RAPPORT FINAL

par

Jeffrey COUSINEAU

RAPPORT FINAL DE PROJET SPÉCIAL
PRÉSENTÉ À TONY WONG
DANS LE CADRE DU COURS GPA791
DU BACCALAURÉAT EN GPA

MONTRÉAL, LE 12 AOÛT 2019

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

REMERCIEMENTS

Merci à mon superviseur Tony Wong qui a gentiment accepté de superviser ce projet et qui m'a soutenu tout au long de ce travail.

Merci également au club Walking Machine grâce auquel j'ai pu acquérir d'innombrables connaissances et qui m'ont permises de réaliser à bien ce projet.

Automatisation de la création d'un ensemble de données pour la détection d'objets

Jeffrey COUSINEAU

RÉSUMÉ

Ce projet permettra au club Walking Machine d'augmenter son efficacité lors de la compétition Robocup@Home en leur fournissant un outil facilitant l'acquisition d'images de différents objets et permettant, par la suite, l'entrainement d'un modèle de reconnaissance d'objet à partir des images acquises grâce au système développé durant ce projet.

Plus précisément, ce rapport présentera tout d'abord le développement d'une plateforme rotative automatisée pouvant accueillir des objets de divers poids et tailles. Cette plateforme présente un fond de couleur uniforme vert qui permettra par la suite au logiciel également développé lors de ce projet, de retirer le fond de couleur et de garder seulement l'objet. La plateforme permettra de faciliter l'acquisition d'image de l'objet sous plusieurs angles et d'accélérer le processus.

TABLE DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

- YOLO: You Only Look Once
- RPM: roulement par minute
- PWM : pulse width modulation
- SSR : solid state relay
- HSV : hue, saturation, value

INTRODUCTION

Dans le contexte de la compétition Robocup@Home, plusieurs tâches impliquent la détection de divers objets dans un environnement représentant un appartement. Ces objets sont inconnus à priori de la compétition et ne sont que présentés lors de la première journée de préparation. Notre stratégie afin de faire la détection de ces objets implique de créer un ensemble de données représentant le plus fidèlement ces derniers et par la suite de les entraîner à l'aide de YOLO (You Only Look Once). Cela se fait en prenant des photos de ceux-ci et par la suite, en identifiant le contour de ces objets à l'aide d'un cadre de sélection. Puisque le nombre de photos nécessaires est très élevé (environ 300 par objet), nous avons remarqué lors de notre compétition en 2018 que l'identification manuelle est une tâche très longue.

OBJECTIFS VISÉS

L'objectif principal est l'automatisation de l'acquisition des images d'objets afin d'accélérer la création d'un ensemble de données. Cela se découpe en trois tâches principales :

- La construction d'une plateforme rotative pouvant accueillir des objets d'un poids variant entre 100g et 1kg ainsi que d'une largeur pouvant atteindre environ 20 cm. Ces dimensions proviennent d'une analyse des différents objets utilisés chaque année en compétition;
- Le développement d'un algorithme de traitement d'image permettant d'enlever un fond d'une couleur uniforme et d'extraire l'objet principal;
- Le développement d'un script permettant la génération d'un ensemble de données prenant comme entrée les images d'objets précédemment générée ainsi que différents fonds représentatifs de l'environnement.

APPROCHE

Afin d'identifier les différentes tâches à effectuer, nous nous sommes basés sur le processus que nous mettions déjà en application lors de notre compétition et nous nous sommes posé la question, de quelle façon pourrions-nous automatiser ces différentes étapes. L'objectif premier étant d'obtenir un ensemble d'images en réduisant le temps de temps de traitement le plus possible.

Donc pour générer ces images, nous avions besoin d'un système qui serait capable d'analyser un objet et de produire un ensemble de photos de cet objet, sous plusieurs angles et en ayant identifié le contour de l'objet, nous permettant ainsi d'extraire le fond et de garder seulement la partie concernant l'objet spécifique.

Selon notre expérience avec le cours de vision artificielle, nous savions qu'il était possible d'obtenir un résultat qui nous convenait grâce à certains filtres permettant d'extraire la couleur d'une image, c'est ainsi que nous avons déterminé la première étape comme étant l'algorithme de segmentation. Par la suite, puisque la caméra que nous utilisions était déjà configurée afin d'utiliser ROS, nous avons décidé d'interfacer notre logiciel avec ROS directement.

En troisième lieu, nous avions besoin d'une interface afin de faciliter la gestion des données et des images, mais également afin de limiter la région de l'image qui serait traiter par notre algorithme. Finalement, les dernières étapes consistent à générer en continue différentes images de l'objet, mais également de générer les fichiers nécessaires pour l'apprentissage de notre modèle de détection d'objets, une étape avec laquelle nous sommes déjà familier.

CHAPITRE 1

CONCEPTION DE LA PLATEFORME ROTATIVE

1.1 Objectifs

L'objectif principal de la plateforme rotative, est comme son nom l'indique d'offrir une plateforme sur laquelle nous pourrions déposer un objet et le faire tourner. Cette rotation serait automatisée par un moteur qui tournerait à très basse vitesse, soit environ 1 RPM. La plateforme accueillera différents objets dont la taille et le poids peuvent varier mais qui se catégorisent principalement comme étant de la nourriture.

