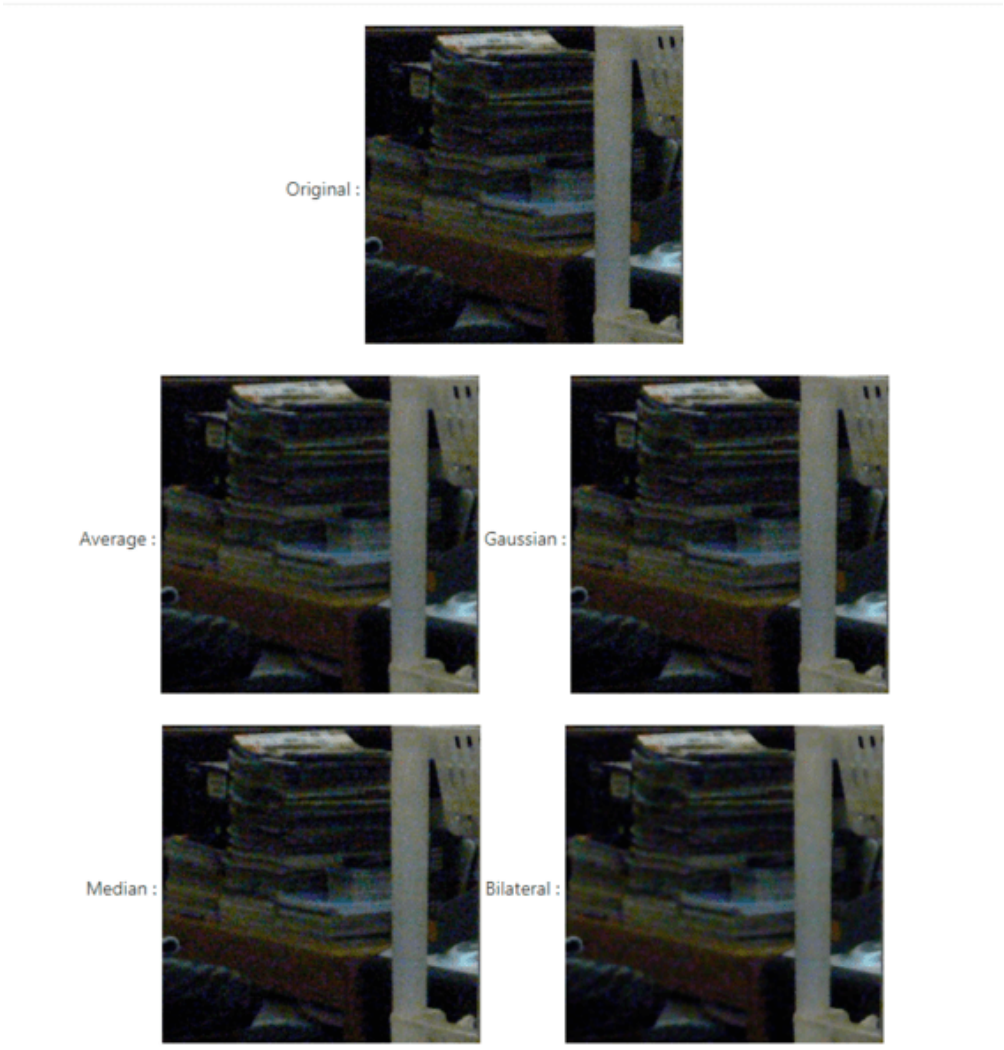
6.4.1 - Code

```
bilateral_image = cv2.bilateralFilter(img,9,75,75)
#affichage
plt.imshow(bilateral_image); plt.show()
```

Paramètres :

1. Image
2. Diameter of each pixel neighborhood. (Diamètre de chaque échantillonnage de pixels)
3. Plus la valeur est élevée plus les couleurs éloignées vont se mélanger.
4. Plus la valeur est élevée plus les couleurs se mélange si elle se trouve dans la plage de donnée.

6.5 - Exemple



On observe une réelle différence sur la barre blanche au premier plan.

(Pour une meilleure qualité : Gitlab - Reconnaissance visuel)

7.0 - Seuillage d'une image

Gitlab : 4_seuillage

Le seuillage ou thresholding en anglais est utile dans le traitement d'image. On peut l'utiliser pour retrouver des informations au sein d'une image ou de venir détourer un sujet dans notre image. Il existe 3 méthodes pour réaliser un seuillage d'image.

7.1 - Seuillage simple

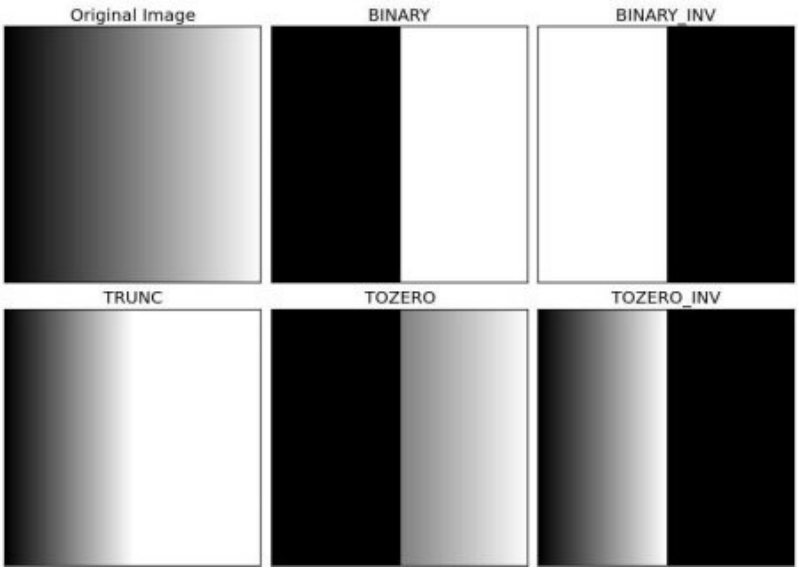
Le seuillage simple vient regarder la valeur du pixel si elle se trouve en dessous de la valeur de référence, le pixel serra noir sinon le pixel sera blanc.

