Livrable fait par Amal, Steeven, Luca et Justine

Convoyeur-Trieur

Introduction

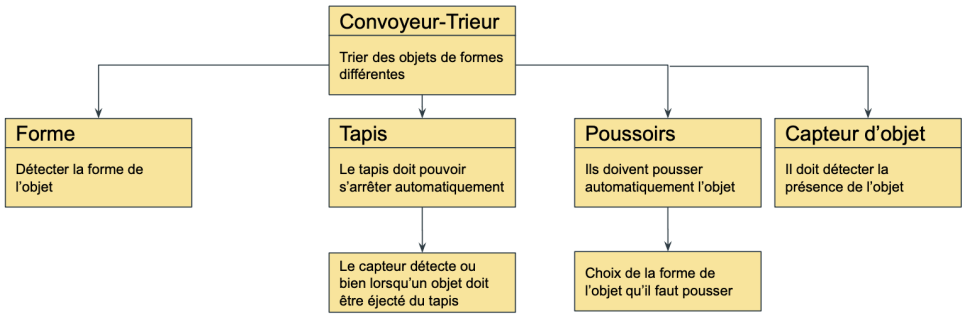
L'objectif de ce projet est de créer un système permettant de trier des objets en fonction de leur forme. En effet, en fonction de leur forme, l'objet à trier va être déplacé à des positions différentes. Cette catégorie de système se retrouve notamment dans le domaine de l'industrie de production. En effet, cela permet d'optimiser les moyens de production en augmentant la vitesse de production.

Objectifs :

Pour réaliser ce projet, il nous a fallu remplir plusieurs objectifs :

- Détecter la forme de l'objet
- Pouvoir arrêter automatiquement le tapis lorsque le capteur détecte un objet ou lorsqu'un objet doit être éjecté du tapis
- Pouvoir pousser l'objet automatiquement grâce aux poussoirs
- Détecter la présence de l'objet à l'aide du capteur

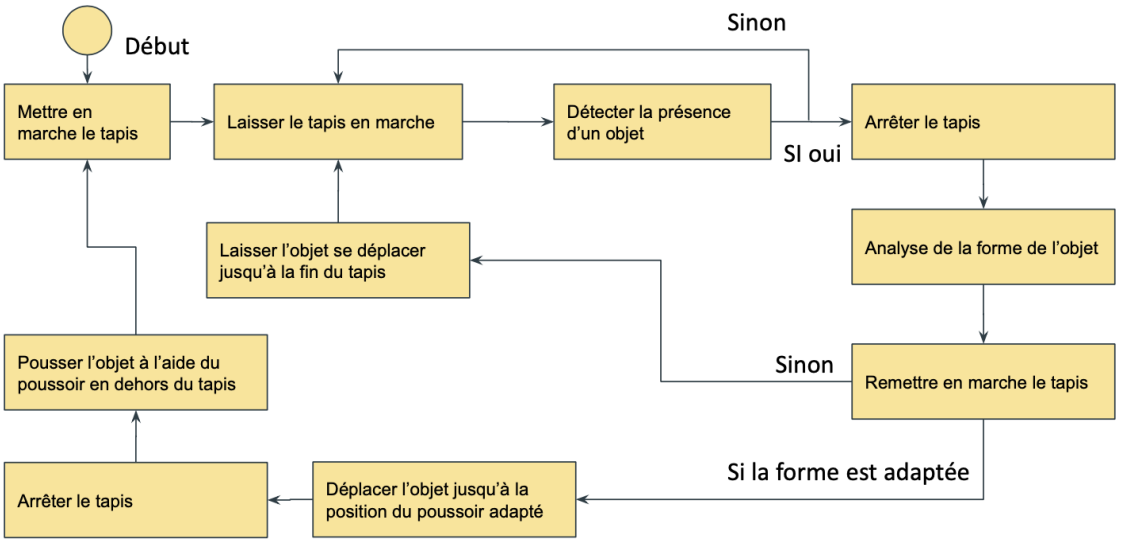
Schéma d'exigence



Fonctionnement global :