1.2 Choix des composants

Afin de concevoir la plateforme rotative, nous devions penser à un moyen facile d'intégrer un moteur à une plateforme afin d'entraîner un mouvement de rotation. Plusieurs idées nous sont venues en tête, soit d'utiliser un centre de table rotatif (lazy susan) ainsi qu'un moteur CC qui entraînerait la rotation par frottement ou bien de dessiner une solution et de l'imprimer en 3D.

La première étape était de commencer par chercher au travers des différents composants déjà disponible au sein du club de robotique Walking Machine afin de permettre la réutilisation. C'est ainsi qu'un moteur CC muni d'un réducteur de vitesse et qui présentait une forme parfaite pour accueillir une plateforme fut identifié (voir figure 1). Par la suite, un bac de plastique (voir Annexe 2) présentait les dimensions idéales afin d'accueillir le futur circuit électronique ainsi que la plateforme.

Figure 1 - Moteur CC de la plateforme rotative

1.3 Conception du circuit électrique

Afin de contrôler le moteur CC, un circuit devait être élaborer. L'objectif principal était d'obtenir la vitesse de rotation la plus faible possible. En utilisant un voltage de 6V, la vitesse du moteur était raisonnablement faible, mais nous devions trouver un moyen de la réduire encore plus afin d'avoir un meilleur contrôle et d'éviter des images floues causées par une rotation trop rapide.

La stratégie utilisée afin de réduire la vitesse fut d'utiliser un PWM envoyé au moteur. Un PWM sert à contrôler la vitesse d'un moteur au moyen d'impulsions. En réduisant la largeur de l'impulsion, nous réduisons la vitesse du moteur (voir figure 2).

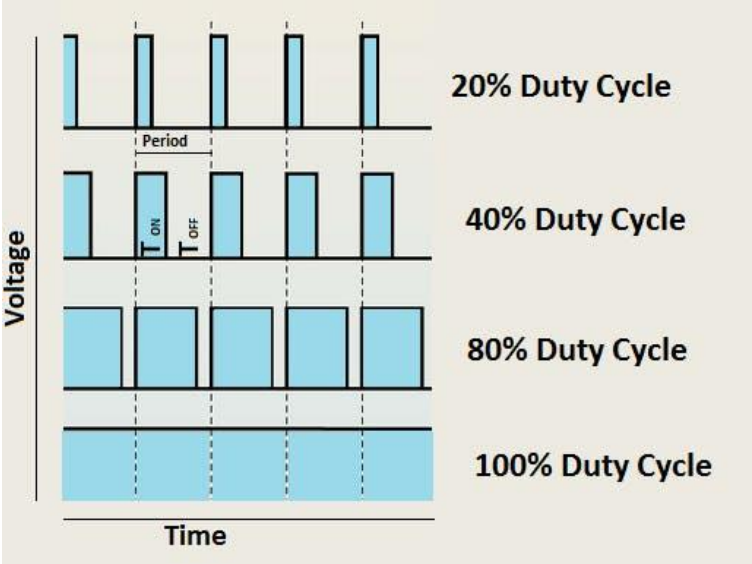

Figure 2 - Régulateur PWM

Ce PWM serait contrôlé grâce à un microcontrôleur (Arduino). Par contre, puisque le moteur nécessitait une alimentation externe qui ne pouvait être fournie par le Arduino, le PWM servait à contrôler un relais SSR qui agissait de la même manière en modulant la vitesse du moteur mais cette fois-ci en coupant l'alimentation menant au moteur plusieurs fois par secondes.

Une fois la partie du contrôle moteur réglé, nous avions également besoin de facilement arrêter la plateforme au moyen d'un interrupteur. Celui-ci fut donc relié au microcontrôleur qui lisait la valeur de l'interrupteur et, lorsque celui-ci changeait d'état, arrêtait ou lançait la rotation du moteur. Le circuit final est présenté en Annexe I et le montage final est présenté en Annexe II.

CHAPITRE 2

CONCEPTION DE L'ALGORITHME DE SEGMENTATION

2.1 Objectifs

L'objectif de cet algorithme est de prendre une image d'une caméra, isoler la région concernée et en extraire l'objet principal de l'image en retirant le fond de couleur uniforme.

2.2 Présentation des concepts utilisés

Pour cet algorithme nous utiliserons de la vision artificielle ainsi que de la manipulation d'image. L'outil principal qui sera utilisé se nomme OpenCV. Ce dernier est une bibliothèque graphique développée pour le traitement d'image. Celle-ci s'utilise principalement en C++ ou en Python et dans notre cas, ce sera en Python.

Nous interfacerons également notre logiciel avec ROS afin de faciliter la configuration des différents paramètres, mais également afin d'accédé à l'image de la caméra que nous utiliserons qui est déjà configurée afin de fonctionner avec ROS. ROS signifie Robot Operating System et est, quant à lui, un ensemble d'outils facilitant le développement de la robotique.

