

Rapport PAN2 (groupe 3.2)

Module de traitement des images

- *Position du module de Traitement des images (TdI) dans le projet PACT :*

Dans notre projet PACT, les enfants participant à notre l'atelier de sensibilisation devront s'identifier auprès du robot, à l'aide de bracelets colorés, après avoir réalisé la bonne action de ramasser un déchet et de l'avoir jeté dans la poubelle qui correspond au critère du tri sélectif.

Ce module de traitement d'images a donc pour objectif de permettre l'identification des enfants concernés, en analysant par traitement d'image les photos de leurs bracelets prises par la caméra du robot. Pour ce faire, les bracelets seront constitués de deux couleurs uniquement, ces dernières étant confinées au sein d'un contour noir épais. Notre algorithme déterminera alors, à partir d'une photo de bracelet, les deux couleurs présentes au sein du contour noir. Les paires de couleurs utilisées étant uniques, cela aboutira à l'identification de l'enfant portant ce bracelet coloré.

- *Description du module de TdI : entrées/sorties, chaîne de traitements, détails techniques nécessaires à la compréhension de notre travail*

Notre algorithme, codé en Python, prend en entrée une image sous format .jpg ou .png. Il se charge par la suite de la convertir en tableau dont chaque élément représente un pixel. Ces pixels sont définis par des tuples à trois composantes qui correspondent respectivement à la nuance de rouge, vert et bleu qu'ils contiennent (chaque composante ayant donc une valeur allant de 0 à 255).

Nous souhaitons ensuite établir un seuillage binaire en intensité de l'image d'entrée. Afin d'améliorer la précision du traitement, nous convertissons donc la dimension RVB de travail en dimension HSI (Hue = teinte, S = saturation, I = Intensity) avant d'effectuer le seuillage, qui lui attribuera une intensité de 0 ou de 1 aux pixels de l'image selon leur valeur d'intensité de départ par rapport au seuil défini au préalable.

Une fois l'image traitée, nous déterminons les pixels qui nous intéressent, c'est à dire les pixels se trouvant à l'intérieur du cadre noir. Pour cela nous réalisons plusieurs étapes :

- Une inversion de l'image (les 0 en 1 et les 1 en 0) car nous utilisons une méthode de labeling pour déterminer la plus grande composante connexe de l'inverse qui ne touche pas le bord (i.e les pixels à l'intérieur du cadre). Il s'agit là du premier parcours de l'image.
- Nous parcourons ensuite une deuxième fois l'image en créant des composantes connexes en suivant la méthode de labeling décrite par M. Roux. Pour cela nous gérons en parallèle du parcours de l'image une variable donnant le numéro de composante connexe en cours de traitement et s'incrémentant lors de la découverte

d'un pixel ayant deux antécédents de composantes non nulles et différentes, un tableau de la même taille que l'image donnant la composante connexe de chaque pixel et une liste donnant les équivalences de composantes. Un antécédent d'un pixel, s'il existe, est son voisin gauche ou son voisin supérieur. Un antécédent peut ne pas exister si le pixel se trouve sur le bord gauche ou supérieur de l'image, ou si sa valeur n'est pas égale à 1, auquel cas nous ne le prendrons donc pas en compte. On parcourt donc l'image en regardant les antécédents de chaque pixel : s'il n'y en a pas, on crée alors une nouvelle composante, et s'il n'y en a qu'un, on donne comme composante au pixel courant celle de son antécédent. S'il y en a deux, soit les composantes des antécédents sont identiques et donc le pixel à la même composante, soit ils sont différents et on attribue la plus petite composante au pixel tout en associant les deux composantes (via la liste des équivalences de composantes). On trie ensuite les correspondances, c'est-à-dire on donne un nouveau numéro aux composantes, tout en les associant. Puis on modifie l'image : pour chaque pixel dont la valeur est 1 (les pixels de valeur 0 ne sont pas modifiés), on lui donne une nouvelle valeur qui est égale à son numéro de composante connexe, c'est le troisième parcours de l'image.

