Institut Supérieur
d'Électronique de Paris

TIPE

Bras mécanique et sudoku

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5	2	6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

Laurent Tainturier & Alphonse Terrier
supervisé par
M. Patrick COUVEZ

2016-2017

Table des matières

Introduction

Chapitre 1

Présentation du sudoku

Chapitre 2

Électronique

Chapitre 3

Mécanique

Chapitre 4

Informatique

Tous les algorithmes sont implémentés en Python. Ils sont disponibles en annexe.

4.1 Reconnaissance du sudoku

Le script de reconnaissance du sudoku a été réalisé sous Python 2 avec le module de traitements d'image OpenCV. Il a été réalisé pour :

1. Reconnaître les chiffres dans une grille du sudoku
2. Déterminer la position spatiale de la grille

On photographie la grille avec une caméra Raspberry Pi (V2) comme celle-ci :

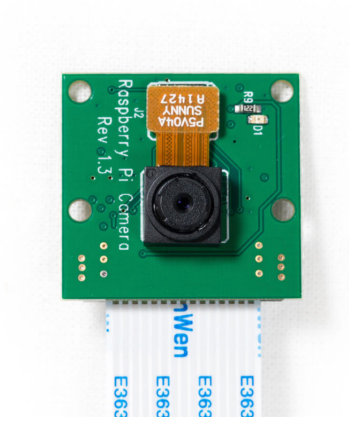

FIGURE 4.1 – Caméra Raspberry Pi V2

Voici la grille de sudoku qui nous servira d'exemple pour montrer toutes les actions du script :

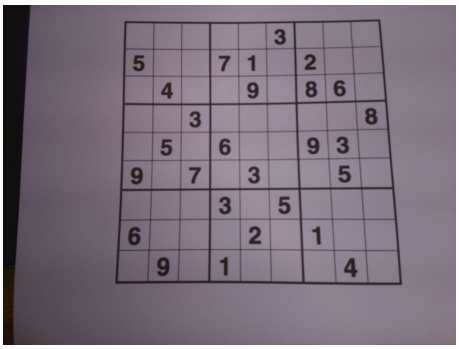

FIGURE 4.2 – Exemple de grille de sudoku

On applique sur cette photographie un filtre de type seuil (en anglais "threshold") qui va ensuite nous permettre de détecter les contours de la grille :

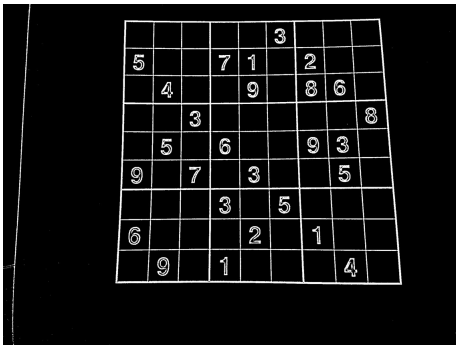

FIGURE 4.3 – Exemple de grille "seuillée"

Par cette transformation, on peut ensuite déterminer des équations de droites des contours extérieurs de la grille. Les coordonnées des intersections des droites seront celle des coins de la grille.

On découpe alors la grille de la photographie initiale en supprimant les éventuels effets de perspective.

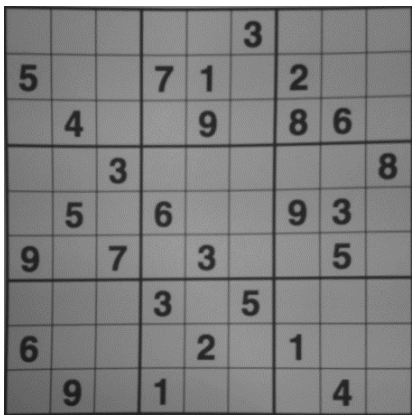

FIGURE 4.4 – Exemple de grille découpée sans perspective

On découpe chaque petite case de la grille comme ci-dessous :