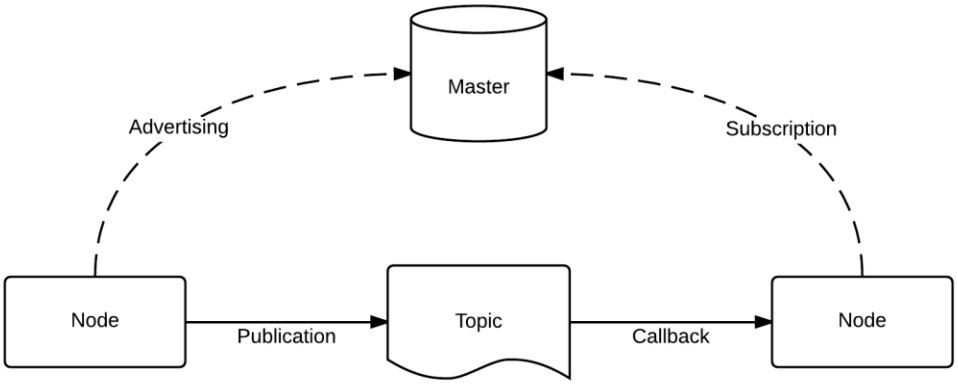

Figure 3 - Schéma de fonctionnement de ROS

Le fonctionnement de ROS est assez simple. Nous y retrouvons des node, qui sont en fait des programmes indépendants qui nous développons. Les topics sont des canaux de communication et qui permettent aux nodes de communiquer entre-elles. Nous pouvons donc souscrire ou publier sur un topic et notre programme réagira selon l'information reçue.

Dans notre cas, nous utiliserons principalement un topic, et plus précisément celui qui renvoie l'image de la caméra. Nous pourrons donc souscrire à celui-ci dans notre programme et appliquer notre algorithme sur l'image directement. Il nous sera même possible de publier l'image modifiée sur un autre topic afin de faciliter la visualisation.

2.3 Conception du programme

Au niveau de l'algorithme même, nous utiliserons plusieurs concepts déjà présents à l'intérieur d'OpenCV et que nous appliquerons sur notre image. Tout d'abord nous appliquerons un filtre permettant de brouiller légèrement l'image (ligne 80 ANNEXE III). Ce filtre est en fait une matrice 2D (voir Figure 4) qui est appliqué en convolution sur les pixels de l'image. Les dimensions de ce filtre peuvent changer, ce qui aura différents effets sur notre image. L'avantage de ce type de filtre lors que nous faisons du traitement d'image, c'est de réduire le bruit.

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 4 - Filtre Blur 3x3

Par la suite, nous convertirons notre image au format HSV (ligne 83 ANNEXE III). L'avantage d'utiliser le format est que les couleurs sont moins influencées par la luminosité. Il donc plus facile d'isoler une couleur en particulier. En utilisant que les paramètres que nous aurons choisi au niveau des valeurs HSV, nous appliquerons un masque qui ne gardera que les couleurs faisant partis de l'intervalle sélectionné (ligne 88 ANNEXE III).

Nous nous retrouvons donc avec une image binaire représentant les endroits où nous retrouvons la couleur sélectionnée par un 1, et le reste est représenté par un 0. Nous pouvons donc utiliser ce résultat et l'inverser afin que les 1 représentent les endroits où notre objet est situé. Cependant, il est possible que le résultat de cette opération affiche encore du bruit au sein de l'image. C'est pourquoi nous utiliserons un deuxième algorithme qui se nomme findContour (ligne 96 ANNEXE III) qui trouver le contour de toutes les formes fermées dans l'image. Nous n'aurons qu'à garder le plus grand pour trouver notre objet. Le résultat obtenu est représenté à la Figure 5. Par la suite, nous appliquerons ce résultat directement sur notre image d'origine afin d'avoir notre objet en couleur et avec un fond transparent (voir Figure 6).

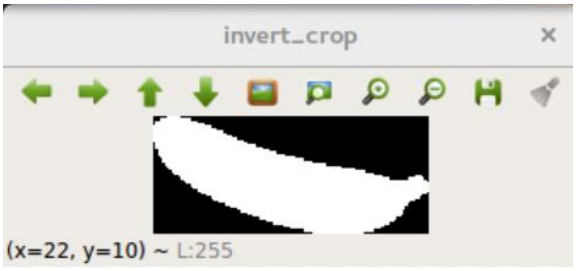

Figure 5 - Résultat suite à findContour

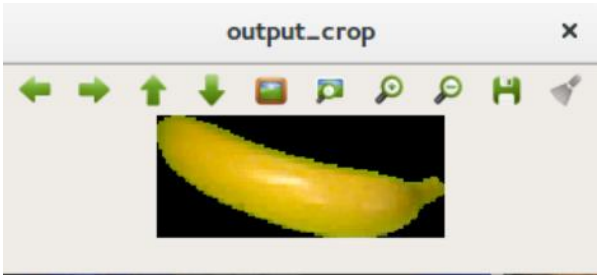

Figure 6 - Résultat final de la segmentation

2.4 Présentation du code

Le programme principal qui s'occupe d'appliquer l'algorithme de segmentation est assez simple. Tout d'abord, comme expliqué plus tôt, il souscrit au topic qui publie l'image de la caméra. Une fois qu'il reçoit une image, elle est tout d'abord transformée au format d'OpenCV

grâce à un package développé par ROS qui se nomme cv_bridge. Une fois l'image source transformée, l'algorithme est appliqué. Par la suite, il sauvegarde l'image résultante et attend la prochaine image. Un outil externe fournis avec ROS et qui se nomme RQT permet de modifier dynamiquement des paramètres définis dans une node. Ainsi, lorsque ceux-ci sont changé, au moment même l'algorithme utilisera les nouveaux paramètres.