On met d'abord en marche le tapis. Ensuite, grâce à un capteur de présence, on détecte s'il y a bien un objet. Si c'est le cas, le tapis s'arrête. A l'aide d'une caméra, on détecte la forme de l'objet. Si la forme correspond à celle voulue, l'objet est déplacé jusqu'à la position du poussoir adapté. Ainsi, il est poussé hors du tapis par celui-ci. Sinon l'objet se déplace jusqu'à la fin du tapis.

Schéma de fonctionnement :



Cahier des charges :

Ce cahier des charges a été établi au début du projet. Il nous a permis de nous donner un ordre de grandeur des critères attendus. Certains critères comme le temps d'exposition n'ont pas pu être vérifiés. Le critère de vitesse du poussoir (bras) a cependant été pris en compte.

Fonction	Expression	Critères	Niveaux
FP1	Permettre de trier des objets		
FC1	Déplacer le bras	t_{bras} : temps aller-retour poussoir d_{bras} : distance maximal de translation v_{bras} : vitesse rotation moteur	t_{bras} = 2 s d_{bras} = 9,5 cm v_{bras} ≈ 5 cm/s
FC2	Analyser l'objet	t_{exp} : temps d'exposition caméra t_{det} : temps de détection Couleur de l'objet	t_{exp} = 13 ms t_{det} = 10 ms
FC3	Déplacer l'objet	v_{mot} : vitesse de rotation du moteur v_{tap} : vitesse du tapis d_{obj} : distance caméra-bras	v_{tap} < 120 mm/s d_{obj} < 15 cm
FC4	Acquérir image sur ordinateur		
FC5	Détecter l'objet	t_{acq} : temps d'acquisition du détecteur	t_{acq} < 1s

Compte rendu :

1ère séance :

On a fait le Cahier des charges, le schéma de principe et effectuer la répartition des tâches et le code du servomoteur (bras).
Distribution des Tâches à faire :

- Steeven et Justine → code caméra (détection + capture d'écran)
- Luca et Amal → faire tourner le tapis

compte rendu séance du 21/03

Pour Justine et Steeven :
On a réussi à détecter les formes et couleurs d'une image donnée sur python, à faire une capture d'écran sur python

Pour Luca et Amal :
On a réussi à écrire le code avec le moteur pas à pas et on l'a testé sur le moteur.
On a testé le code avec le moteur du tapis, ça a marché mais il faut changer les paramètres de vitesse(temps pour le clock) pour optimiser la vitesse et les vibrations.

Objectif de la prochaine séance :
Il faut utiliser cette capture pour détecter les formes et couleurs et automatiser le refresh et faire le lien avec le moteur bras et tapis roulant

compte rendu séance du 28/03

Pour Steeven et Justine :
On a réussi à relier détection de forme et capture d'écran. On a commencé le lien entre la carte nucléo et python. C'est à dire envoyer la forme (une lettre) à la carte nucléo

Pour Amal et Luca:
On a trouvé la valeur optimale de la vitesse. On a créé la fonction marche, arrêt, poussoirs et on travaille sur le code principal du main. On a essayé de mettre des conditions d'arrêt sur la distance (ca n'a pas marché). On essaie alors avec le module ticker

Objectif de la prochaine séance :
-Réussir le code sur keil studio pour lire la forme et donner l'ordre au moteur.
-Combiner tous les codes sur le main (avec les conditions d'arrêt,camera, tapis et poussoirs)

compte rendu séance du 04/04

Pour Steeven et Justine : Rien a fonctionné

Pour Amal et Luca: On a relié les poussoirs au tapis, on a réussi à mettre le ticker mais finalement on a changé de stratégie en mettant une boucle while.