7.1.1 - Code

```
ret, thresh = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

Paramètres :

1. Img => image originale
2. 127 => valeur de seuille
3. 255 => valeur qui sera attribuée aux pixels
4. cv2.THRESH_BINARY => peut être remplacer par les exemples ci-dessous.



7.2 - Seuillage adaptif

Le lissage adaptif est très utile dans un cas ou l'image n'as pas la même luminosité partout. Elle vient réalisé une lissage par zone. Il y a deux méthodes, la première, demande comme troisième paramètre : cv2.ADAPTIVE_THRESH_GAUSSIAN_C. Elle réalisera un lissage Gaussian. La deuxième réalisera un lissage par moyenne : cv2.ADAPTIVE_THRESH_MEAN_C.

7.2.1 - Code

```
thresh = cv2.adaptiveThreshol(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 91, 5)
thresh2 = cv2.adaptiveThreshol(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 91, 5)
```

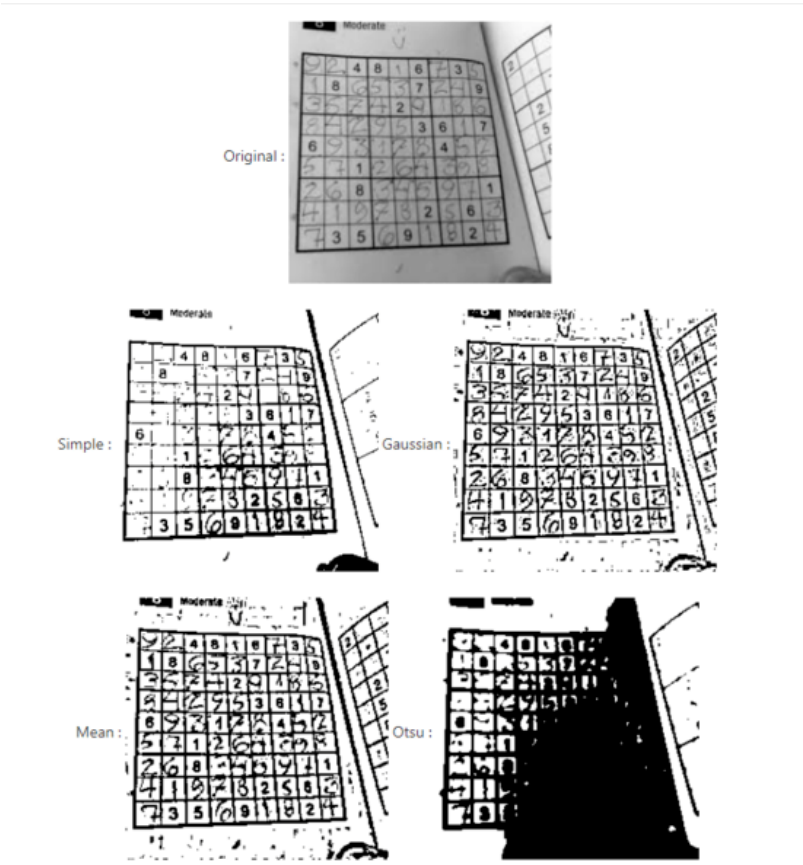
7.3 - Seuillage OTSU

Le lissage OTSU est utilisé quand une image présente deux piques au sein de son histogramme.
On vient d'abord lisser l'image avec un lissage Gaussian puis on applique le lissage. La spécificité de ce lissage c'est qu'il vient lui-même calculer le seuille et pour cela on le met à 0.

7.3.1 - Code

```
blur = cv2.GaussianBlur(img,(5,5),0)
ret,th = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

7.4 - Exemples



On voit une réelle différence avec la méthode adaptive.

(Pour une meilleure qualité : Gitlab - Reconnaissance visuel)

8.0 - Masquage de couleur

Gitlab : 5_masquage

Le but de ce chapitre va être d'isoler une ou plusieurs couleurs affichées à l'écran.
Pour cela, nous aurons besoin d'une plage de couleur que l'on veut isoler.

Pour essayer nous allons prendre cette photo de la terre :

La première chose est de l'importer dans notre programme.
La deuxième sera de la convertir en HSV.
Ensuite de créer notre plage de couleur.
Créer le mask.
Créer le reste.
Afficher l'image, le masque et le reste.

Cette méthode fonctionne à merveille avec une image mais, avec la caméra certaines difficultés s'ajoutent.
La lumière modifie énormément la couleur, la luminosité aussi.

8.0.1 - Code

```
import cv2
import numpy as np
#importation de l'image terre
frame = cv2.imread('terre.png')
#convertion en HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
#création de la plage de couleur (Bleu pour les océans)
lower_blue = np.array([110,50,50])
upper_blue = np.array([130,255,255])
#création de mask
mask = cv2.inRange(hsv, lower_blue, upper_blue)
#création du reste
rest = cv2.bitwise_and(frame,frame,mask = mask)
#affichage
cv2.imshow('frame',frame)
cv2.imshow('mask',mask)
cv2.imshow('rest',rest)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8.0.2 - Calcul de la plage de couleur

Pour calculer la plage de couleur, utiliser ce code :

```
import cv2
import numpy as np
color = np.uint8([[[0,0,0]]])
hsv_color = cv2.cvtColor(color,cv2.COLOR_BGR2HSV)
print(hsv_color)
```

Remplacer à la troisième ligne le 0,0,0 par le code **BGR** de la couleur choisie.

On garde le premier résultat, on soustrait dix.
On reprend le premier résultat et on lui ajoute dix.

Le premier donnera la valeur basse x,50,50.
Le deuxième donnera la valeur haute x,255,255.

8.0.3 - Exemple de plage de couleur

RGB => 25,50,200
BGR => 200,50,25

HSV => 116,223,25

lower_blue => 106,50,50
upper_blue => 126,255,255

8.0.4 - Exemple

Image de la terre

Image des déserts de sable

Mask pour les océans

images des océans

9.0 - Canny edge detection

Gitlab : 6_canny_edge

Détourage de forme et affichage de contour d'objet.

9.0.1 - Code

```
edge = cv2.Canny(img, 100, 150)
```

Paramètres :

1. Votre image
2. minVal
3. maxVal

minVal et maxVal permette de détecter les contours (les bords), plus ils sont réglés bas plus les détails seront présents.

10.0 - Template matching

Gitlab : 7_template_matching

Le template matching permet la reconnaissance d'un bout d'image dans une image.
L'algorithme est assez restrictif, il sera donc compliqué de reconnaître un objet d'une autre image.
Pour le test, vous aurez besoin de couper un endroit de votre image d'origine.

Original :

Template :

(Pour une meilleure qualité : Gitlab - Reconnaissance visuel)

Pour le code, on va utiliser la fonction **cv2.matchTemplate()** et la fonction : **cv2.minMaxLoc()**.
On commence par importer nos deux images en **gris**.

Et on vient récupérer la taille de template.
On appelle la fonction **cv2.matchTemplate()** en lui envoyant les paramètres : img, template, method.
Il existe 6 méthode différentes :

cv.TM_CCOEFF / cv.TM_CCOEFF_NORMED
cv.TM_CCORR / cv.TM_CCORR_NORMED
cv.TM_SQDIFF / cv.TM_SQDIFF_NORMED

10.1 - Code