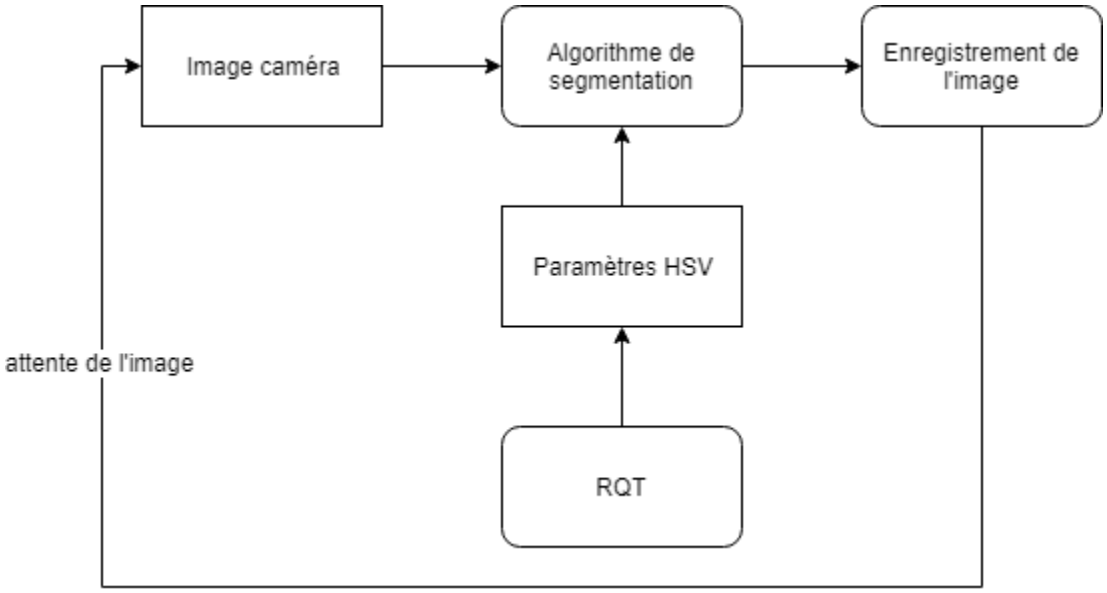

Figure 7- Boucle de l'extraction d'image

CHAPITRE 3

GÉNÉRATION DES DONNÉES

3.1 Objectifs

Le but de la génération des données est d'utiliser le résultat de la segmentation et de produire un ensemble de données. Cela commencera par accumuler les images de l'objet segmenté sous différents angles. Par la suite, ces différentes images seront transposées sur un fond aléatoires représentant l'environnement de compétition avec un fichier accompagnant chaque indiquant l'emplacement de l'objet dans l'image. Finalement, un fichier texte sera généré afin d'indiquer quelles images seront utilisées pour l'entraînement et les tests de notre réseau de neurones.

3.2 Présentation des concepts utilisés

Dans le cas de la génération des données, les concepts utilisés sont basiques. Nous parlons tout d'abord de la copie de fichiers en utilisant un script python. Par la suite, nous aurons besoin de faire une copie d'une image sur une autre. Cela implique seulement des concepts en traitement d'image comme la lecture et l'écriture des pixels d'une image. Finalement, nous aurons besoin d'écrire dans un fichier texte afin de générer les fichiers nécessaires à l'entrainement ainsi qu'aux tests de reconnaissance d'objets.

3.3 Conception du programme

Cette partie du programme utilise trois scripts différents. Le premier est appelé lorsque nous voulons sauvegarder les images actuelles pour une classe d'objet et est présent dans le même fichier python que celui qui est exécuté pour l'exécution de la segmentation (voir ANNEXE III). Ce code souscrit à un topic et attend de recevoir une commande via ce dernier. Nous n'avons qu'à publier une valeur qui représentera le nombre d'image à sauvegarder. Le programme attend une seconde entre chaque sauvegarde. Ces images sont envoyées dans un dossier spécifié dans le script. La commande permettant d'exécuter ce script est : « rostopic

pub /dataset/save_image std_msgs/Int32 ``data : 30`` », celle-ci signifie que nous publions sur le topic « /dataset/save_image » la valeur 30 représentée par un Int32.

Par la suite une fois que les images qui nous intéressaient sont enregistrées. Nous voulons prendre ces dernières et générer une image contenant un fond contextuel, donc qui représente l'environnement de la compétition afin d'avoir un meilleur taux de reconnaissance.