compte rendu séance du 11/04

justine et Steeven : on a reussi à lié le python avec nucléo et créer un refresh automatique

Amal et Luca : On a relié python et nucléo, ajouté la condition if et créer le code du détecteur

conclusion : Le système marche

Bilan :

Finalement, le projet a abouti. On a réussi à créer un convoyeur qui trie les objets selon leur forme. Cependant, il est limité à un objet à la fois sur le tapis ce qui est contraignant. En réalité, il est nécessaire de pouvoir traiter plusieurs objets à la suite pour optimiser la rapidité de la tâche. Il existe aussi un nombre limité de formes et de poussoirs dans notre code limitant ainsi notre système. Justine et Steeven se sont occupés de la détection de forme. Ils ont commencé par établir le code permettant à partir d'une image de détecter les formes présentes. Ensuite, ils ont créé le code permettant la capture d'image à partir de la caméra de manière automatique. Enfin, ils ont créé le code permettant de faire la liaison entre le code python et la carte Nucléo.
Luca et Amal se sont occupés de l'automatisation du tapis. Ils ont commencé par coder le moteur pour qu'il puisse mettre en marche le tapis. De plus, ils ont codé les poussoirs et le détecteur de présence. Enfin, ils ont synchronisé le tout pour mettre en marche le système.
Finalement, on a appris à se documenter sur bibliothèque de code qu'on ne connaissait pas et à concevoir un code permettant de mener à bien le projet. Ce projet a aussi permis d'avoir une bonne expérience de travail en équipe.

ANNEXE CODE :

Main :

```
#include "mbed.h"

#define     DEBUG   0


PwmOut          convoyor_speed(D3);     // Commande pour la vitesse du
convoyeur
DigitalOut      convoyor_dir(A2);       // Commande pour la direction :
1 sens horaire, 0 sens anti-horaire
DigitalOut      convoyor_half(D13);     // Mode de fonctionnement du
moteur
DigitalOut      convoyor_enable(D4);    // Activation du moteur
DigitalOut      convoyor_reset(A3);     // Réinitialisation des
paramètres

InterruptIn     detecteur(D8); // sortie du capteur
int     piece_detected = 0;

  // 0 si pas de mouvmement, 1 si mouvement

PwmOut servo_mot1(D10); //servo_mot poussoir 1
PwmOut servo_mot2(D9);//poussoir 2
UnbufferedSerial        my_pc(USBTX, USBRX);// Communication série avec
l'ordinateur
char        str_char[128];
void stopConvoyor();
void marcheConvoyor();
void detect_piece();


int main(){
    my_pc.baud(9600);
    char data = '0';

    detecteur.rise(&detect_piece);

    servo_mot1.period_ms(20);           // Initialisation période
    servo_mot1.pulsewidth_us(1000);     // Initialisation en position 0
pour le poussoir 1
    servo_mot2.period_ms(20);           // Initialisation période
    servo_mot2.pulsewidth_us(1000);     // Initialisation en position 0
pour le poussoir 2
```

Tout d'abord, on initialise les entrées et les sorties de la carte Nucléo : le convoyeur, les deux poussoirs et le détecteur.

On lie les données récupérées à la carte nucléo pour pouvoir les utiliser ensuite dans le code python.