```
#template_matching.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque OpenCv et numpy
import cv2
import numpy as np
#charge l'image minecraft.jpg dans my_image
img = cv2.imread('minecraft.jpg',0)
#charge l'image a rechercher steve.jpg dans template
template = cv2.imread('steve.jpg',0)
#taille de template
w, h = template.shape[::-1]
#matching des images
res = cv2.matchTemplate(img, template, cv2.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where(res >= threshold)
#affichage des carrés noirs
for pt in zip(*loc[::-1]):
    cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h),(0,255,255))
#affiche img / titre = Detected
cv2.imshow('Detected',img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

Ce code permet de reconnaître aussi plusieurs fois le même objet dans l'image.

10.2 - Exemple

Résultat :

(Pour une meilleure qualité : Gitlab - Reconnaissance visuel)

11.0 - Vidéo

Gitlab : 8_video

Le but sera d'utiliser la caméra pour afficher en direct ce qu'elle voit.

Pour commencer, il vous faudra installer et brancher la caméra au Raspberry pi comme indiqupé aux points 3 et 3.1.

Ensuite, nous allons afficher en direct ce que voit notre caméra. Pour cela, on utilise la fonction **cv2.VideoCapture(0)** le **0** sert à sélectionner la caméra, si vous en avez plusieurs, changez le 0 par 1 etc.

On vient récupérer le flux image par image, on le transforme en noir et blanc et on le réaffiche.

Ne pas oublier Libérer la capture.

11.1 - Code

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

11.2 - Qualité et FPS

Modifier la qualité et le nombre de FPS (images par seconde).

```
cap = cv2.VideoCapture(0)
#réglage des FPS à 60
cap.set(cv2.CAP_PROP_FPS, 60)
#réglage calité 480p
ret = cap.set(3,704)
ret = cap.set(4, 480)
```

Pour afficher ces valeurs.

```
fps = int(cap.get(5))
h = int(cap.get(3))
w = int(cap.get(4))
print("fps : ", fps, " quality : ", h, ",", w)
```

12.0 - Dessiner

Gitlab : 9_dessiner

Avec OpenCv il est possible de créer : des lignes, des rectangles, des cercles, des ellipses, des polygones et du texte.

Pour commencer il faut créer un fond noir :

```
img = np.zeros((512,512,3), np.uint8)
```

Puis on va créer une diagonale bleue :

```
img = cv2.line(img,(0,0),(511,511),(255,0,0),5)
```

Puis un rectangle vert dans le coin en haut a droite :

```
img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)
```

Puis un cercle rouge dans le carré vert :

```
img = cv2.circle(img,(447,63), 63, (0,0,255), -1)
```

Puis dessiner une demi-éclipse bleu aux centres :

```
img = cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
```

Puis un polygone jaune :

```
pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)
pts = pts.reshape((-1,1,2))
img = cv2.polylines(img,[pts],True,(0,255,255))
```

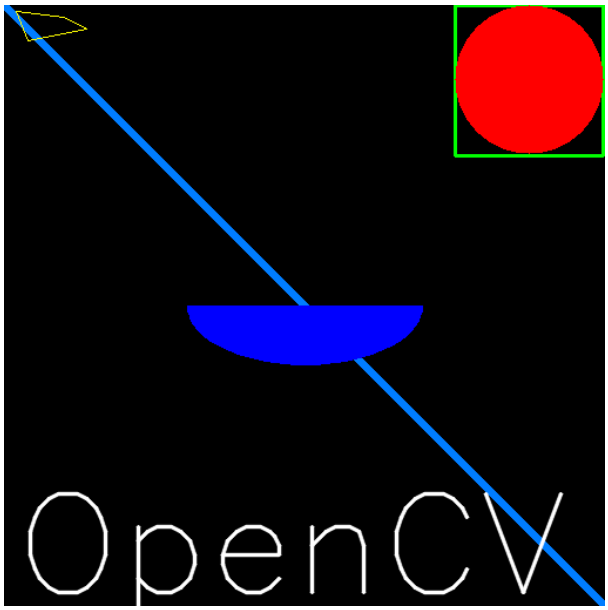
Puis insérez du texte :

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,500), font, 4,(255,255,255),2,cv2.LINE_AA)
```

12.1 - Code

```
#drawing.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque opencv et numpy
import cv2
import numpy as np
#création d'un background noir
img = np.zeros((512,512,3), np.uint8)
#création d'une diagonale bleu
img = cv2.line(img,(0,0),(511,511),(255,127,0),5)
#création d'un rectangle vert dans le coin gauche
img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),2)
#création d'un cercle rougle dans le carré
img = cv2.circle(img,(447,63),63,(0,0,255),-1)
#création d'une demi-elipse bleu aux centre
img = cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
#création d'un polygone
pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)
pts = pts.reshape((-1,1,2))
img = cv2.polylines(img,[pts],True,(0,255,255))
#insertion du texte "OpenCV"
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,500),font,4,(255,255,255),2,cv2.LINE_AA)
#affichage de img
cv2.imshow('test',img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

12.2 - Exemple



13.0 - Harris corner detetcion

Gitlab : 10_harris_corner_detection

Cette méthode permet de détecter des coins dans une image.

la ligne :
img[dst>0.04*dst.max()]=[0,0,255]
Permet le réglage, du seuil (en rapport avec votre image) et la couleur des points.

