Spot It! Mathematics through computer vision

J.-C. Canonne[1,5], M. Fratu[2], L. Vermeiren[3,5], E. Cartignies[4,5]
jean-charles.canonne@uphf.fr
(1) Université Polytechnique Hauts de France, LAMAV EA 4015
(2) Universitatea Transilvania din Brasov (ROMANIA)
(3) Université Polytechnique Hauts de France, LAMIH UMR CNRS 8201
(4) Université Polytechnique Hauts de France, IEMN UMR CNRS 8520
(5) IUT de Valenciennes

ABSTRACT: The Electrical Engineering and Computer Science Department of the institute of Technology of Valenciennes has been participating to Festival of Science for many years. Workshops about scientific subjects are organised in sessions of one and a half hour for groups of twelve participants. Participants come from different types of schools: from primary schools up to high schools. Workshops are designed, prepared and then animated by the students of the institute. The workshop that is presented in this article has been designed by 6 second year students as a project linked with a Computer Vision and Robotics module. Knowledge from the first-year module about Logic and Numeration (especially binary calculation) and more generally from the modules of Mathematics (especially homogenous coordinates for geometric transformations used in Robotics). The workshop is built around a popular matching cards game called "*Spot It!*" or *"Dobble"* depending on the countries. In the first part of the article the context of the experiences is described. In the second part, the game is described, a mathematical model is explained and allows the design of a simplified Spot It! game. The third part explains the method chosen to write the screenplay of the workshop, while part four gives the screenplay of the workshop as it results from the work of students.

Key words: Game, Dobble, Spot It!, Computer Vision, Modelling, Festival of Science, Mathematics.

1 EXPERIENCE CONTEXT

1.1 Hardware

Most experiments described in the article have been made on a Raspberry Pi 3. The Raspberry Pi Foundation is a UK based charity founded in 2009.This organization intends to help people all around the planet to access to digital world. To accomplish this goal, the Raspberry Pi Foundation has worked with academic, scientific and research community, and in particular with Cambridge University, for the development of a low cost and flexible hardware for experimenting with programming and electronics. This collaboration is at the origin of the Raspberry Pi family, and in particular the Raspberry Pi 3, which we have been working with.

fig 1: Raspberry Pi3

The Raspberry Pi is equipped with General Purpose Input Output pins which allow the use of a wide range of actuators and sensors. A huge number of low-cost projects based on Raspberry Pi has been imagined by makers and gives a source of inspiration for students: retrogaming, drone designing for example.
As for the camera, a Raspberry Picamera is used.

fig 2: Raspberry Picamera

The Picamera module has a Sony IMX219 8-megapixel sensor.

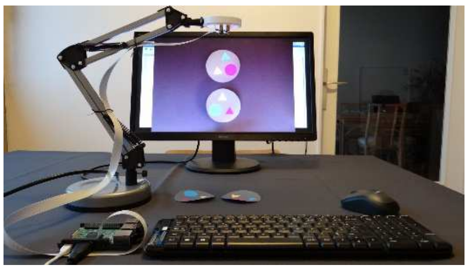
fig 3: The whole material.

1.2 Software

Python is an interpreted language created in 1991 by Guido von Rossom. Python interpreters are available for most operating systems, and in particular available for the Rasbian Strech version of Linux that is installed on Raspberry Pi3. Python supports oriented-object programming.

Free distributions of Python and of useful libraries for Python can have been found on the Internet. Among the libraries that are used for the work presented here are Numpy (for numerical work), Matplotlib (for graphical work), and OpenCV (for computer vision).

1.3 Reference books

This teaching experience of Computer Vision in the context of Mathematics teaching was first made by Mathematics teachers: they found great help in two books, one by Jain [1], the other by Gonzalez [2]. Giving students the habit of searching answers in these books was one of the challenges of the experience.

1.4 Students

The theater set of the experiences described here is a group of twelve students with no experience in computer vision. These students are given an initiation to computer vision in the context of the Festival of Science or University open days, and within courses on Robotics and Vision. A more precise description of the context can be found in [3]. One goal of the initiation is to show them that Mathematics are a useful tool and are used in domains they appreciate like image construction.

.

2 SPOT IT!

2.1 The official game

Spot It! is a matching game by Asmodee Company. This card game is sold under the name Dobble in Europe.

fig 4: Spot It! card game.

The game consists in 55 cards. There are 8 different symbols on each card. Two different cards have at the most one common symbol. The total amount of symbols used is 57.