On initialise les poussoirs à leurs période et positions initiales avant d'entrer dans la boucle

```
    // Init convoyeur
    conveyor_enable = 1;                                // Commandes à
envoyer pour faire fonctionner le moteur
    conveyor_reset = 1;
    conveyor_half = 0;
    conveyor_dir = 1;                       // Sens horaire (forward)
    conveyor_speed.period_ms(3);                    // Période du signal
PWM initialisée à 3 ms
    // Fin Init convoyeur

    marcheConveyor();

    while(1){
        if (my_pc.readable()){
            my_pc.read(&data,1);
            my_pc.write(&data,1);
        }

    // marcheConveyor();

        sprintf( str_char, "a = %d\r\n", piece_detected);
        if(DEBUG)    my_pc.write(str_char, strlen(str_char));

        if (piece_detected == 1) { // objet détecté

            sprintf( str_char, "P_OK\r\n");
            if(DEBUG)    my_pc.write(str_char, strlen(str_char));
            char temp = '1';
            my_pc.write(&temp,1);
            stopConveyor();
            thread_sleep_for(2000); // temps d'attente caméra
            marcheConveyor();
        }



        if (data=='S')
        {
            sprintf( str_char, "T\r\n");
            if(DEBUG)    my_pc.write(str_char, strlen(str_char));
            thread_sleep_for(7500); // temps d'attente objet-poussoir
            stopConveyor();
```

Ici, on contrôle le moteur pas à pas (convoyeur): sens de rotation, vitesse, période du signal PWM..

Il s'agit ici de la boucle while qu'on a utilisée,elle a besoin d'avoir l'information du capteur, en cas de présence ou pas de l'objet et ensuite on pousse/ou pas l'objet selon le critère de la forme choisie à chaque fois.On utilise aussi les fonction marche convoyeur, arrêt convoyeur et 'thread sleep for' pour mettre un temps d'attente suffisant pour l'acquisition de l'image à l'aide de la caméra ou pour mettre le temps nécessaire d'un aller-retour des poussoirs, selon le besoin.

Si on a la présence d'un objet de la forme 'square' on active le premier poussoir.
Sinon,si la forme détectée est 'rectangle',on active le second poussoir. Si c'est ni l'un ni l'autre, le tapis continue à marcher et on reprend la boucle.

```
            servo_mot1.pulsewidth_us(6000);     // (Angle positif)
sortir le 1er poussoir
            thread_sleep_for(1000);
            servo_mot1.pulsewidth_us(1000);     // (Angle négatif)
rentrer le 1er poussoir$
            piece_detected = 0;
            data = '0';
            marcheConveyor();
        }


        if (data=='R')
        {
            sprintf( str_char, "R\r\n");
            if(DEBUG)    my_pc.write(str_char, strlen(str_char));
            thread_sleep_for(12000); // temps d'attente objet-poussoir
            stopConveyor();
            servo_mot2.pulsewidth_us(6000);    // (Angle positif) sorti
le 2ème poussoir
            thread_sleep_for(1000);
            servo_mot2.pulsewidth_us(1000);    // (Angle négatif)
rentrer le 2ème poussoir
            piece_detected = 0;
            data = '0';
            marcheConveyor();
        }

        /*
        marcheConveyor();
        thread_sleep_for(6000);//évacuer les objets qui restent du
tapis
        stopConveyor();//arret final du tapis
        thread_sleep_for(6000);
        */

        if ((data!='S') && (data!='R')){
            sprintf( str_char, "noRnoT\r\n");
```

La carte nucléo récupère l'information de la forme à partir de la caméra , la présence ou pas de l'objet par le détecteur et on effectue cette liaison entre python et C; d'une part pour déclencher l'acquisition de l'image par la caméra (en cas de présence d'objet détecté), et d'autre part pour récupérer l'information sur la forme, chose qui permet de réaliser les conditions sur les deux poussoirs.

```
}



//void toggle_tapis(void)




void detect_piece(){
    piece_detected = 1;
}

//                              Mettre en marche le
convoyeur                                        //
 void marcheConvoyor()
{
    conveyor_enable = 1;
    conveyor_speed.write(0.5);

}



//                              Arrêter le
convoyeur                                        //

void stopConvoyor()     // Arrête le convoyeur
{ conveyor_enable = 0;                // Désactiver le convoyeur
    conveyor_speed.write(0);            // Rapport cyclique nul (signal
de commande nul)

}
```

Il s'agit ici d'une fonction simple qui sert à faire marcher le convoyeur.