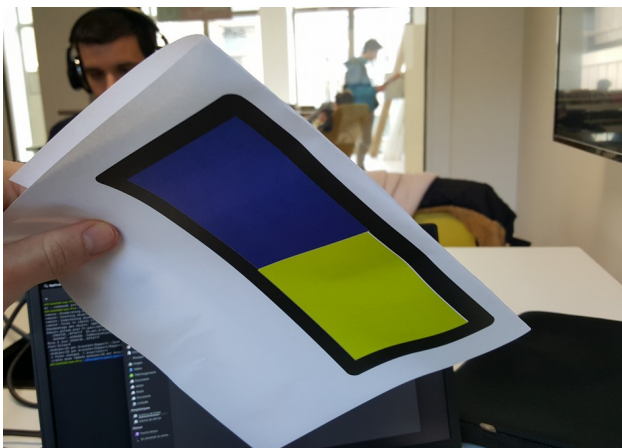
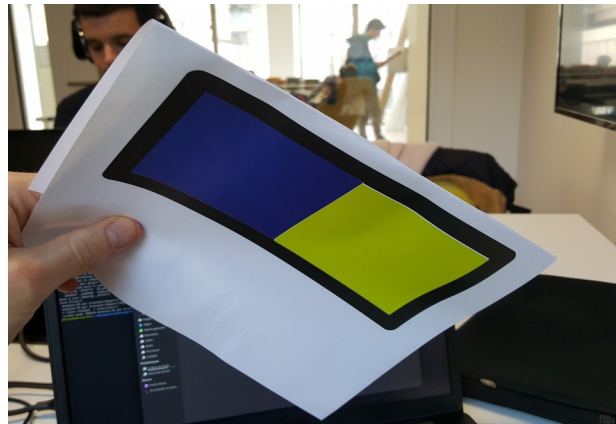
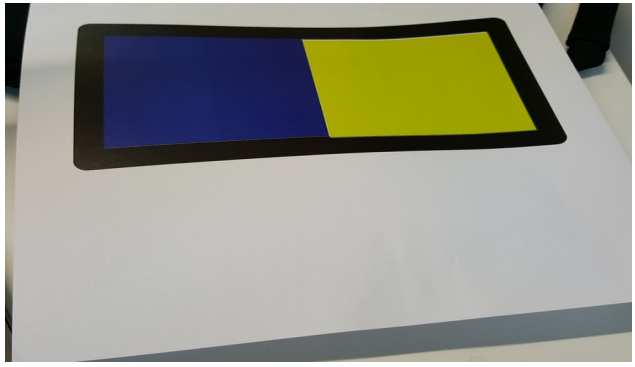
- On détermine enfin les pixels faisant partie de l'intérieur du cadre, c'est à dire ceux faisant partie de la plus grande composante connexe qui ne touche pas le bord. On note le numéro de composante connexe qui nous intéresse.

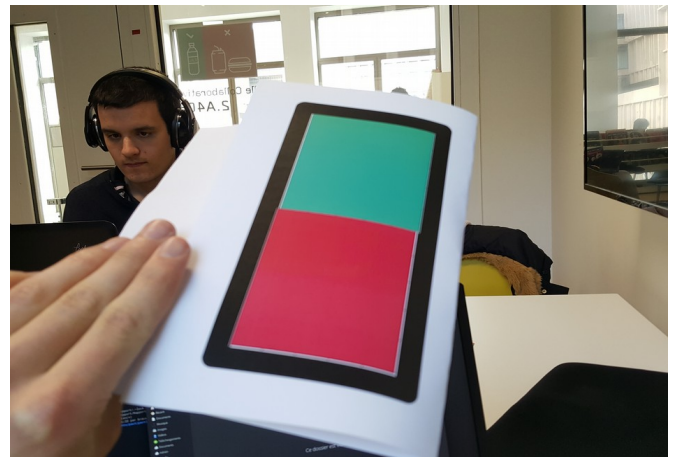
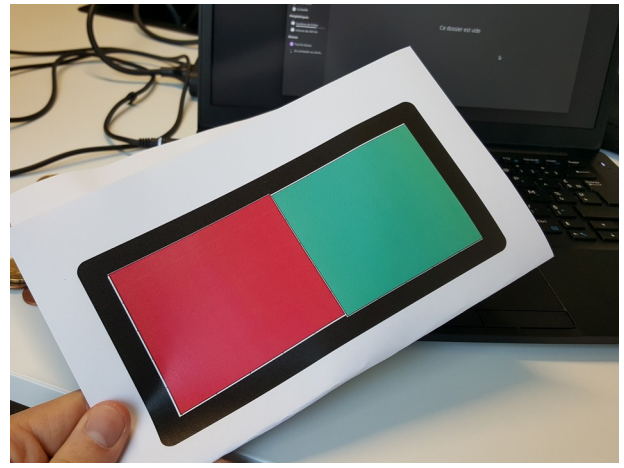
Ensuite, nous définissons un tableau qui contient l'ensemble des valeurs de saturation (de 0 à 180) à qui il associe le nombre de pixels de la composante connexe principale qui ont une telle valeur d'intensité. Pour remplir ce tableau, on parcourt les pixels de la composante connexe principale et on ajoute un au nombre de pixels correspondant à la teinte du pixel courant en cours d'analyse. Pour cela on parcourt en parallèle l'image de départ et le tableau des pixels avec les composantes connexes : pour chaque pixel de l'image de départ, s'il appartient à la composante voulue alors on ajoute sa valeur de saturation au tableau.