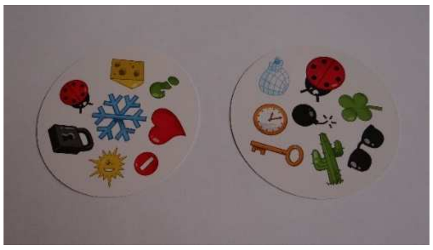

fig 5: Two cards matching with ladybird symbol

The paper used for the cards is a glossy paper which makes it difficult to take pictures of quality from the cards without an elaborate lighting.
Playing with fewer symbols and with less complicated symbols would allow to focus on main ideas about Computer Vision: students are supposed to be working with Computer Vision for the first time.
For these reasons, an adapted game of Spot It! has to be designed, which implies to think about the underlying mathematical model of the game.

2.2 The mathematical model

2.2.1 Questions

The main questions that arise when trying to design a Spot It! game find their answer from combinatorics. What is the minimal number of symbols required to design a Spot It! game? For a suitable number of symbols, what is the greatest number of cards that can be designed? A Spot It! game with such a number of cards will be said to be an optimal game.
In particular, it appears that the official game is not optimal and could contain 57 cards, satisfying then a more precise property: two different cards have exactly one common symbol.

2.2.2 Answers

Answers to these questions are exposed in [4] and [5].
The main answer is: "An optimal Spot It! game can be modelled by a finite projective space".
A projective space is a pair of two sets (P, D) with an incidence relation between elements P (called points) and elements of D (called lines). Furthermore, the incidence relation has to satisfy four axioms:
- (A1) Two different points belong to a unique line
- (A2) Two different lines have a unique common point
- (A3) Every line has at least three points.
- (A4) There exist three points at least that are not on a same line.

More precisely the model for Spot It! game is the finite projective plane called $P_2(F_7)$, that is the finite projective plane built upon the finite field F_7. The finite field F_7, also denoted $GF(7)$, for Galois Field of order 7, is the set of integers modulo 7:

$$F_7 = \{0,1,2,3,4,5,6\}$$

It corresponds to an affine plane $F_7 \times F_7$ of 49 points:

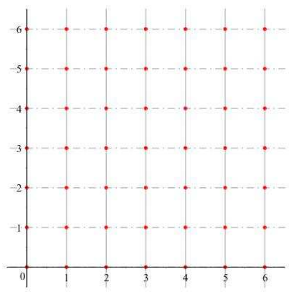

fig 6: The affine plane $F_7 \times F_7$.

$F_7 \times F_7$ is completed to obtain $P_2(F_7)$ with 8 ideal points (points at infinity) corresponding to the 8 possible directions for lines (slopes from 0 to 6 and vertical direction). Therefore $P_2(F_7)$ has 57 elements.
The correspondence between $P_2(F_7)$ and the game Spot It! is obtained taking Cards as Points and Symbols as Lines. For example, axiom (A1) gives "two different cards have a unique common symbol".

2.3 Design of a simplified game

2.3.1 The geometry of the simplified game.

The simplified game is built from the smallest projective plane, that is $P_2(F_2)$, the finite projective plane built from the set of integers modulo 2:

$$F_2 = \{0,1\}$$

The affine plan $F_2 \times F_2$ has 4 points:

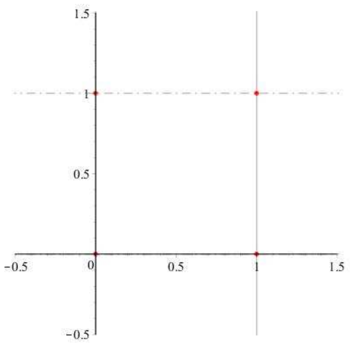

fig 7: The affine plane $F_2 \times F_2$.

The projective plan $P_2(F_2)$ has 7 elements: the 4 previous points and 3 ideal points