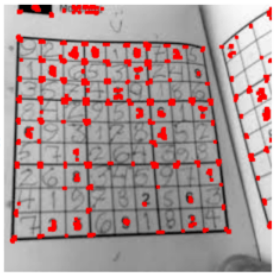
13.1 - Code

```
#corner_detection.py / 18.11.2020 / Robin Forestier
#Harris corner detection
#import de la bibliothèque opencv
import cv2
import numpy as np
#charge l'image minecraft.jpg dans img
img = cv2.imread('sudoku.jpeg')
#convertion de l'image en nuance de gris
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
#detection des coins
dst = cv2.cornerHarris(gray,2,3,0.04)
dst = cv2.dilate(dst,None)
#choix de la couleur et  la valeurmde seuil
img[dst>0.04*dst.max()]=[0,0,255]
#affiche img / titre = mon image
cv2.imshow('mon image', img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

13.2 - Exemple

Original :

Detection :



14.0 - shi tomasi corner detection

Gitlab : 11_shi_tomasi_corner_detection

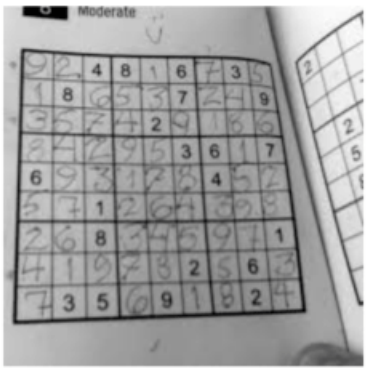
Shi-tomasi corner detection est une méthode plus récente et plus optimisée pour détecter les coins dans une image.

14.1 - Code

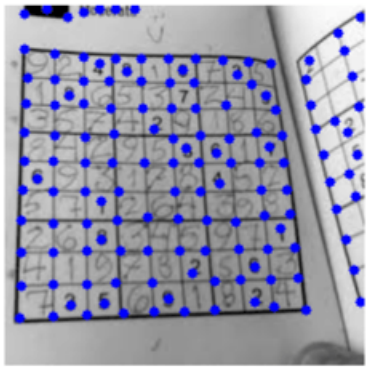
```
#corner_detection.py / 18.11.2020 / Robin Forestier
#Shi-Tomasi Corner Detection
#import de la bibliothèque opencv et numpy
import cv2
import numpy as np
#charge l'image minecraft.jpg dans img
img = cv2.imread('sudoku.jpeg')
#convertion de l'image en nuance de gris
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#détections des coins (140 => nbr de coins)
corner = cv2.goodFeaturesToTrack(gray,140,0.01,10)
corner = np.int0(corner)
#affichage des coins
for i in corner:
    x,y = i.ravel()
    cv2.circle(img,(x,y),3,255,-1)
#affiche img / titre = mon image
cv2.imshow('mon image', img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

14.2 - Exemple

Original :



Detection :



(Pour une meilleure qualité : Gitlab - Reconnaissance visuel)

15.0 - Tracking vidéo

Gitlab : 12_tracking_video

Pour le tracking vidéo je vais vous présenter deux méthodes : meanshift et camshift. Le but serra de repérer une personne et de l'entourer avec un carré bleu. Les deux méthode le permette mais, camshift permet une rotation et une modification de taille du rectangle et donc un meilleur suivi.

15.1 - meanshift

Le principe de meanshift est, imaginer avoir un cercle et plein de petite tache. Le but serra de placer le cercle ou il y a le plus de petite tache. Pour ce faire on récupère l'image de notre caméra, on la converti en HSV, réalise un mask avec une range de couleur. Puis dans une boucle while, on recherche la nouvelle position et on affiche le rectangle.

15.1.1 - Code

```
#meanshift.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque OpenCv
import cv2
import numpy as np
#sélection de la caméra
cap = cv2.VideoCapture(0)
#première image
ret, frame = cap.read()
#première position de la fenetre
r,h,c,w = 250,90,400,125
track_window = (c,r,w,h)
#set up the ROI for traking
roi = frame[r:r+h,c:c+w]
hsv_roi = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi,np.array((0.,60.,32.)),np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,10,1)
#boucle infinie
while(True):
  #enregistrement de l'image dans frame
   ret, frame = cap.read()
  if ret == True:
      hsv= cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
      dst= cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

     #aplique meanshift pour detecter la nouvelle localisation
      ret, track_window= cv2.meanShift(dst, track_window, term_crit)

     #dessine sur l'image
      x,y,w,h= track_window
      img2= cv2.rectangle(frame,(x,y),(x+w,y+h),255,2)

     #affiche
      cv2imshow('img2',img2)
      cv2imshow('f',dst)

     #esc to quit
      k= cv2.waitKey(60) & 0xff
     if k == 27:
       break

  else:
     break
#décharge de la mémoire
#fermeture de toutes les fenêtres
cap.release()
cv2.destroyAllWindows()
```

15.2 - camshift

Cette méthode permet que le rectangle puisse modifier sa taille est son orientation. Pour le faire, on vient réaliser une ellipse sur la zone de pixel.

15.2.1 - Code

```
#camshift.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque OpenCv
import cv2
import numpy as np
#sélection de la caméra
cap = cv2.VideoCapture(0)
#première image
ret, frame = cap.read()
#première position de la fenetre
r,h,c,w = 250,90,400,125
track_window = (c,r,w,h)
#set up the ROI for tracking
roi = frame[r:r+h,c:c+w]
hsv_roi = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi,np.array((-10.,40.,150.)),np.array((20.,90.,190.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,10,1)
#boucle infinie
while(True):
  #enregistrement de l'image dans frame
   ret, frame = cap.read()
  if ret == True:
       hsv= cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
       dst= cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

      #aplique CamShift pour detecter la nouvelle localisation
       ret, track_window= cv2.CamShift(dst, track_window, term_crit)

      #dessine le polygone
       pts= cv2.boxPoints(ret)
       pts= np.int0(pts)
       img2= cv2.polylines(frame,[pts],True,255,2)

      #affiche
       cv2imshow('img2',img2)
       cv2imshow('f',dst)

       k= cv2.waitKey(60) & 0xff
      if k == 27:
        break

  else:
     break
#décharge de la mémoire
#fermeture de toutes les fenêtres
cap.release()
cv2.destroyAllWindows()
```

16.0 - Detection de contour

Gitlab : 13_edge_detection

Pour la détection de contour et le dessin de contour on vas utiliser les méthodes : **cv2.findContours()**, **cv2.drawContours().**

La détection de contour et très utile pour la détection et la reconnaissance de formes. Pour une meilleure utilisation utilisée des images binaire. Donc avant de réaliser une détection de contours, appliquer un seuillage ou un lissage gaussien.