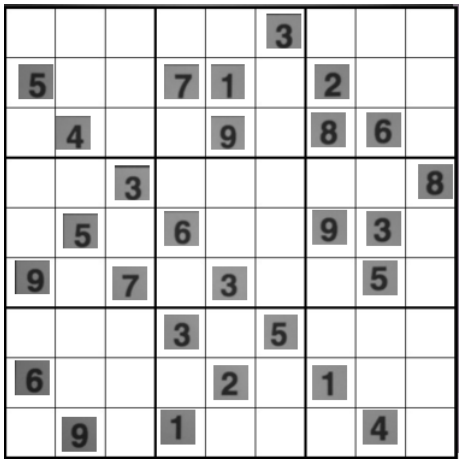

FIGURE 4.5 – Exemple de grille où chaque chiffre a été découpé

On utilise ensuite un module de reconnaissance de digits pour détecter les chiffres et on obtient la grille suivante, prête à être résolue :

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5	2	6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

FIGURE 4.6 – Exemple de grille prête à être résolue

4.2 Résolution du sudoku

4.3 Écriture et contrôle des moteurs

Annexe A

Fichier principal

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

import display as dp
import resolution as rs
import camera as cm
#import traitement_image as tr


class Sudoku:
    """
    Permet la gestion des sudoku à savoir :
        - leur édition par l'utilisateur pour obtenir le sudoku à résoudre
        - leur résolution à l'aide de différentes méthodes de résolution:
            - inclusion
            - exclusion
            - backtracking
            - ...
        - leur affichage à l'aide du module tkinter
    """

    def __init__(self):
        self.beta_version = True
        self.error = []
        self.taille = (3, 3)
        self.nb_cases = self.taille[0] * self.taille[1]
        self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
        self.liste_position = []
        self.methode_resolution = "Globale"

        self.Camera = cm.Camera(self)
        #self.Traitement = tr.Traitement(self)
        self.Resolution = rs.Resolution(self)
        self.Display = dp.Display(self)
        self.Display.updateSudoku(self.sudoku)

        self.Display.mainloop()

    def setError(self, error, off=True):
        if off:
            if error not in self.error:
                self.error.append(error)
        else:
            if error in self.error:
                self.error.remove(error)

    def getError(self):

        return self.error

    def startResolution(self, sudoku):
        self.sudoku, self.liste_position = self.Resolution.start(sudoku, self.methode_resolution)
```

```
55          self.Display.updateSudoku(self.sudoku, self.liste_position)
56
57      def stopResolution(self):
58          pass
59
60      def setMethodeResolution(self, methode):
61          self.methode_resolution = methode
62
63
64  Sudoku()
```

Annexe B

Script de résolution des sudokus

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import time
import numpy as np
from copy import copy


class Resolution:
    """
    Classe permettant de résoudre un sudoku grâce à différentes méthodes, à savoir :
        - inclusion
        - exclusion
        - bactracking
        - ...
    Si le sudoku n'est pas résoluble, lève une erreur
    """
    def __init__(self, boss):
        self.boss = boss
        self.taille = self.boss.taille
        self.nb_cases = self.boss.nb_cases
        self.sudoku = self.boss.sudoku
        self.methode_resolution = None
        self.possibilities = []
        self.starting_possibilities = []
        self.resolution = False
        self.carre = []
        self.ligne = []
        self.colonne = []

    def beforeStart(self):
        self.possibilities = []
        self.carre = []
        self.ligne = []
        self.colonne = []
        for i in range(self.nb_cases):
            values = [i + 1 for i in range(self.nb_cases)]
            self.carre.append(copy(values))
            self.ligne.append(copy(values))
            self.colonne.append(copy(values))
            x = 3 * (i // 3)
            y = 3 * (i % 3)
            for j in range(self.nb_cases):
                if self.sudoku[x + j // 3, y + j % 3] != 0 \
                        and self.sudoku[x + j // 3, y + j % 3] in self.carre[i]:
                    self.carre[i].remove(self.sudoku[x + j // 3, y + j % 3])
                if self.sudoku[i, j] != 0 and self.sudoku[i, j] in self.ligne[i]:
                    self.ligne[i].remove(self.sudoku[i, j])
                if self.sudoku[j, i] != 0 and self.sudoku[j, i] in self.colonne[i]:
                    self.colonne[i].remove(self.sudoku[j, i])
                if not self.sudoku[i][j] and (i, j) not in self.possibilities:
                    self.possibilities.append((i, j))
        self.starting_possibilities = copy(self.possibilities)