Le script « generate_image.py » (voir ANNEXE III), chargera au hasard, une image se trouvant dans le dossier spécifié. Par la suite, une position au hasard sur l'image sera générée et l'objet sera recopié à cet endroit. Finalement, l'image finale sera enregistrée dans le dossier portant le numéro de la classe de l'objet et un fichier texte sera également créé. Ce fichier contiendra les coordonnées de l'objet dans l'image au format (X, Y, Largeur, Hauteur).

Figure 8 - Image transposée sur un fond aléatoire

Finalement, le dernier script « generate_test_train.py » se chargera de faire la liste de toutes les images présentes dans le dossier et de générer un fichier qui contiendra le chemin vers

chaque fichier. 70% de ces images seront présentes dans le fichier train.txt pour l'entraînement du réseau de neurones et le 30% restant servira pour les tests dans le fichier test.txt.

CONCLUSION

En conclusion, ce projet fut une bonne occasion pour mettre à profit mes diverses connaissances avec ROS ainsi que la vision artificielle. Je crois que ce projet offrira un gain de temps significatif au niveau de l'entrainement du modèle de reconnaissance d'objets.

Malgré le fait que je n'ai pas pu tester l'entièreté des résultats du processus, j'ai pu m'assurer que le processus en lui-même était fonctionnel et que tous les outils nécessaires étaient développés. De plus, je suis fier des résultats de l'algorithme de segmentation qui permet d'avoir facilement un objet sans fond.

Plusieurs points seraient encore à améliorer sur la modularité des différents scripts, comme l'ajout de paramètres et de fichiers de configuration permettant de facilement changer les chemins des fichiers et le nom des classes d'objets. De plus, un script permettant de faire de l'augmentation des données serait nécessaire afin de rendre la détection plus robuste aux différents changements dans l'environnement, comme des changements de luminosité ou des occlusions. Cependant, je crois que l'équipe a maintenant assez de documentation pour continuer ce projet et l'améliorer.