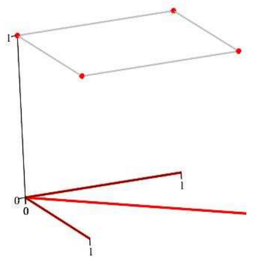

fig 8: Spatial representation of $P_2(F_2)$

The homogenous coordinates of the affine points are $p_1(0,0,1)$, $p_2(1,0,1)$, $p_3(0,1,0)$ and $p_4(1,1,1)$. The homogenous coordinates of ideal points are $u_1(1,0,0)$, $u_2(0,1,0)$ and $u_3(1,1,0)$.
There are 7 lines of three points: $d_1 = \{p_1, p_2, u_1\}$, $d_2 = \{p_3, p_4, u_1\}$, $d_3 = \{p_1, p_3, u_2\}$, $d_4 = \{p_2, p_4, u_2\}$, $d_5 = \{p_1, p_4, u_3\}$, $d_6 = \{p_2, p_3, u_3\}$ and $d_7 = \{u_1, u_2, u_3\}$.
The line d_7 is called ideal line or line at infinity.
The projective plan $P_2(F_2)$ is also called Fano plane and is currently represented with the following diagram:

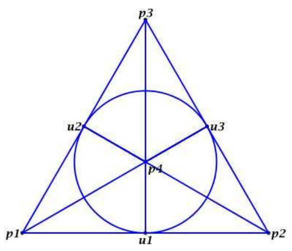

fig 9: Fano plane.$P_2(F_2)$

In this diagram, the ideal line is represented by a circle. Applying these results to the problem of the construction of a simplified Spot It! game indicates that 7 symbols (lines) are needed to design 7 cards (points) with 3 different symbols on each card (three lines through each point).

2.3.2 The choice of the symbols

The choice of symbols is made to allow both a work on colour representation and a work on shapes.
Choosing 7 different colours would imply that the work on shapes is useless, and would also imply a very performant lighting. On the other hand, one colour would imply a very poor work on colours (only colour for background and colour for symbols). A choice of three different colours has been made.
The final selection is:
- Yellow (with a maximal intensity in Red and in Green, and an intensity of 0 in Blue in RGB description)
- Violet (with a maximal intensity in Red and in Blue, and an intensity of 0 in Green in RGB description)
- Cyan (with a maximal intensity in Green and in Blue, and an intensity of 0 in Blue in RGB description)
- Dark Grey for the background (with an intensity of a third of maximal intensity in Red, Green, and Blue in RGB description)
Three basic shapes have been chosen for symbols: circle, square and triangle. The calculation of moments with OpenCV makes it easy to separate those shapes. In a purpose of simplification, only the moments of order 0, i.e. the areas of the symbols are used in the project. It is made possible by choosing constant sizes for the different symbols: a big circle, a medium square and a smaller triangle. This condition is a very restrictive condition, but is necessary for a computer vision workshop that is supposed to last one hour and a half.

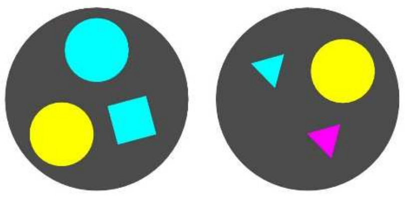

fig 10: Simplified cards

The following diagram shows the placement of symbols on the cards according to the model.

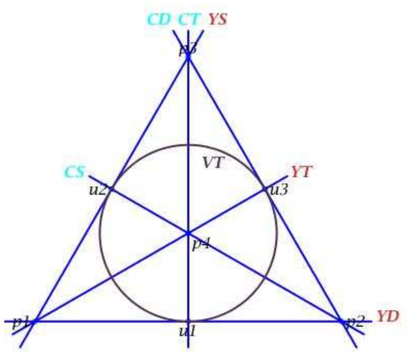

fig 11: Fano representation of cards.

For example, the point *p3* is the card "Cyan Disk, Cyan Triangle, Yellow Square" as this point is the intersection of these three lines.

3 WRITING THE PEDAGOGICAL SEQUENCE

3.1 The DARE method

Students have to organize the workshop using the pedagogical DARE method developed by the authors [6]. DARE is the acronym for Discover, Apply and Resolve. This method can be perceived as a rewriting, for computer languages, of the classical three-steps tests developed for language teaching: first translating from foreign language to native language, then translating from native language to foreign language and last writing an essay in foreign language.
The Discover step is devoted to translating computer language (here Python and OpenCV instruction) into native language. Workshop participants are given scripts they have to run and comment. In the Apply step, workshop participants have to answer very simple questions using the instructions seen in the first step: they simply translate basic works into computer language. The last step Resolve is the most complicated but also most interesting step: workshop participants must use their judgement, creativity and skills to solve a specific problem, in link with previous steps.