```

```
55      def start(self, sudoku, methode):
56          self.sudoku = copy(sudoku)
57          self.beforeStart()
58          self.methode_resolution = methode
59          zero_time = time.time()
60          n = 0
61          print(self.methode_resolution)
62          if self.methode_resolution == "Backtracking":
63              self.backTracking()
64          else:
65              self.resolution = True
66              while self.resolution:
67                  n += 1
68                  if self.methode_resolution == "Inclusion":
69                      self.inclusion()
70                  if self.methode_resolution == "Exclusion":
71                      self.exclusion()
72                  if self.methode_resolution == "Globale":
73                      self.inclusion()
74                      self.exclusion()
75                  if np.all(self.sudoku == sudoku):
76                      self.resolution = False
77                  sudoku = copy(self.sudoku)
78              if np.any(self.sudoku == np.zeros((self.nb_cases, self.nb_cases), int)):
79                  self.methode_resolution = "Backtracking"
80                  print("Backtracking")
81                  self.backTracking()
82
83          print(time.time() - zero_time, n)
84          return self.sudoku, self.starting_possibilities
85
86      def checkListe(self, x, y):
87          """
88          Renvoie la liste des valeurs possibles pour la case de coordonnées x et y
89          :param x: int: ligne
90          :param y: int: colonne
91          :return: liste: list
92          """
93          liste = []
94          if self.sudoku[x][y] == 0:
95              liste = [i + 1 for i in range(self.nb_cases)]
96              block_x = x - x % self.taille[0]
97              block_y = y - y % self.taille[1]
98              for i in range(self.nb_cases):
99                  if self.sudoku[x, i] in liste:
100                     liste.remove(self.sudoku[x, i])
101                 if self.sudoku[i, y] in liste:
102                     liste.remove(self.sudoku[i, y])
103                 if self.sudoku[block_x + i % self.taille[1], block_y + i // self.taille[0]] in liste:
104                     liste.remove(self.sudoku[block_x + i % self.taille[1], block_y + i // self.taille[0]])
105         return liste
106
107     def inclusion(self):
108         for x in range(self.nb_cases):
109             for y in range(self.nb_cases):
110                 self.checkValues(x, y)
111
112     def exclusion(self):
113         for n in range(self.nb_cases):
114             for k in self.carre[n]:
115                 x, y = 3 * (n // 3), 3 * (n % 3)
116                 x_possible = []
117                 y_possible = []
118                 for i in range(self.taille[0]):
119                     if k in self.ligne[x + i]: x_possible.append(x + i)
120                     if k in self.colonne[y + i]: y_possible.append(y + i)
121                 case_possible = []
122                 for x in x_possible:
123                     for y in y_possible:
124                         if (x, y) in self.possibilities: case_possible.append((x, y))
125                 self.setValuesEsclusion(case_possible, k)
126
127             j = n
```

```
128             for k in self.colonne[j]:
129                 carre_possible = []
130                 case_possible = []
131                 for i in range(self.taille[0]):
132                     if k in self.carre[3 * i + j // 3]:
133                         carre_possible.append(3 * i + j // 3)
134                 for i in range(self.nb_cases):
135                     if (i, j) in self.possibilities:
136                         if k in self.ligne[i] and (i, j) not in case_possible and \
137                                 3 * (i // 3) + j // 3 in carre_possible:
138                             case_possible.append((i, j))
139                 self.setValuesEsclusion(case_possible, k)
140 
141             i = n
142             for k in self.ligne[i]:
143                 carre_possible = []
144                 case_possible = []
145                 for j in range(self.taille[0]):
146                     if k in self.carre[3 * (i // 3) + j]:
147                         carre_possible.append(3 * (i // 3) + j)
148                 for j in range(self.nb_cases):
149                     if (i, j) in self.possibilities:
150                         if k in self.colonne[j] and (i, j) not in case_possible and \
151                                 3 * (i // 3) + j // 3 in carre_possible:
152                             case_possible.append((i, j))
153                 self.setValuesEsclusion(case_possible, k)
154 
155     def backTracking(self):
156         """