ANNEXE I

SCHÉMA ÉLECTRIQUE DE LA PLATEFORME ROTATIVE

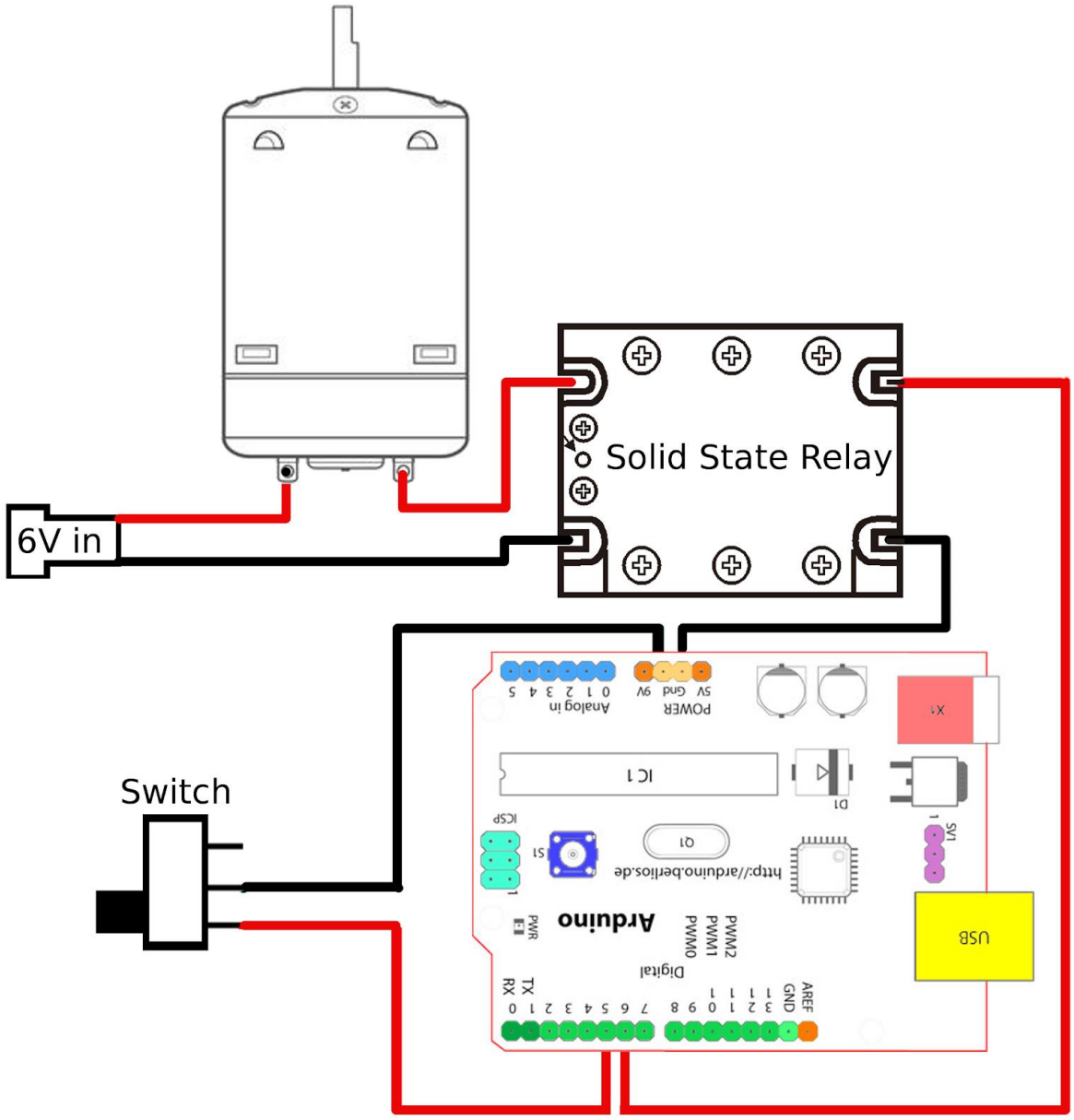

Figure 9 – Schéma électrique de la plateforme rotative

ANNEXE II

PLATEFORME ROTATIVE

Figure 10 - Plateforme rotative avec plateau, sans plateau et vue de dessous

ANNEXE III

SCRIPT DE SEGMENTATION

```python
#!/usr/bin/env python

# Author        : Jeffrey Cousineau
# Date          : 11 août 2019
# Description   : Script principal permettant la sélection de la région d'intérêt,
#                 l'extraction du fond de couleur et l'enregistrement des images.

from __future__ import print_function

import roslib
roslib.load_manifest('wm_dataset_preparation')
import sys
import rospy
import cv2
import numpy as np
import os
from std_msgs.msg import String
from std_msgs.msg import Bool, Int32
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from dynamic_reconfigure.server import Server
from wm_dataset_preparation.cfg import object_extractionConfig
from shutil import copyfile
import time

class image_converter:

  def __init__(self):
    self.image_pub = rospy.Publisher("image_topic_2",Image)
    self.srv = Server(object_extractionConfig, self.srv_callback)
    self.bridge = CvBridge()
    self.save_image_sub = rospy.Subscriber("/dataset/save_image",Int32,self.save_image)

    self.image_sub = rospy.Subscriber("/camera/rgb/image_raw/slow",Image,self.callback)
    self.fgbg = cv2.createBackgroundSubtractorMOG2()
    self.blur = np.ones((5, 5), np.float32) / 25
    self.tapis_low = np.array([0,3,0])
    self.tapis_high = np.array([152,247,255])
    self.first = True
    self.counter = 0
    self.save = True

  # Callback pour configurer l'interval HSV
  def srv_callback(self, config, level):
    self.tapis_low = np.array([config["H_low"],config["S_low"],config["V_low"]])
    self.tapis_high = np.array([config["H_high"],config["S_high"],config["V_high"]])
    return config

```

```
# Fonction pour enregistrer l'image actuelle avec le fond extrait
# TODO: utiliser un chemin relatif
def save_image(self,data):
  img_dir = os.listdir("/home/jeffrey/dataset_ws/src/wm_dataset_preparation/images/transparent/")
  for i in range(0,int(data.data)):
      time.sleep(1)
      copyfile("/home/jeffrey/dataset_ws/src/wm_dataset_preparation/result.png",
      "/home/jeffrey/dataset_ws/src/wm_dataset_preparation/images/transparent/result"+str(i+len(img_dir))+".png")

# Callback de l'image de la caméra
def callback(self,data):
  # Passer les premières images sinon il y a un problème avec les couleurs
  if self.counter < 10:
    self.counter += 1
  else:
    try:
      # Conversion vers bgr8
      cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
      print(e)

    if self.first:
        # Sélection de la région d'intérêt
        self.r = cv2.selectROI(cv_image)
        self.first = False

    # Découpe de l'image selon la région d'intért
    cv_image = cv_image[int(self.r[1]):int(self.r[1] + self.r[3]), int(self.r[0]):int(self.r[0] + self.r[2])]

    # Substraction du fond de couleur
    output= cv_image
    output_no_contour = cv_image.copy()
    cv_image = cv2.filter2D(cv_image,-1,self.blur)

    # Conversion vers HSV
    hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
    # Blur pour enlever du bruit
    cv_image = cv2.filter2D(cv_image, -1, self.blur)

    # Appliquer le masque HSV
    mask = cv2.inRange(hsv, self.tapis_low, self.tapis_high)
    mask2 = mask # Sauvegarde du masque pour visualisation seulement
    mask3 = mask
    # Inversion du masque binaire
    mask3 = cv2.bitwise_not(mask3)
    invert = mask3

    # Algorithme de recherche de contours
    im2, contours, hierarchy = cv2.findContours(mask3, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