3.2 Writing the screenplay of the workshop.

First thing to do when writing the screenplay of the workshop is to begin by the end, that is here, playing Spot It! with the Raspberry, i.e. "Matching symbols between two cards". Students have to resolve the problem, compare their solutions and draw up a list of the different instructions used. This is the" Resolve" part. Those instructions correspond to the different steps of the resolution of the problem: these steps will give the subjects of the different exercises of the "Apply" part. Once again students have to propose a list of exercises for each subject, solve them and then write scripts corresponding to solutions of similar exercises. Those solutions give the "Discover" part.
Three main steps can be established in the resolution of the problem.
The first mathematical object encountered is the colour BGR image.

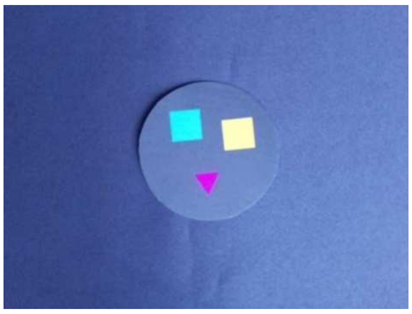

fig 12: BGR image of a card.

This image is converted in the HSV format. In both formats a colour image can be thought as three greyscale (or intensity) images

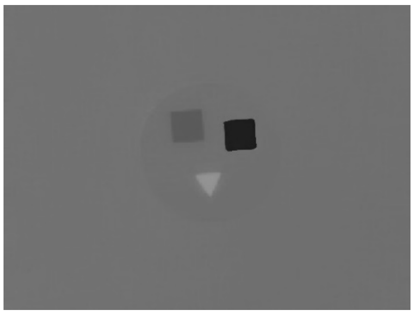

fig 13: Greyscale image (Hue component)

These greyscale images are usually transformed in **black and white** images using a threshold method.

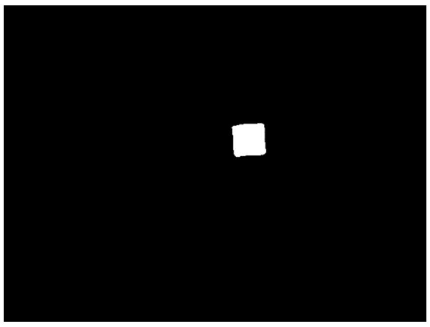

fig 14: Black and White after Thresholding

The OpenCV function findContours has then to be used to find blobs, that is white connected parts, in the image. Other functions allow the calculation of the moments of those blobs, and in particular the calculation of their area.

Three main steps have been obtained for our pedagogical sequence: first Black and White images, then Greyscale images, and finally Colour images.

4 THE WORKSHOP

4.1 Black and White images

Discover 1: Given the following 7*16 pixels image:

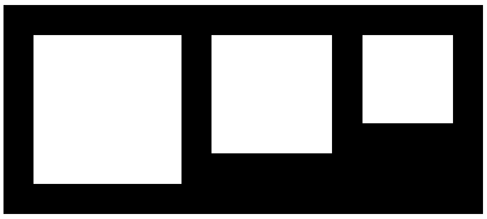

fig 15: Image nb.bmp

execute the following program:

```
# Program blobs finding
import cv2
import numpy as np
# Printing the image Img
Img = cv2.imread('nb.bmp',0)
print ('Img = ',Img)
# Printing main facts about the image Img
print ('Dimensions of the image = ' , Img.shape)
print ('Number of white pixels =',np.count_non-
zero(Img))
# Finding the blobs of the image Img
ye,contours,hi=cv2.findContours(Img, -1 , 1 )
number_of_blobs= len(contours)
print('Number of blobs = ', number_of_blobs)
# Areas of the blobs
for i in range(0,number_of_blobs):
  print('Aire blob’, i ,'=',cv2.contourArea(contours[i]))
```

Comment the answers.

Apply 1: Apply the preceding program to the image NB.jpeg

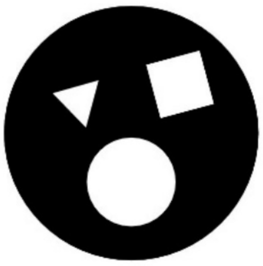

fig 16: Image NB.jpeg

4.2 Greyscale images

Discover 2: Given the following 7*16 pixels image:

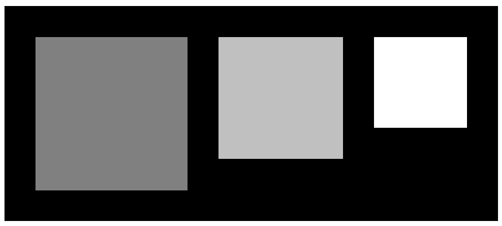

fig 17: Image gs.bmp

execute the following program:

```
# Program pixels selection
import cv2
import numpy as np
Img = cv2.imread('gs.bmp',0)
cv2.namedWindow('GS',0)
cv2.imshow('GS',Img)
cv2.waitKey(0)&0xFF
min=180
max=200
selection = cv2.inRange(np.array(Img),np.ar-
ray(min),np.array(max))
cv2.namedWindow('Threshold',0)
cv2.imshow('Threshold',selection)
```

Comment the answers.