Lorsque ce tableau est construit, on identifie les deux pics principaux afin de déterminer les valeurs de teinte principales présentes dans l'image. Afin de ne pas prendre en compte les pics secondaires autour des pics principaux dû aux défauts de l'image par exemple, nous avons fait en sorte que les deux pics ne puissent pas présenter une teinte plus proche de 45 entre eux. Les couleurs choisies pour constituer les paires étant justement sélectionnées de telle sorte qu'elle ne soit pas de teinte trop proche entre elles, cela permet de déterminer effectivement les deux pics correspondants aux deux teintes dominantes sur l'image.

- Base de données d'images que vous avez acquises pour le module :







- L'état d'avancement du module, difficultés rencontrées :

Au stade actuel, le module est quasiment achevé en ce qui concerne son aspect informatique. Il reste à effectuer de nombreux tests avec la caméra du robot afin de déterminer le seuil le plus adéquat à utiliser pour le seuillage binaire. Il est envisageable également d'essayer d'améliorer la complexité et donc l'efficacité de notre programme, le temps de traitement étant de l'ordre d'une seconde pour une image prise par la caméra du robot. Afin de déterminer quelle était la partie du code qui augmentait significativement la complexité de notre algorithme, nous avons effectué des tests sur des images de plus grande résolution (prises par nos téléphone portable). Le temps de traitement ayant considérablement augmenté, nous en avons déduit que c'est le temps de parcours de l'image qui est le plus déterminant dans la complexité finale de notre algorithme.

La deuxième phase de notre module consistera désormais à élaborer les bracelets que porteront les enfants, aussi bien les couleurs, que les formes, en passant par la taille et matière du bracelet. Une fois fabriqués, il nous restera encore à tester si l'algorithme reconnaît effectivement les couleurs présentes sur les bracelets à partir d'une photo prise par le robot.

Les principales difficultés rencontrées ont été la compréhension de l'utilisation

de la base HSI, ainsi que l'algorithme de détermination des composantes connexes au sein de l'image seuillée. Nous avons également eu quelques difficultés à constituer des paires de couleur qui soient reconnaissables sans confusion par notre algorithme, il nous reste d'ailleurs encore à sélectionner des couleurs plus claires pour les bracelets à l'avenir.

- *Exemples de traitements sur des images significatives :*

(images numérotées de 1 à 14 de haut en bas de gauche à droite)