Pour le test je vais réaliser le contour de cette éclaire :
Cette image et très facile a traité car l'éclaire est clairement définie dans l'image

Pour détecter les contours de notre éclaire, on va commencer par l'importer, puis s'assurer qu'il soit en nuance de gris. On lui appliquera un seuillage et on vient détecter les contours avec **cv2.findContours().**

16.1 - Code

```
#contour.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import cv2
import numpy as np
#charge l'image
img = cv2.imread('lightning.png')
#passe image en gris
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#réalisation d'un seuillage
ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)
#recherche des contours
contours, hierarchy = cv2.findContours(thresh,1, 2)
cnt = contours[0]
```

Il reste plus qu'as afficher : un rectangle, un crecle,etc... autour de notre éclaire et d'afficher l'image.

```
#Rectangle
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
cv2.drawContours(img,[box],0,(0,255,0),2)
#Cercle
(x,y),radius = cv2.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
cv2.circle(img,center,radius,(0,255,0),2)
#Ellipse
ellipse = cv2.fitEllipse(cnt)
```

```
cv2.ellipse(img,ellipse,(0,255,0),2)
#Trait
rows,cols = img.shape[:2]
[vx,vy,x,y] = cv2.fitLine(cnt, cv2.DIST_L2,0,0.01,0.01)
lefty = int((-x*vy/vx) + y)
righty = int(((cols-x)*vy/vx)+y)
cv2.line(img,(cols-1,righty),(0,lefty),(0,255,0),2)
#Affichage
x,y,w,h = cv2.boundingRect(cnt)
aspect_ratio = float(w)/h
print("ratio : ",aspect_ratio)
(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
print("angle  : ",angle)
#affiche my_image / titre = mon image
cv2.imshow('mon image', img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

16.2 - Exemple

Figure 6 - image de base

Figure 7 - contour avec un cercle

Figure 8 - contour avec un rectangle

Figure 9 - ligne central

Figure 10 - contour avec une ellipse

17.0 - Line detection

Gitlab : 14_line_detection

Pour la détection de ligne on va utiliser « Hough Line Transform ». Il existe une modification de cette première méthode permettant de mieux détecter la fin et le début des lignes.

Pour la première méthode il faudra importer votre image, la passer en gris, réaliser une détection de contour Canny puis on utilisera la fonction : cv2.HoughLines() pour détecter les lignes.

17.0.1 - Code

```
#HoughLines detection
#line_detect_1.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import cv2
import numpy as np
#charge l'image
img = cv2.imread('lightning.png')
#passe image en gris
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)
lines = cv2.HoughLines(edges,1,np.pi/180,40)
for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)
#affiche my_image / titre = mon image
cv2.imshow('mon image', img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

17.0.2 - Exemple

Figure 11 – image de base

Figure 12 – résultat line detection

17.1 - Line detction 2

La deuxième méthode et une méthode mieux optimisée basée sur celle vue précédemment.

Pour cela on utilisera la fonction : cv2.HoughLinesP**()** qui nous demandera deux nouveaux arguments.

Elle a pour avantage de fortement réduire les instructions a l'intérieur de notre boucle for et de nous donner un contour plus réel.

17.1.1 - Code

```
#HoughLinesP detection
#line_detect_2.py / 18.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import cv2
import numpy as np
#charge l'image
img = cv2.imread('lightning.png')
#passe image en gris
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)
lines = cv2.HoughLinesP(edges,1,np.pi/180,10,minLineLength=10,maxLineGap=10)
for line in lines:
    x1,y1,x2,y2 = line[0]

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)
#affiche my_image / titre = mon image
cv2.imshow('mon image', img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

17.1.2 - Exemple

Figure 13 - image de base

Figure 14 – image avec line detection

18.0 - Circle detection

Gitlab : 15_circle_detection

Pour détecter des cercles on va utiliser « Hough Circle Transform ».
On utilisera la fonction : cv2.HoughCircles**()** avec 8 paramètres.

1. Image
2. Methode (dans notre cas : cv2.HOUGH_GRADIENT)
3. dp (raport de résolution de l'image 1 : 1)
4. minDist (distance minimum entre les centre des cercles)
5. param1 (valeur pour Canny detection)
6. param2 (plus il est petit plus il y aura de faux cercle)
7. minRadius (rayon minimum d'un cercle)
8. maxRadius (rayon maximum d'un cercle / si <= 0, utilise la dimension maximal de l'image)

18.1 - Code

```
#Circle detection
#circle_detect.py / 23.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import cv2
import numpy as np
#charge l'image
img = cv2.imread('rond.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray,5)
circles = cv2.HoughCircles(gray,cv2.HOUGH_GRADIENT,1,50,
                    param1=50,param2=30,minRadius=0,maxRadius=150)
#vérification si il y a un cercles
if circles is not None :
    circles = np.uint16(np.around(circles))
  #affiches les cercles et leur centre
  for i in circles[0,:]:
      cv2circle(img,(i[0],i[1]),i[2],(0,255,255),2)
      cv2circle(img,(i[0],i[1]),2,(0,0,0),3)
#affiche l'image
cv2.imshow('mon image', img)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

18.2 - Exemple

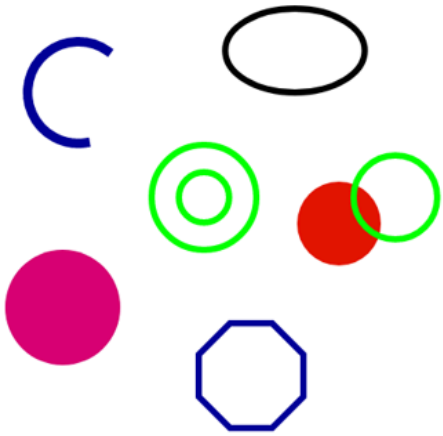


Figure 15 – image de base

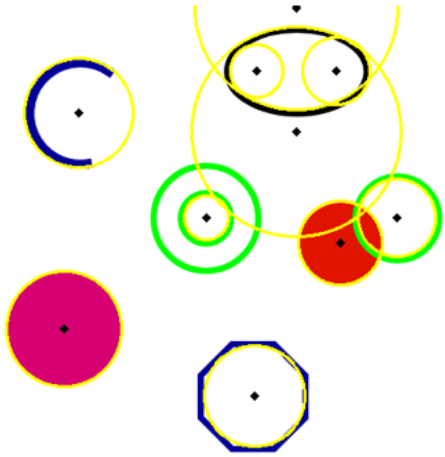


Figure 16 – résultat

J'ai créé une image de base « rond.png » pour tester mon programme. Les cercles jaunes sont les cercles obtenus. Je remarque que la détection de deux cercles avec le même centre n'est pas possible, pour l'ellipse on voit qu'il détecte deux cercles interne et deux cercles externe et il détecte parfaitement mon octogone.

19.0 - Segmentation d'image

Gitlab : 16_image_segmentation

La segmentation d'image est utilisée pour séparer plusieurs objets aux seins d'une image.