Apply 2: Apply the preceding program to the image GS.jpeg

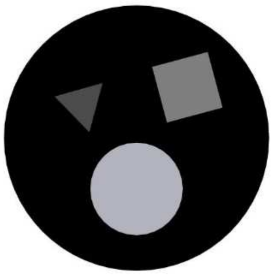

fig 18: Image GS.jpeg

4.3 Colour images

Discover 3: Given the following 7*16 pixels image:

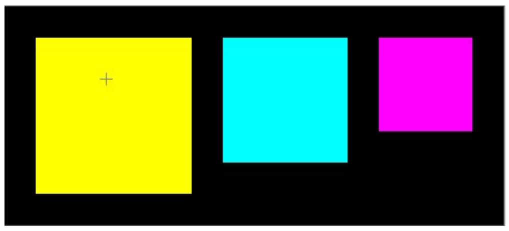

fig 19: Image co.bmp

execute the following program:

```
# Finding the hue component
import cv2
import numpy as np
Img = cv2.imread('co.bmp',-1)
cv2.namedWindow('co2',0)
cv2.imshow('co2',Img)
hsv = cv2.cvtColor(Img,cv2.COLOR_BGR2HSV)
hue,sat,val = cv2.split(hsv)
cv2.namedWindow('hue',0)
cv2.imshow('hue',hue)
cv2.imwrite('hue.bmp',hue)
```

Comment the answers.

Apply 3: Apply the preceding program to the image CO.jpg

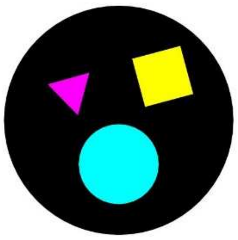

fig 16: Image CO.jpeg

4.4 Taking pictures

The following program explains how to get images from the Picamera:

```
import cv2
import time
cv2.namedWindow('view of the cards)
cv2.resizeWindow('view of the cards',480,640)
# capturing the video flux from the picamera
cap = cv2.VideoCapture(-1)
while(1):
   success,frame= cap.read()
   if success==True :
      cv2.imshow(view of the cards,frame)
  # esc to escape from the loop
  k = cv2.waitKey(5) & 0xFF
  if k == 27:
     break
```

4.5 Resolve

The following challenge is proposed at the end of the workshop:
"Two cards are placed under the Picamera.
Write a program that indicates the name of their common symbol"

5 CONCLUSION

This workshop has been proposed to pupils from high-school visiting the University in June 2019. The work of our students has been appreciated by pupils and their teachers who specially enjoyed making Mathematics in an applied context. Students have appreciated the modelling activity of the project, and some of them have extended their work discovering the links between error correcting codes and finite projective plans. Maybe the subject for another article!

Bibliography

[1] Jain, A K, « Fundamentals of Digital Image Processing », Prentice Hall, ISBN 0-13-336165-9, 1989.

[2] Gonzalez R.C., Woods R.E., « Digital Image Processing », 3rd ed, Prentice Hall, Upper Saddle River, NJ, 2008.

[3] Bécar, J.-P., Canonne, J.-C., Vermeiren, L., Taleb, A., « How robotics vision and computer vision improve interdisciplinarity », Proceedings of 10th annual International Conference on Education and New Learning Technologies, Barcelona (Spain),2nd-4th of July, 2018

[4] Hézard M., Hézard D. « Jouons un peu…à DOBBLE », Quadraure n°27 (2013), 20-21

[5] Bourrigan M., "Dobble et la géométrie finie", Image des Mathématiques, CNRS, 2011. http://images.maths.cnrs.fr/Dobble-et-la-geometrie-finie.html

[6] Bécar, J.-P., Robert, F., Canonne, J.-C., Vermeiren, L., Cartignies, E, « A method to connect mathematics and sciences using a computer algebra system », Proceedings of 9th annual International Conference on Education and New Learning Technologies, Barcelona (Spain),3rd-5th of July, 2017