SEUILLAGE



image 1, seuillage de 100
résultat : pics à 15 et 120
→ bleu et jaune, correct



image 6, seuil de 80
pics à 115 et 15
→ bleu et jaune, ok



image 13, seuil de 100
pics à 177 et 103
→ turquoise et rouge, ok



image 14, seuil de 100
pics à 174 et 86
→ turquoise et rouge, ok

```

# prog.py

001| ## MAIN
002|
003| '''
004| découpage de l'algorithme en quatre étapes, détermination du seuillage, de la composante connexe, de
005| l'histogramme et enfin des pics
006| il y a donc 4 parties qui suivent, chacune implémentant une étape en une ou plusieurs fonctions
007| '''
008| def main(img):
009|     imageFiltree = seuillageGris(img, 100)
010|     imageComp, composante = main_composantes(imageFiltree)
011|     histogramme = histo(img, imageComp, composante)
012|     l1, l2 = determinationLABEL(histogramme)
013|     return (l1,l2)
014|
015| ## FILTRE IMAGE
016|
017| '''
018| algorithme de seuillage de l'image
019| '''
020|
021| import cv2
022| import matplotlib.pyplot as plt
023| from PIL import Image
024| import numpy as np
025|
026|
027| def seuillageGris(img, seuil):
028|     image = Image.open(img)
029|     ImageTab = np.array(image)
030|     imgHSI = cv2.cvtColor(ImageTab, cv2.COLOR_RGB2HSV)
031|     dim = imgHSI.shape
032|     res = np.zeros((dim[0],dim[1]))
033|     for i in range(dim[0]):
034|         for j in range(dim[1]):
035|             if imgHSI[i][j][2] < seuil:
036|                 res[i,j] = 1
037|     return res
038|
039|
040| ## COMPOSANTES CONNEXES
041|
042| '''
043| Algo de reconnaissance des composantes connexes d'un tableau composé de cases ayant soit 0 soit 1 pour
044| valeur
045| image : tableau de 0 et 1
046| '''
047| import numpy as np
048|
049| # Main
050|
051| '''
052| découpage de cette étape en sous étape, car cette étape est la plus longue et la plus compliquée
053| On inverse d'abord l'image, puis on détermine les composantes connexes de l'image, puis celle qui nous
054| intéresse
055| '''
056| def main_composantes(img):
057|     (a,b) = np.shape(img)
058|     inversionImage(img)
059|     determinationComposantes(img)
060|     interieur = determineInterieurCadre(img)
061|     return (img,interieur)
062|
063| # renvoie le numéro de la plus grande composante connexe de l'inverse de l'image qui ne touche pas le
064| bord, donc l'intérieur du cadre
065| def determineInterieurCadre(image):
066|     (a,b) = np.shape(image)
067|     composantes = [0]
068|     for i in range(a):
069|         for j in range(b):
070|             if image[i,j] not in composantes:
071|                 composantes.append(image[i,j])
072|     composantesBord = [0]
073|     for i in range(a):
074|         if image[i,0] not in composantesBord:
075|             composantesBord.append(image[i,0])
076|         if image[i,b-1] not in composantesBord:
077|             composantesBord.append(image[i,b-1])
078|     for j in range(b):
079|         if image[0,j] not in composantesBord:
080|             composantesBord.append(image[0,j])
081|         if image[a-1,j] not in composantesBord:

```

```

081|         composantesBord.append(image[a-1,j])
082|     tailleComposantesPasBord = []
083|     for elem in composantes:
084|         if elem not in composantesBord:
085|             tailleComposantesPasBord.append([elem, 0])
086|     for i in range(a):
087|         for j in range(b):
088|             pixel = image[i,j]
089|             if Existe(tailleComposantesPasBord, pixel):
090|                 newVal = Recup(tailleComposantesPasBord, pixel) + 1
091|                 Modif(tailleComposantesPasBord, pixel, newVal)
092|     max = 0
093|     plusGrandeComposante = -1
094|     for couple in tailleComposantesPasBord:
095|         if couple[1] > max:
096|             max = couple[1]
097|             plusGrandeComposante = couple[0]
098|     return plusGrandeComposante
099|
100| def Recup(tab, cle):
101|     for elem in tab:
102|         if elem[0] == cle:
103|             return elem[1]
104|
105| def Modif(tab, cle, val):
106|     for elem in tab:
107|         if elem[0] == cle:
108|             elem[1] = val
109|
110| def Existe(tab, cle):
111|     for elem in tab:
112|         if elem[0] == cle:
113|             return True
114|     else:
115|         return False
116|
117| # Fonctions
118|
119| def triCorrespondances(tab):
120|     for k in range(len(tab)):
121|         temp = tab[k]
122|         tab[k] = tab[temp]
123|     k = 1
124|     while k < len(tab):
125|         continu = True
126|         val = tab[k]
127|         valinf = tab[k-1]
128|         if val > valinf:
129|             tab[k] = valinf + 1
130|         if k == len(tab) - 1:
131|             break
132|         while continu and k < len(tab):
133|             k += 1
134|             if tab[k] == val:
135|                 tab[k] = valinf + 1
136|         else:
137|             continu = False
138|
139| def numsPrecedent(table, pixel, image):
140|     (i, j) = pixel
141|     res = []
142|     if i > 0 and image[i-1,j] == 1:
143|         res.append(HashRecup(table, (i-1,j)))
144|     if j > 0 and image[i,j-1] == 1:
145|         res.append(HashRecup(table, (i, j-1)))
146|     return res
147|
148| def inversionImage(image):
149|     (a,b) = np.shape(image)
150|     for i in range(a):
151|         for j in range(b):
152|             image[i,j] = (image[i,j] + 1)%2
153|
154| def determinationComposantes(image):
155|     (a,b) = np.shape(image)
156|     composantes = HashCreation(a,b)
157|     etiquette = 0
158|     correspondances = [0]
159|     for i in range(a):
160|         for j in range(b):
161|             if image[i,j] == 1:
162|                 etiquettesPrec = numsPrecedent(composantes, (i,j),image)
163|                 long = len(etiquettesPrec)
164|                 if long == 0:
165|                     HashAjout(composantes, (i,j), etiquette)
166|                     etiquette += 1