Pour le code, il faudra commencer par un seuillage de l'image, puis une suppression du bruit. On vient ensuite une fois dilater les formes et réduire pour s'assurer la suppression de tous les bruit dans l'image.

19.1 - Code

```
#Segmentation d'image
#segmentation.py / 23.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import cv2
import numpy as np
#charge l'image
img = cv2.imread('coins.jpg')
#convertion en gris
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#seuillage de l'image
ret,thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
#suppretion du bruit
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel,iterations=2)
#back ground
sure_bg = cv2.dilate(opening,kernel,iterations=3)
#first plan
dist_tranform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_tranform,0.7*dist_tranform.max(),255,0)
#unknow region
sure_fg = np.uint8(sure_fg)
```

```
unknow = cv2.subtract(sure_bg,sure_fg)
ret, markers = cv2.connectedComponents(sure_fg)
#+ 1 pour que background oit 1 et pas 0
markers += 1
#markers unknow = 0
markers[unknow==255] = 0
#apply watersheld
markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]
#affiche l'image
cv2.imshow('mon image', opening)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

19.2 - Exemple

Figure 17- segemntation

20.0 - Detection de formes

Gitlab : 17_shape_detection

Pour la détection de formes, on réalise une détection de contours, puis on réaliser une approximation du nombre de contour puis tester et l'afficher.

20.1 - Code

```
#Shape detect
#shape_detect.py / 25.11.2020 / Robin Forestier
#import de la bibliothèque opencv
import numpy as np
import cv2
#charge l'image robin.png dans my_image
img = cv2.imread('forme.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray,250,255,cv2.THRESH_BINARY_INV)
contours, h = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    perimetre = cv2.arcLength(cnt,True)
    approx = cv2.approxPolyDP(cnt,0.01*perimetre,True)

    M = cv2.moments(cnt)
  if M["m00"] != 0:
       cx= int(M["m10"] / M["m00"])
       cy= int(M["m01"] / M["m00"])
  else:
       cx,cy= 0,0
   cv2.drawContours(img,[cnt],-1,(0,255,0),2)
  if len(approx)==3:
       shape= "triangle"
  elif len(approx) == 4:
       (x,y,w,h)= cv2.boundingRect(approx)
       ratio= w/float(h)
      if ratio >= 0.95 and ratio <= 1.05:
            shape= "carre"
      else:
            shape= "rectangle"
  elif len(approx) == 5:
         shape= "pentagone"
  elif len(approx) == 6:
       shape= "hexagone"
  else:
       shape= "cercle"
   cv2.putText(img,shape,(cx,cy),cv2.FONT_HERSHEY_SIMPLEX,0.5,(255,0,0),2)
#affichage du résultat
cv2.imshow('image', img)
cv2.imshow('iage', thresh)
#attendre qu'une touch soit pressée
cv2.waitKey(0)
#ferme toutes les fenêtre
cv2.destroyAllWindows()
```

20.2 - Exemple

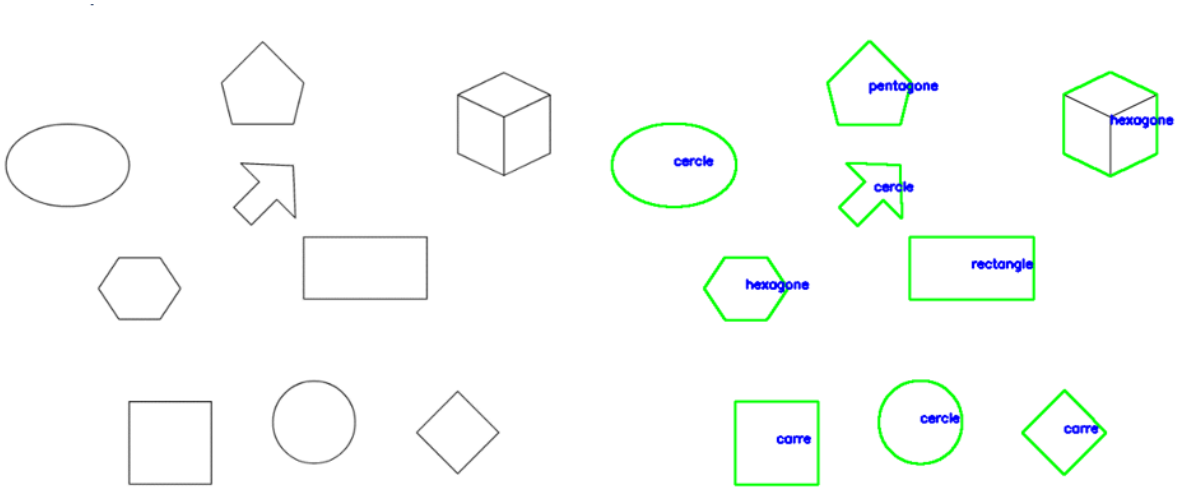


Figure 18 - détection de formes

21.0 - QR code

Gitlab : 18_QR

Le but serra de décoder des code QR.

Pour le faire on utilisera la classe QRCodeDetector.

21.1 - Code

```
#QR
#qr.py / 25.11.2020 / Robin Forestier
#Programme de test de lecture de qr code
#import de la bibliothèque OpenCv et numpy
import cv2
import numpy as np
#sélection de la caméra
cap = cv2.VideoCapture(0)
#boucle infinie
while True:
  #enregistrement de l'image dans img
   ret, img = cap.read()

  #creation de l'objet qr CodeDetector
   qrCodeDetector = cv2.QRCodeDetector()

  #détection du qr code
   decodedText, points, straight_qrcode = qrCodeDetector.detectAndDecode(img)
  #test si il y a un qr code
  if points is not None:
     #affichage d'un carré bleu sur le qr code
      nrOfPoints= len(points)
      ret= cv2.minAreaRect(points)
      pts= cv2.boxPoints(ret)
      pts= np.int0(pts)
      img= cv2.polylines(img,[pts],True,255,2)
```

```
    #affiche le contenu du qr code
    print(decodedText)

    #récupération du qr et enregistrement
    #if straight_qrcode is not None :
       #cv2.imshow('t',straight_qrcode)
       #cv2.imwrite('lastqr.png',straight_qrcode)
  else :
    print("QR not detect")

  #affiche du résultat
   cv2.imshow('image',img)