Par contre ici, c'est la fonction qui sert à arrêter le convoyeur.

Programme de la capture d'écran et de la détection de forme :

```python
#   Libraries to import
# Camera
from pyueye import ueye
import camera
# Graphical interface
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import (
    QApplication, QDialog, QMainWindow, QMessageBox, QWidget
)
from PyQt5.uic import loadUi
from PyQt5.QtGui import QPixmap, QImage
from serial import Serial

# Standard
import numpy as np
import matplotlib.pyplot as plt
import cv2
import sys
import matplotlib.image as mpimg
import imutils
import serial.tools.list_ports
import threading as th
#-----------------------------------------------------------------
from detection_couleur_formes import *

ports = serial.tools.list_ports.comports()
# To obtain the list of the communication ports
for port, desc, hwid in sorted(ports):
    print("{}: {}".format(port, desc))
    # To select the port to use
selectPort = input("Select a COM port : ")
print(f"Port Selected : COM{selectPort}")
    # To open the serial communication at a specific baudrate
serNuc = Serial('COM'+str(selectPort), 9600)

appOk = 1
```

D'abord, on importe les bibliothèques utiles pour le code.

Ce Code permet d'importer le code sur la détection de forme

Ici, ce code permet de sélectionner un port pour connecter la caméra.

```python
class MainWindow(QMainWindow):
    '''
    Graphical Interface for IDS Camera control class
    '''

    def __init__(self):
        '''
        Constructor of the MainWindow class

        Returns
        -------
        None.

        '''
        super().__init__(parent=None)

        # Camera
        self.camera = None
        self.max_width = 0
        self.max_height = 0

        # Graphical interface
        loadUi("LEnsE_version1.ui", self)
        self.cameraInfo.setText('LEnsE')
        self.cameraExposureInfo.setText('Exposure : NO CAMERA')
        self.refreshBt.setEnabled(False)
        self.initListCamera()
        self.refreshBt.clicked.connect(self.refreshGraph)
        self.connectBt.clicked.connect(self.connectCamera)
        self.closeBt.clicked.connect(self.closeApp)
        self.refreshListBt.clicked.connect(self.initListCamera)

        # Other variables
        self.frameWidth = self.cameraDisplay.width()
        self.frameHeight = self.cameraDisplay.height()
```

Ce code permet d'avoir un panneau d'affichage où il est possible d'initialiser la caméra. En effet, il faut appuyer sur le bouton "Connect" afin de connecter la caméra grâce à la fonction connectCamera. De plus, il est possible d'appliquer la fonction refreshGraph en cliquant sur "Refresh". Cette fonction est détaillée ultérieurement.
Cette fonction permet aussi de régler la taille de l'affichage de la caméra.

```python
def refreshGraph(self):
    '''
    Refresh the frame with a new image.

    Returns
    -------
    None.

    '''
    if(serNuc.inWaiting()>0):
        info = serNuc.read(1)
    else:
        info = bytes('0', 'ascii')
    info_dec = info.decode()

    if info_dec == '1' :
        print("refresh")
        array = self.camera.get_image()
        X, Y, W, H = self.camera.get_aoi()

        print(f'X = {X} / Y = {Y} / W = {W} / H = {H}')
        self.frame_raw = np.reshape(array,(H, W, -1))
        cv2.imwrite('image.jpg', self.frame_raw)
        #plt.imshow(frame)

        frame = cv2.resize(self.frame_raw, dsize=(self.frameWidth, self.frameHeight), interpolation=cv2.INTER_CUBIC)
        image = QImage(frame, frame.shape[1],frame.shape[0], frame.shape[1], QImage.Format_Grayscale8)
        pmap = QPixmap(image)
        self.cameraDisplay.setPixmap(pmap)
        #mpimg.imsave("resultat.png", image)

        shape_ = detect2('image.jpg')


        serNuc.write(bytes(shape_, 'ascii'))
        info = 0

    S = th.Timer(0.1, self.refreshGraph)
    S.start()
```