```

```

167|         correspondances.append(etiquette)
168|     elif long == 1:
169|         HashAjout(composantes, (i,j), etiquettesPrec[0])
170|     else:
171|         num1 = int(etiquettesPrec[0])
172|         num2 = int(etiquettesPrec[1])
173|         HashAjout(composantes, (i,j), num1)
174|         if num1 > num2:
175|             correspondances[num1] = num2
176|         elif num1 == num2:
177|             None
178|         else:
179|             correspondances[num2] = num1
180|     triCorrespondances(correspondances)
181|     for i in range(a):
182|         for j in range(b):
183|             composantes[i,j] = correspondances[int(composantes[i,j])]
184|     for i in range(a):
185|         for j in range(b):
186|             if image[i,j] == 1:
187|                 image[i,j] = HashRecup(composantes, (i,j)) + 1
188|
189| # Dictionnaire sous forme de tableau (complexité en temps bien meilleure)
190|
191| def HashCreation(a,b):
192|     res = np.zeros((a,b))
193|     for i in range(a):
194|         for j in range(b):
195|             res[i,j] = -1
196|     return res
197|
198| def HashAjout(table, cle, elem):
199|     table[cle] = elem
200|
201| def HashRecup(table, cle):
202|     return table[cle]
203|
204| def HashExiste(table, cle):
205|     return table[cle] != -1
206|
207| def HashModif(table, cle, newVal):
208|     table[cle] = newVal
209|
210| def HashSuppr(table, cle):
211|     table[cle] = -1
212|
213| ## CONSTRUCTION DE L'HISTOGRAMME ET DETERMINATION DES PICS
214|
215| import cv2
216| import matplotlib.pyplot as plt
217| from PIL import Image
218| import numpy as np
219|
220| def convertToHSI(img):
221|     image = Image.open(img)
222|     ImageTab = np.array(image)
223|     return cv2.cvtColor(ImageTab, cv2.COLOR_RGB2HSV)
224|
225| def histo(img, imgConnex, n):
226|     ImHSV = convertToHSI(img)
227|     histogramme = np.zeros((180,2))
228|     for i in range(180):
229|         histogramme[i,0] = i
230|     dim = ImHSV.shape
231|     for i in range(dim[0]):
232|         for j in range(dim[1]):
233|             if imgConnex[i][j] == n :
234|                 histogramme[ImHSV[i][j][0],1] += 1
235|     return histogramme
236|
237| import math
238|
239| def tableauCorresp():
240|     tab = []
241|     for i in range(15):
242|         tab.append("ROUGE")
243|     for i in range(30):
244|         tab.append("JAUNE")
245|     for i in range(30):
246|         tab.append("VERT")
247|     for i in range(30):
248|         tab.append("TURQUOISE")
249|     for i in range(30):
250|         tab.append("BLEU")
251|     for i in range(30):
252|         tab.append("MAGENTA")

```



```

253|     for i in range(15):
254|         tab.append("ROUGE")
255|     return np.array(tab)
256|
257| def determinationPics(histogramme):
258|     premierPic = max(histogramme)
259|     deuxiemePic = max2(histogramme, premierPic)
260|     return (premierPic,deuxiemePic)
261|
262| def determinationLABEL(histogramme):
263|     p1,p2 = determinationPics(histogramme)
264|     t = tableauCorresp()
265|     print(p1,p2)
266|     return (t[int(p1)], t[int(p2)])
267|
268| def max(tab):
269|     max = -math.inf
270|     indice_max = -1
271|     for i in range(len(tab)):
272|         if tab[i,1] > max:
273|             max = tab[i,1]
274|             indice_max = tab[i,0]
275|     return indice_max
276|
277| def max2(tab,max1):
278|     max = -math.inf
279|     indice_max = -1
280|     for i in range(len(tab)):
281|         if (i < (max1 - 45) or i > (max1 + 45)) and (i < (max1 + 180 - 45) or i > (max1 - 180 + 45)) and
tab[i,1] > max:
282|             max = tab[i,1]
283|             indice_max = tab[i,0]
284|     return indice_max

```