GHENT
UNIVERSITY

Multimedia project: puzzle solver

Youri Vassiliev, Francis Begyn

11 december 2017

academiejaar 2017-2018

FACULTY OF ENGINEERING
AND ARCHITECTURE

1 Inleiding

Het project voor multimedia dit jaar betreft hem het chrijven van een programma dat in staat is om op basis van een afbeelding van puzzelstukjes, puzzel op te lossen. Het oplossen van puzzel brengt bepaalde uitdagingen met zich mee, men moet zich hierop toeleggen en benadrukken waarop men zal focussen bij het oplossen van een de puzzel.

- Soort puzzel: Er zijn veel soorten puzzels en elk type brengt zo zijn uitdagingen en voordelen met zich mee
- Matching: Dit hangt deels af van het type, maar binnen elk type zijn er ook meerdere manieren om de puzzelstukjes te matchen (vorm, pixel, knn-match, ...)
- Snelheid: Ligt de nadruk op snelheid of op schaalbaaarheid. Het is heel lastig van een algoitme te schrijven dat zowel snel als schaalbaar werkt. Puzzels oplosen is namenlijk een NP-compleet probleem.

Wij hebben gekozen om te kijken voor een goede manier te vinden om de puzzelstukjes uit te knippen en een zo groot mogelijk aantal puzzels op te lossen. met een voorkeur voor de tiles. De tiles hebben een voorkeur gekregen vanwege het nut van ons project op verdere videobewerking. Met jigsaw zou men kunnen matching op basis van de vorm, maar een algoritme ontwikkeld voor tiles kan heel makkelijk aangepast worden om op afbeeldingen in het algemeen toegpast worden.

Het project wordt net zoals de labos in Python geschreven, we hebben echter wel gekozen voor Python3.6. dit is vanwege de voorkeur voor de syntax in die python versie. Er wordt gebruik gemaakt van de meest recente versie van OpenCV, versie 3.3.0 voor de bewerkingen en manipulaties op afbeeldingen. Daarnaast wordt er voor array operaties te versimpelen ook nog Numpy 1.13.3 gebrukt.

2 Structuur van het programma

Het programma wordt geschreven in Python, wat zich heel sterk de gewoonte heeft naar *class* georienteerd programmeren. Er is de keuze gemaakt om voornamenlijk 2 *classen* te ontwikkelen:

- Puzzle: Puzzle is de klasse die alles een puzzel zal opslaan. Hierin wordt ten eerste de volledige puzzel opgeslagen, samen met varianten van de afbeelding die men nodig zou hebben voor bewerkingen. Deze klasse is ook verantwoordelijk voor het halen van de puzzelstukjes uit de input voor de puzzel, deze stukjes worden daarna ook opgeslagen in deze klasse.

- Puzzlesolver: Puzzlesolver is de klasse die de effectieve algoritmes bezit voor het oplossen van de puzzel. Hierin bevindt de *matching* en *solving* functie.

Het idee achter deze klassen is dat men voor elke puzzel een Puzzle object aanmaakt, die men dan gewoon kan meegeven aan 1 Puzzlesolver. Dit zorgt ervoor dat onze klassen gemakkelijk geïntegreed kunnen zorden in andere programmas

3 Implementatie

3.1 Puzzle klasse

De meeste code in de puzzle klasse is niet geheel complex. De instantiatie van de klasse gebeurt simpelweg door een pad naar een afbeelding mee te geven aan de standaard constructor `Puzzle(some/path/to/image)`. Dit slaat de afbeelding op in de klasse, alsook een grijsschaal versie (want veel OpenCV functies maken gebruik van grijsschaal varianten).

Er is nog een basisfunctie zoals `show()` die de afbeelding weergeeft en een paar functies met betrekking to de contouren waarvan de voornaamste `contours()` is, die de contouren in de hoofdafbeelding bepaalt en opslaat in de klasse voor later gebruik.