```

```
        # Si un contour est trouvé
        if len(contours) != 0:

            # Trouver le plus gros contour de la liste
            c = max(contours, key=cv2.contourArea)
            cv2.drawContours(output, c, -1, 255, 3)
            x, y, w, h = cv2.boundingRect(c)

            # Dessiner le rectangle identifiant l'objet
            cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 2)

            inverted_image = invert[y:y+h, x:x+w]
            out_crop = output_no_contour[y:y+h, x:x+w]
            out_crop = cv2.cvtColor(out_crop, cv2.COLOR_RGB2RGBA)

            for line in range(0, len(inverted_image)):
              for elem in range(0, len(inverted_image[0])):
                if inverted_image[line,elem] != 255:
                  out_crop[line,elem] = 0

            # Enregistrer dans un fichier temporaire
            cv2.imwrite("/home/jeffrey/dataset_ws/src/wm_dataset_preparation/result.png", out_crop)
            cv2.imshow("invert_crop", invert[y:y+h, x:x+w])
            cv2.imshow("output_crop", out_crop)

        cv2.waitKey(3)

        try:
          # Envoie de l'image sur un topic ROS pour visualisation
          self.image_pub.publish(self.bridge.cv2_to_imgmsg(output, "bgr8"))
        except CvBridgeError as e:
          print(e)

def main(args):
    rospy.init_node('image_converter', anonymous=True)
    ic = image_converter()

    try:
      rospy.spin()
    except KeyboardInterrupt:
      print("Shutting down")
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)
```

>

ANNEXE IV

SCRIPT DE GÉNÉRATION DES IMAGES

```python
#!/usr/bin/env python
from __future__ import print_function

# Author        : Jeffrey Cousineau
# Date          : 11 août 2019
# Description   : Script permettant de prendre les images des différentes classes d'objet
#                 avec un fond transparent et de les transposer sur un fond aléatoire afin
#                 de générer plusieurs images. Ce la génère également le fichier contenant
#                 les coordonées de l'objet dans l'image

import roslib
roslib.load_manifest('wm_dataset_preparation')
import sys, os
import rospy
import cv2
import numpy as np
import random
from std_msgs.msg import String
from std_msgs.msg import Bool
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from dynamic_reconfigure.server import Server
from wm_dataset_preparation.cfg import object_extractionConfig

class image_generator:

    def __init__(self):
        # Liste des classes utilisées pour la génération d'images
        # TODO : utiliser un fichier de configuration
        self.classes = ["biscuit","frosty fruits","snakes","cloth","dishwasher tab","sponge",
        "trash bags","beer","chocolate milk","coke","juice","lemonade","tea bag","water","carrot",
        "cereals","noodles","onion","vegemite","apple","kiwi","lemon","orange","pear","cheetos",
        "doritos","shapes chicken","shapes pizza","twisties","bowl","shot glass"]

        self.script_dir = sys.path[0]
        self.image_path = os.path.join(self.script_dir, '../images/')

    # Fonction qui génère les nouvelles images ainsi que le fichier de coordonnées
    def generate_new_image(self):
        count = 0
        self.class_id = 0

        # Boucle pour toutes les classes  d'objet
        for i in self.classes:
            self.object_name = i

            # Chemin vers le dossier comprenant les différents fonds
            background_path = "/home/jeffrey/dataset_ws/src/wm_dataset_preparation/images/background"
            # Chemin vers le dossier final où seront enregistrées les images
            final_folder = "/home/jeffrey/dataset_ws/src/wm_dataset_preparation/images/objects_w_bg/"+self.object_name
            bg_list = os.listdir(background_path)
            try:
                os.mkdir(final_folder)
            except:
                print("Folder already exists")
            image_path = "/home/jeffrey/dataset_ws/src/wm_dataset_preparation/images/transparent/"+self.object_name
            img_list = os.listdir(image_path)

```

```
            # Génère 3 images différentes pour chaque image d'origine
            for i in range(0,2):
                for img in img_list:
                    f = open(final_folder + "/" + str(count)+".txt", "w+")
                    print(str(self.class_id) + " " + self.object_name + " "+str(count))

                    # Lecture de l'image de fond et conversion vers RGBA
                    self.background = cv2.imread(background_path+"/"+bg_list[random.randint(0,len(bg_list)-1)])
                    self.background  = cv2.cvtColor(self.background , cv2.COLOR_RGB2RGBA)
                    self.object = cv2.imread(image_path+"/"+img, cv2.IMREAD_UNCHANGED)
                    out_image = self.background.copy()

                    try:
                        height_object = len(self.object)
                    except:
                        print(img)
                    width_object = len(self.object[0])
                    height_bg = len(self.background)
                    width_bg = len(self.background[0])

                    # Génération d'une position aléatoire
                    # Grandeur de l'image - grandeur de l'objet
                    random_y = random.randint(0,height_bg - height_object)
                    random_x = random.randint(0,width_bg - width_object)

                    # Transposition de l'image de l'objet sur le fond
                    for line in range(0, height_object):
                        for elem in range(0, width_object):
                            if self.object[line,elem][3] != 0:
                                out_image[line+random_y,elem+random_x][0] = self.object[line,elem][0]
                                out_image[line+random_y,elem+random_x][1] = self.object[line,elem][1]
                                out_image[line+random_y,elem+random_x][2] = self.object[line,elem][2]

                    # Enregistrement de l'image finale
                    cv2.imwrite(final_folder+"/"+str(count)+".JPEG", out_image)
                    count += 1

                    # Génération du fichier de coordonnées (bounding boxes)
                    x_min = float(random_x) / float(width_bg)
                    x_max = (float(random_x) + float(width_object)) / float(width_bg)
                    final_width = x_max-x_min

                    y_min = float(random_y) / float(height_bg)
                    y_max = (float(random_y) + float(height_object)) / float(height_bg)
                    final_height = y_max - y_min

                    # Enregistrement du fichier
                    f.write(str(self.class_id) + " " + str(x_min+final_width/2) + " "
                        + str(y_min+final_height/2) + " " + str(final_width) + " " + str(final_height))
                    f.close()
            self.class_id += 1

def main(args):
    ic = image_generator()
    ic.generate_new_image()

if __name__ == '__main__':
    main(sys.argv)
```