  #si touche "q" pressée arrêt de la boucle
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break
#décharge de la mémoire
#fermeture de toutes les fenêtres
cap.release()
cv2.destroyAllWindows()
```

En affichant straight_qrcode vous obtiendrez le QR code cadrer et tourner.

22.0 - Feature matching

Gitlab : 19_feature_matching

Le but du feature matching et de retrouver tous les points semblables dans deux images, de les relier pour pouvoir soit, reconnaître un objet ou relier deux images.

Il existe 3 méthodes, en utilisant Shift, Surf ou ORB. ORB et un mélange des deux méthodes permettant une détection plus précise et plus rapide.

Mais elle reste fonctionnelle uniquement dans des cas précis ou dans des conditions parfaites.

22.1 - Code

```
#Feature matching
#ORB_detection.py / 27.01.2021 / Robin Forestier
#import de la bibliothèque opencv et numpy
import cv2
import numpy as np
def ORB_detector(new_image, image_template):
  # Fonction qui compare l'image envoyer avec la template
  # Et retourn le nombre de "matche" trouver entre les deux
   image1 = cv2.cvtColor(new_image, cv2.COLOR_BGR2GRAY)
  # Création d'un ORB detector avec 1000 "points" et un facteur pyramide de 1.2
   orb = cv2.ORB_create(1000, 1.2)
  # Detection des "points" (keypoints) dans l'image
   (kp1, des1) = orb.detectAndCompute(image1, None)
  # Detection des "points" dans la template
   (kp2, des2) = orb.detectAndCompute(image_template, None)
  # Creation d'un matcher
   bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
  # Match
   matches = bf.match(des1,des2)
  # Trit des matches en rapport a la distance
  # Petite distance est meilleur
   matches = sorted(matches, key=lambda val: val.distance)
  return len(matches)
```

```
cap = cv2.VideoCapture(0)
# Image template, image de référence
image_template = cv2.imread('red-bull.jpeg', 0)
while True:
    ret, frame = cap.read()
   # Récupérer le nombre de matches
    matches = ORB_detector(frame, image_template)
   # Afficher le nombre de matches
    output_string = "Matches = " + str(matches)
    cv2.putText(frame, output_string, (50,450), cv2.FONT_HERSHEY_COMPLEX, 2, (250,0,150), 2)
   # Le threshold et le seuille après lequelle on décide,
   # que l'objet se trouva dans l'image
   # Note : Avec 1000 matches, un threshold a 350 serrait égale a min 35% d matches
    threshold = 190
   if matches > threshold:
      # Affiche object found
       cv2putText(frame,'Object Found',(50,50), cv2.FONT_HERSHEY_COMPLEX, 2 ,(0,255,0), 2)

   #affiche image
    cv2.imshow('Object Detector using ORB', frame)

   if cv2.waitKey(1) == 13: #13 is the Enter Key
      break
cap.release()
cv2.destroyAllWindows()
```

23.0 - Soustraction d'image

Gitlab : 20_soustraction

La soustraction d'image come son nom l'indique, serra de soustraire deux images pour en faire ressortir les différences. Dans mon cas mon but a été, de trouver un objet en mouvement, de le détourer et de l'extraire de l'image pour un reconnaissance futur.

Pour cela je procède en 4 étapes :

1. Soustraction de ma dernière image avec la précédente.
2. Réalisation du lissage et d'une dilatation
3. Trouver la plus grande forme
4. L'isolé

23.1 - Code

```
#Feature matching
#soustraction.py / 19.01.2021 / Birthday... / Robin Forestier
#Programme de spustraction d'image et de récupération d'information
#import de la bibliothèque OpenCv et numpy
import cv2
import numpy as np
#sélection de la caméra
cap = cv2.VideoCapture(0)
#capture la première image
ret, frame = cap.read()
s_frame = frame
#transformation en gris et dilatation
s_gray = cv2.cvtColor(s_frame, cv2.COLOR_BGR2GRAY)
s_gray = cv2.dilate(s_gray, None, iterations=2)
#boucle infinie
```

```
while(True):
  #enregistrement de l'image dans frame
   ret, frame = cap.read()
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

  #soustraction des deux image (en gris)
   res =cv2.subtract(s_gray, gray,dst=None,dtype=None)
  #enregistrement de la dernoère image
   s_gray = gray

  #dans l'image soustraitem, réalisation  d'un lissage
   res = cv2.threshold(res, 20, 255, cv2.THRESH_BINARY)[1]
   res = cv2.dilate(res, None, iterations=10)
  #détection des contours
   cnts, hierarchy = cv2.findContours(res, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

   maxy = 100
  #recherche du contour avec le plus grand périmètre
  for c in cnts:
      perimetre= cv2.arcLength(c, True)
     if maxy <= perimetre :
         new_cnts= c
         maxy= perimetre

  #création d'une image avec la l'objet trouver et dessin du rectangle vert
   x,y,w,h = cv2.boundingRect(new_cnts)
   cut_img = frame[y:y+h, x:x+w]
   cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0),1)

  #resize de l'image de l'objet trouver
   aff_cut_img = cv2.resize(cut_img,(640,480),interpolation=cv2.INTER_AREA)

  #affichage des images
   cv2.imshow('caméra',aff_cut_img) #image zommée
   cv2.imshow('frame',frame) #image complète
   cv2.imshow('sus',res) #zonne de pixel en mouvement

  #si touche "q" pressée arrêt de la boucle
  if cv2.waitKey(1) & 0xFF == ord('q'):
     break
#décharge de la mémoire
#fermeture de toutes les fenêtres
cap.release()
cv2.destroyAllWindows()
```

24.0 - Tic Tac Toe

Gitlab : x_tick_tac_toe

Pour choisir le mode de jeux ouvrer le ficier ticTacToe.py et changer les constantes,
RANDOM_PLAYER, **RANDOM_HARD**, **AI_PLAYER** pour selectionner le mode de jeu.
Celle a True serra celectionnée (si aucune => 2 player).
Vous pouvez aussi choisir que l'ordi joue en premier en changeant *self.player = 2*

24.1 - But

Le but est de réaliser le développement d'un Tic Tac Toe. La detection du jeu ce fait à l'aide d'une caméra (raspberry pi cam). L'ajout d'une "AI" pour ensuite peutêtre l'implementer sur une machine qui dessine.

24.2 - Méthodologie

1. Détection de la grille.

Pour que ma detection fonctionne je dois utiliser un quadrillage fermé. PDF du quadrillage

Pour la detection de a grille, j'ai commencé par réaliser un threshold et des transformation de base pour isolé les traits noir dans mon image.
J'utilise en suite la fonction cv2.findContours (explication sous 16.0 - Detection de contour) dans ma fonction **detect_grid.**