La fonction refreshGraph est la fonction principale du code. Elle permet d'envoyer l'information de la forme à la carte nucleo. Tout d'abord, il vérifie si la carte nucléo est bien connectée. Ensuite si elle est connectée, la fonction prend une capture d'écran en la décrivant à l'aide d'une matrice. Ensuite à l'aide de la bibliothèque cv2, la commande cv2.imwrite crée une image à l'aide de l'image capturée (la matrice redimensionnée caractérisant l'image). La fonction renvoie aussi l'image sur l'écran d'affichage de la fonction précédente. Ensuite, à l'aide de la fonction detect2, la fonction détermine les formes représentées dans l'image. Finalement, on a ajouté un timer pour que le programme se relance automatiquement.

```python
def initListCamera(self):
    '''
    Initialize the list of the USB IDS Camera connected to the computer

    Returns
    -------
    Complete the list box of the graphical interface with the list of cameras.

    '''
    self.nb_cam = camera.get_nb_of_cam()
    self.cameraInfo.setText('Cam Nb = '+str(self.nb_cam))
    if(self.nb_cam > 0):
        self.cameraList = camera.get_cam_list()
        self.cameraListCombo.clear()
        for i in range(self.nb_cam):
            cam = self.cameraList[i]
            self.cameraListCombo.addItem(f'{cam[2]} (SN : {cam[1]})')

def connectCamera(self):
    '''
    Event link to the connect button of the GUI.

    Returns
    -------
    None.

    '''
    self.connectBt.setEnabled(False)
    self.refreshBt.setEnabled(True)
    self.connectBt.setText('Connected')

    self.selectedCamera = self.cameraListCombo.currentIndex()
    self.camera = camera.uEyeCamera(self.selectedCamera)

    self.max_width = int(self.camera.get_sensor_max_width())
    self.max_height = int(self.camera.get_sensor_max_height())
    print(self.max_width)
    self.camera.set_exposure(15)
    self.cameraExposureInfo.setText(f'Exposure : {self.camera.get_exposure()} ms')
    self.camera.set_colormode(ueye.IS_CM_MONO8)
    self.camera.set_aoi(0, 0, self.max_width-1, self.max_height-1)
    self.camera.alloc()
    self.camera.capture_video()
    #self.refreshGraph()
```

Cette fonction permet d'initialiser la caméra

Celle ci permet de connecter la caméra

```python
    def closeApp(self):
        self.close()
        self.closeEvent(None)

    def closeEvent(self, event):
        if(self.camera != None):
            self.camera.stop_camera()
        QApplication.quit()




if __name__ == '__main__':

    app = QApplication(sys.argv)
    main = MainWindow()
    main.show()
    sys.exit(app.exec_())
    serNuc.close()
```

Ce code permet de quitter le panneau d’affichage.

Programme de détection de forme

```python
#from pyueye import ueye
import numpy as np
import cv2
#import sys

import imutils

#colors

from webcolors import rgb_to_name,CSS3_HEX_TO_NAMES,hex_to_rgb #python3 -m pip install webcolors
from scipy.spatial import KDTree

def convert_rgb_to_names(rgb_tuple):
   # a dictionary of all the hex and their respective names in css3
    css3_db = CSS3_HEX_TO_NAMES#css3_hex_to_names
    names = []
    rgb_values = []
    for color_hex, color_name in css3_db.items():
        names.append(color_name)
        rgb_values.append(hex_to_rgb(color_hex))

    kdt_db = KDTree(rgb_values)
    distance, index = kdt_db.query(rgb_tuple)
    return names[index]
```