157         Résoud un sudoku selon la méthode de backtracking
158         :return: None or -1
159         """
160         liste_sudoku = []
161         i = 0
162         while i < len(self.possibilities):
163             x, y = self.possibilities[i]
164             liste = self.checkListe(x, y)
165             if liste:
166                 self.sudoku[x][y] = liste.pop(0)
167                 liste_sudoku.append(liste)
168                 i += 1
169             else:
170                 while not liste:
171                     i -= 1
172                     x, y = self.possibilities[i]
173                     try:
174                         liste = liste_sudoku[i]
175                     except IndexError:
176                         self.sudoku = self.boss.sudoku
177                         self.boss.setError("sudoku_insoluble")
178                         return -1
179                     if liste:
180                         self.sudoku[x][y] = liste.pop(0)
181                         liste_sudoku[i] = liste
182                         i += 1
183                         break
184                     else:
185                         liste_sudoku.pop(i)
186                         self.sudoku[x][y] = 0
187 
188     def checkValues(self, x, y):
189         if (x, y) in self.possibilities:
190             possibilities = []
191             n = 3 * (x // 3) + y // 3
192             for i in range(1, self.nb_cases + 1):
193                 if i in self.ligne[x] and i in self.colonne[y] and i in self.carre[n]:
194                     possibilities.append(i)
195             self.setValues(possibilities, x, y)
196 
197     def setValuesEsclusion(self, case_possible, k):
198         if len(case_possible) == 1:
199             x, y = case_possible[0][0], case_possible[0][1]
200             n = 3 * (x // 3) + y // 3
```

```
201                 self.sudoku[x, y] = k
202                 self.possibilities.remove((x, y))
203                 self.ligne[x].remove(k)
204                 self.colonne[y].remove(k)
205                 self.carre[n].remove(k)
206 
207         def setValues(self, possibilities, x, y):
208             if len(possibilities) == 1:
209                 k = possibilities[0]
210                 n = 3 * (x // 3) + y // 3
211                 self.sudoku[x, y] = k
212                 if (x, y) in self.possibilities: self.possibilities.remove((x, y))
213                 self.ligne[x].remove(k)
214                 self.colonne[y].remove(k)
215                 self.carre[n].remove(k)
216 
217 
218     if __name__ == "__main__":
219         class Boss:
220             def __init__(self):
221                 self.taille = (3, 3)
222                 self.nb_cases = 9
223                 self.sudoku = np.array([[3, 0, 0, 2, 0, 0, 0, 0, 5],
224                                         [0, 0, 0, 0, 0, 0, 0, 0, 0],
225                                         [0, 7, 8, 0, 0, 0, 2, 4, 0],
226                                         [0, 5, 0, 4, 0, 7, 0, 9, 0],
227                                         [0, 6, 0, 0, 2, 0, 0, 8, 0],
228                                         [0, 9, 0, 5, 0, 3, 0, 1, 0],
229                                         [0, 8, 1, 0, 0, 0, 6, 3, 0],
230                                         [0, 0, 0, 0, 0, 0, 0, 0, 0],
231                                         [7, 0, 0, 8, 0, 5, 0, 0, 1]])
232                 self.Resolution = Resolution(self)
233                 self.sudoku, position = self.Resolution.start(self.sudoku, "Globale")
234                 print(self.sudoku)
235 
236 
237         Boss()
```

Annexe C

Script de gestion de la caméra

```python
#!/usr/bin/env python3


class Camera:
    """
    Permet la gestion de la camera de la raspberry pi
    Si celle-ci n'est pas disponible ou le module 'picamera'
    n'a pas été installé correctement, lève une exception.
    """

    def __init__(self, boss):
        self.boss = boss
        self.camera = None
        self.tryError()

    def tryError(self):
        try:
            import picamera
            self.camera = picamera.PiCamera()
        except:
            self.boss.setError("camera_error")

    def takePhoto(self):
        try:
            self.camera.capture("Images/photos.jpg")
            print("The photo has been taken")
        except:
            self.boss.setError("camera_error")


if __name__ == '__main__':
    class Boss:
        def setError(self, error):
            if error == "module_camera":
                print("Le module 'picamera' n'a pas été installé correctement !")
            if error == "disponibilite_camera":
                print("La caméra n'est pas disponible !")

    Camera = Camera(Boss())
    Camera.takePhoto()
```