```
def detect_grid(self, frame):
    self.grid.clear()
    #convertion de l'image en nuance de gris
     gray= cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #egalisation de l'histogramme
     gray= cv2.equalizeHist(gray)
    #threshold
     _, thresh= cv2.threshold(gray, 10, 255, cv2.THRESH_BINARY)
    #sup noise
     kernel= np.ones((5,5),np.uint8)
     opening= cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
    #detect contour
     contours, hierarchy= cv2.findContours(opening, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
     hierarchy= hierarchy[0]
    for i in zip(contours, hierarchy):
        currentContours= i[0]
        currentHierarchy= i[1]
        x, y, w, h= cv2.boundingRect(currentContours)
       #contour intern
       if currentHierarchy[0] > 0 or (currentHierarchy[0] == -1 and currentHierarchy[1] > 0):
          #cv2.putText(frame, str(currentHierarchy), (x, y + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,0),
1) #show hierarchy
            cv2.rectangle(frame, (x,y), (x+w,y+h),(255,0,0),2)
          self.grid.append([x,y,w,h])
    return frame, self.grid
```

Avec la hiérarchie des contours, je récupère les coordonnées de l'intérieurs des 9 carrés.

La fonction **sort_grid** trille les 9 cases, en commençant par la coordonnée Y puis 3 par 3 la coordonné X.

```
def sort_grid(self, frame, grid):
    self.grid = sorted(self.grid, key=lambda l:l[1])
    self.grid[0:3] = sorted(self.grid[0:3], key=lambda l:l[0])
    self.grid[3:6] = sorted(self.grid[3:6], key=lambda l:l[0])
    self.grid[6:9] = sorted(self.grid[6:9], key=lambda l:l[0])
    return frame, self.grid
```

2. Dessin du rond et de la croix.

Pour le dessin du rond j'avais besoin de 3 valeurs impotantes. Les 2 coordonnées X et Y du centre du cercle et son rayon.

La fonction **draw_rond** dans ticTacToe.py calcule le tout et le dessine sur l'image.

```
def draw_rond(self, frame, loc, color, thinkness):
        x,y,w,h= loc
        center_x= int((x + (x + w)) / 2)
        center_y= int((y + (y + h)) / 2)
      if y + h < x + w :
          radius= int(((y + h) - y) / 2)
      else :
          radius= int(((x + w) - x) / 2)
      if (radius - 20) <= 0 : radius = 20
       frame= cv2.circle(frame, (center_x, center_y) , radius - 20, color, thinkness)
      return frame
```

Pour la croix je viens simplement relier les coin opposé de mon carré par deux traits.

```
def draw_croix(self, frame, loc, color, thinkness):
        x, y, w, h= loc
        frame= cv2.line(frame, (x,y), (x + w, y + h), color, thinkness)
        frame= cv2.line(frame, (x,y + h), (x + w,y), color, thinkness)
      return frame
```

Puis la fonction **draw_all** viens dessiner toutes les formes sur l'image.

3. Détection du rond.

Dans la fonction **detect_shape**, on viens pour chaque case non occupée regardé si on y trouve un rond puis si on trouve une croix.

J'utilise ensuite **HoughCircles** (18.0 - Circle detection) pour trouver les cercles.

```
circles = cv2.HoughCircles(roiG, cv2.HOUGH_GRADIENT, 1, 50,
                           param1=50, param2=30, minRadius=20, maxRadius=300)
  if circles is not None:
     self.grid_play[i][j] = self.player
      frame= self.dessine(frame, grid, c)
```

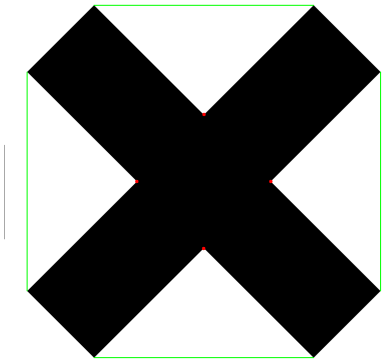
4. Détection de la croix

Pour détecter la croix j'ai réutilisé la fonction **findContours** et j'ai ensuite cherché si la forme était convexe et si elle a "4 défauts de convexes".

```
roi = frame[y:y+h, x:x+w]
roiG = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
gray = cv2.equalizeHist(roiG)
_, th = cv2.threshold(roiG, 30, 255,cv2.THRESH_BINARY_INV)
contours,hierarchy = cv2.findContours(th,2,1)
if len(contours) > 0:
    cnt = max(contours, key = cv2.contourArea)
    hull = cv2.convexHull(cnt,returnPoints = False)
    defects = cv2.convexityDefects(cnt,hull)
  if defects is not None:
       n_dot= 0
     for x in range(defects.shape[0]):
          s,e,f,d= defects[x,0]
          start= tuple(cnt[s][0])
          end= tuple(cnt[e][0])
          far= tuple(cnt[f][0])
```

```
#cv2.line(roi,start,end,[0,255,0],2)
#cv2.circle(roi,far,5,[0,0,255],-1)
if far is not None: n_dot += 1
if n_dot == 4:
    self.grid_play[i][j] = self.player
      frame= self.dessine(frame, grid, c)
    #print(self.grid_play)
    self.shape_detect = 1
```

Exemple :



On voit bien les 4 points rouges que nous cherchons.

5. AI et random

Pour que le jeux soit plus intéressant, j'ai souhaité ajouté un mode 1 joueur.
Pour cela j'ai utiliser l'algorithme **Minimax**.

Minimax | Wikipedia : https://en.wikipedia.org/wiki/Minimax

Minimax | Geek for Geek : https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

Minimax | The coding Train : https://thecodingtrain.com/CodingChallenges/154-tic-tac-toe-minimax.html

```
Pour trouver le meilleur endroit ou l'ordinateur doit placer sa croix ou son rond,
il réalise un arbre de possibilité et se dirige dans la branche ou il a le plus de chance de gagner.
```

L'algorithme ce trouve dans le fichier **minimax.py**.
code : (minimax.py | best_move())

Après plusieur essay, je me suis aperçu que je ne pouvais pas gagner contre cette IA.
Pour palier a ce problème, je réalisé un second mode nommé randome qui vient posé sa forme aléatoirement sur la grille.

En ajoutant le faite que le randome vérifie si il peut perdre ou gagner, j'ai obtenu le résultat que je souhaitait.
code : (ticTacToe.py | rand_player() | rand_check_win())

Robin Forestier 2020/2021