ANNEXE V

SCRIPT DE GÉNÉRATIONS DES FICHIERS TRAIN/TEST

```python
#!/usr/bin/env python

# Author        : Jeffrey Cousineau
# Date          : 11 août 2019
# Description   : Script qui génére le fichier indiquant à YOLO chaque images utilisées
#                 pour l'entrainement ainsi que les tests
#                 => train.txt et test.txt

import sys, os

class image_generator:
    def __init__(self):
        # Liste des classes utilisées
        # TODO : utiliser un fichier de configuration
        self.classes = ["biscuit","frosty fruits","snakes","cloth","dishwasher tab","sponge",
        "trash bags","beer","chocolate milk","coke","juice","lemonade","tea bag","water","carrot",
        "cereals","noodles","onion","vegemite","apple","kiwi","lemon","orange","pear","cheetos","doritos",
        "shapes chicken","shapes pizza","twisties","bowl","shot glass"]
    def generate_new_image(self):
        # Dossier contenant les images générées par generate_image.py
        # TODO : utiliser un fichier de configuration
        objects_folder = "/home/jeffrey/dataset_ws/src/wm_dataset_preparation/images/objects_w_bg"
        train_txt = open(objects_folder + "/train_robocup2019.txt", "w+")
        test_txt = open(objects_folder + "/test_robocup2019.txt", "w+")

        # Boucle pour toutes les classes définies
        for i in self.classes:
            image_path = objects_folder + "/" + i
            img_list = os.listdir(image_path)
            img_list_no_txt = []

            # Récupère la liste de toutes les images de la classe
            for j in img_list:
                if j.endswith(".JPEG"):
                    img_list_no_txt.append(j)

            # Mélange la liste d'image
            random.shuffle(img_list_no_txt)
            count = 0

            # Choisi 30% d'images pour les tests et 70% pour l'entrainement
            # TODO : déterminer les pourcentage dans un fichier de configuration
            for img in img_list_no_txt:
                if count > len(img_list_no_txt)*0.3:
                    train_txt.write("data/" + str(i) + "/" + img+'\n')
                else:
                    test_txt.write("data/" + str(i) + "/" + img+'\n')
                count += 1

        train_txt.close()
        test_txt.close()

def main(args):
    ic = image_generator()
    ic.generate_new_image()


if __name__ == '__main__':
    main(sys.argv)
```

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Opencv-python-tutroals.readthedocs.io. (2019). *Welcome to OpenCV-Python Tutorials's documentation! — OpenCV-Python Tutorials 1 documentation*. [En ligne] Accessible à : https://opencv-python-tutroals.readthedocs.io/en/latest/ [Accédé le 13 Juin. 2019].

Kummarikuntla, T. (2019). *Blue or Green Screen Effect with OpenCV [Chroma keying]*. [En ligne] Medium. Available at: https://medium.com/fnplus/blue-or-green-screen-effect-with-open-cv-chroma-keying-94d4a6ab2743 [Accédé le 25 Mai. 2019].

Docs.opencv.org. (2019). *OpenCV: Contours : Getting Started*. [En ligne] Accessible à: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html [Accédé le 10 Juin. 2019].

Wiki.ros.org.
(2019). *cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages - ROS Wiki*. [En ligne] Accessible à: http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages [Accédé le 02 Juin. 2019].

Wiki.ros.org. (2019). *dynamic_reconfigure - ROS Wiki*. [En ligne] Accessible à: http://wiki.ros.org/dynamic_reconfigure [Accédé le 28 Juin. 2019].

https://www.learnopencv.com/how-to-select-a-bounding-box-roi-in-opencv-cpp-python/

Consulting, A. (2019). *Alpha Blending using OpenCV (C++ / Python)* | *Learn OpenCV*. [En ligne] Learnopencv.com. Accessible à: https://www.learnopencv.com/alpha-blending-using-opencv-cpp-python/ [Accédé le 1er Juillet. 2019].

GitHub. (2019). *Paperspace/DataAugmentationForObjectDetection*. [En ligne] Accessible à: https://github.com/Paperspace/DataAugmentationForObjectDetection [Accédé le 10 Juillet. 2019].

Hackernoon.com. (2019). *Understanding YOLO*. [En ligne] Accessible à: https://hackernoon.com/understanding-yolo-f5a74bbc7967 [Accédé le 19 Mai. 2019].