De meest complexe functie is de functie die de puzzelstukjes uitsnijdt, `calc_pieces(margin, draw)`

```
1     def calc_pieces(self, margin=25, draw=False):
2         """ Determine the pieces if they're seperated from eachother and there's
              black space in between"""
3         # Initialise the used variables for finding the pieces
4         self.pieces = []
5         self.__new_pieces = []
6         self.pieces_gr = []
7         offset = int(margin/2)
8         width, height = self.puzzle.shape[:2]
9
10        # If we're drawing, make sure we don't draw on the original image
11        if draw:
12            mask = self.puzzle.copy()
13
14        # If theres no contours calculated yet, make sure to do so
15        if self.__contours is None:
16            self.contours()
17
18        # Create a bounding box with some margin around the contours
```

```
19          for i in self.__contours:
20              _, _, angle = cv2.minAreaRect(i)
21              rect = cv2.boundingRect(i)
22              x, y, w, h = rect
23              y_start = y-offset
24              x_start = x-offset
25              y_end = y+h+offset
26              x_end = x+w+offset
27
28              # Make sure that the bounding boxes remain within the size of the
                    image
29              if y_start < 0:
30                  y_start = 0
31              if x_start < 0:
32                  x_start = 0
33              if y_end > height:
34                  y_end = height-1
35              if x_end > width:
36                  x_end = width-1
37
38              # Copy the puzzle piece from the image
39              piece = self.puzzle[y_start:y_end,x_start:x_end].copy()
40
41              # Based on the minAreaRect function we can use the angle of that to
                    make sure the
42              # images are all orientated the same way
43              cols, rows = piece.shape[:2]
44              rotM = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
45              piece = cv2.warpAffine(piece, rotM, (cols, rows))
46
47              # Store the pieces in a publicly available attribute of Puzzle
48              self.pieces.append(piece)
49              # Automaticly store a grayscale version of the puzzle piece
50              self.pieces_gr.append(cv2.cvtColor(piece, cv2.COLOR_BGR2GRAY)
                    )
51
52              # On top of a copy of the image, draw the bounding boxes
53              if draw:
54                  cv2.rectangle(mask, (x_start, y_start), (x_end, y_end), (0, 255,
                        0), 1)
55
56          # Once we have the pieces, there's still a lot of black margin around
                them (because of the space
```

```
57          # needed for the rotation). We rerun the above code on each seperate
                piece so we get a better
58          # puzzle piece to work with with less black borders in it.
59          for piece, gray in zip(self.pieces, self.pieces_gr):
60              self.__thresh = np.asarray((gray>0)*255,dtype= np.uint8)
61              _, contours, hierarch = cv2.findContours(self.__thresh.copy(), cv2.
                    RETR_EXTERNAL,
62                                                       cv2.
                                                            CHAIN_APPROX_NONE
                                                            )
63              for i in contours:
64                  _, _, angle = cv2.minAreaRect(i)
65                  rect = cv2.boundingRect(i)
66                  x, y, w, h = rect
67                  y_start = y+1
68                  x_start = x+1
69                  y_end = y+h-1
70                  x_end = x+w-1
71                  if y_start < 0:
72                      y_start = 0
73                  if x_start < 0:
74                      x_start = 0
75                  if y_end > height:
76                      y_end = height-1
77                  if x_end > width:
78                      x_end = width-1
79
80                  p = piece[y_start:y_end,x_start:x_end].copy()
81
82                  cols, rows = p.shape[:2]
83                  # Make sure that we don't have some weird anomalies that get
                        threated as a puzzle piece
84                  if cols < 10 or rows < 10:
85                      continue
86                  rotM = cv2.getRotationMatrix2D((cols/2, rows/2), int(angle), 1)
87                  p = cv2.warpAffine(p, rotM, (cols, rows))
88                  self.__new_pieces.append(p)
```

4 Conclussie