On importe les bibliothèques utiles. La fonction convert_rgb_to_names permet de détecter la couleur de l'objet. Or ici, on s'intéresse qu'à la forme puisque nous étions munis que d'une caméra monochromatique.

```python
class ShapeDetector:
    def __init__(self):
        pass
    def detect(self, c):
        # initialize the shape name and approximate the contour
        shape = "unidentified"
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.04 * peri, True)
        # if the shape is a triangle, it will have 3 vertices
        if len(approx) == 3:
            shape = "T"
        # if the shape has 4 vertices, it is either a square or
        # a rectangle
        elif len(approx) == 4:
            # compute the bounding box of the contour and use the
            # bounding box to compute the aspect ratio
            (x, y, w, h) = cv2.boundingRect(approx)
            ar = w / float(h)
            # a square will have an aspect ratio that is approximately
            # equal to one, otherwise, the shape is a rectangle
            shape = "S" if ar >= 0.95 and ar <= 1.05 else "R"
        # if the shape is a pentagon, it will have 5 vertices
        elif len(approx) == 5:
            shape = "P"
        elif len(approx) == 6:
            shape = "H"
        elif len(approx) == 10 or len(approx) == 12:
            shape = "E"
        # otherwise, we assume the shape is a circle
        else:
            shape = "C"
        # return the name of the shape
        return shape
```

Ce code permet de détecter la forme de l'objet. En effet, l afonction detect commence par délimiter l'objet trouvé. Il repasse sur le contours de l'objet détecté. Ensuite, il compte le nombre de côtés afin de déterminer la forme. Par exemple, si l'objet possède 5 cotés, la fonction va attribuer à cet objet la lettre P puisqu'il correspond à un pentagone. Il va aussi faire la différence entre un rectangle et un carré.

```python
def detect2(name_image):
    # load the image and resize it to a smaller factor so that
    # the shapes can be approximated better
    image = cv2.imread(name_image)
    resized = imutils.resize(image, width=300)
    ratio = image.shape[0] / float(resized.shape[0])

    # convert the resized image to grayscale, blur it slightly,
    # and threshold it
    gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)[1]

    # find contours in the thresholded image and initialize the
    # shape detector
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    sd = ShapeDetector()
    shape = 'v'
    # loop over the contours
    for c in cnts:
        # compute the center of the contour
        M = cv2.moments(c)
        cX = int((M["m10"] / M["m00"]) * ratio)
        cY = int((M["m01"] / M["m00"]) * ratio)

        #detect shape from contour
        shape = sd.detect(c)
        ##########################################################
        print(shape)

        # resize the contour
        c = c.astype("float")
        c *= ratio
        c = c.astype("int")
        cv2.drawContours(image, [c], -1, (0, 255, 0), 2)

        #draw contour with mask
        mask = np.zeros(image.shape[:2], np.uint8)
        cv2.drawContours(mask, [c], -1, 255, -1)
        #img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

       # Convert to RGB and get color name
        imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        mean=cv2.mean(imgRGB, mask=mask)[:3]
        named_color = convert_rgb_to_names(mean)

        #get complementary color for text
        mean2 = (255-mean[0],255-mean[1],255-mean[2])

        #display shape name and color
        objLbl=shape+" {}".format(named_color)
        textSize = cv2.getTextSize(objLbl,cv2.FONT_HERSHEY_SIMPLEX,0.5,2)[0]
        cv2.putText(image, objLbl, (int(cX-textSize[0]/2),int(cY+textSize[1]/2)), cv2.FONT_HERSHEY_SIMPLEX,0.5,mean2, 2)

        #show image
        #cv2.imshow("Image", image)
        cv2.imwrite('image_shape.jpg', image)
        #cv2.waitKey(0)
    #cv2.waitKey(0)
    return shape


if __name__ == '__main__':
    sh = detect2('image.jpg')
    #print(sh)
```

Ce code permet de lire l'image où il faut détecter la forme. Il détermine le contour adapté pour effectuer le code de détection de forme vu précédemment. Ensuite, il crée une nouvelle image avec l'inscription de la forme et de la couleur.