Inteligencia Artificial

Juan Pablo Garcia Gallego, Luis Alberto Borrero Velez

Ingeniería de Sistemas y Computación, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: juanpagg11@gmail.com

Resumen
La realización de este proyecto consiste en un software en lenguaje de programacion Python en su versión 2.7 para el analisis y posible traslación de imagenes de laberintos para luego ser solucionados.

I. INTRODUCCIÓN

Se uso la librería NumPy que es una modulo de Python 2.7 con una biblioteca de funciones matemáticas de alto nivel, también se integro la librería OpenCV que esta diseñada para la visión artificial por medio de estos modulos se espera realizar las actividades a tener como objetivo y como editor de texto se utilizo SublimeText.

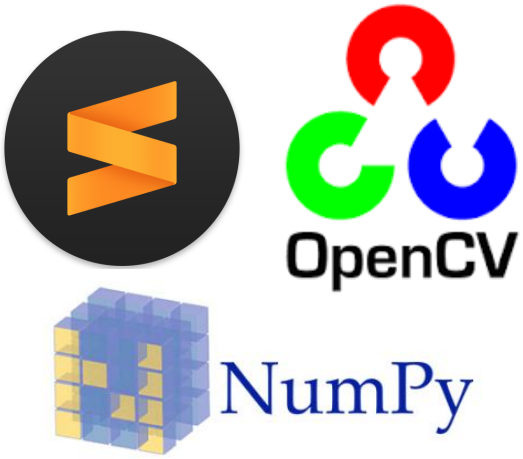

Imagen 1

En la imagen 2 se puede apreciar el funcionamiento general del software.

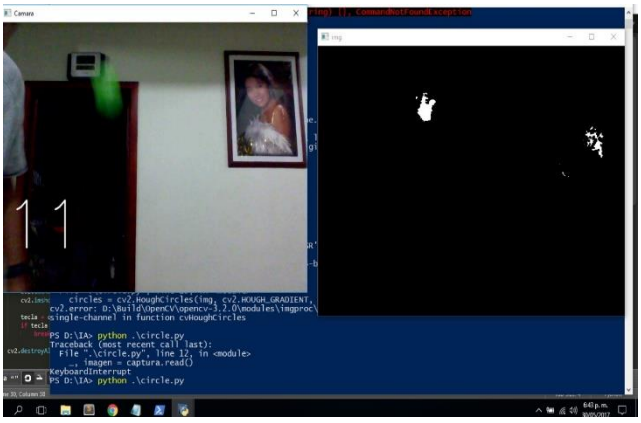

Imagen 2

II. CONTENIDO

Para la realización de este software se tienen los siguientes archivos:

1. ImaProcessor.py

Este archivo tiene como función el procesar la imagen haciendo uso de la librería OpenCV aplicando filtros o transformaciones.

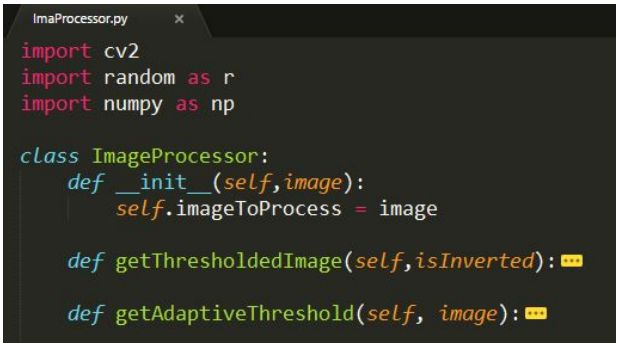

Imagen 3

```
def getThresholdedImage(self,isInverted):
    threshType = cv2.THRESH_BINARY_INV if isInverted else cv2.THRESH_BINARY
    retval, threshold = cv2.threshold(self.imageToProcess,127,255,threshType)
    return threshold

def getAdaptiveThreshold(self, image):
    return cv2.adaptiveThreshold(image,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C ,\
                                 cv2.THRESH_BINARY,21,0)
```

Imagen 4

Con estas funciones aplica un filtro a gris para que la imagen puede clasificarse en tonos blancos o negros, asi mismo un filtro de suavizado para mejorar la definición.

2. MazSolver.py

Este seria el archivo principal pues sera la encargada de tomar la imagen como matriz con sus datos y buscar la solucion para el respectivo laberinto.

```
import cv2
import numpy as nps
import Queue as q

class MazeSolver:

    def __init__(self,proccessedImage,granularity):
        self.image = proccessedImage
        self.height,self.width = proccessedImage.shape[:2]
        self.nodes_to_visit = q.PriorityQueue()
        self.node_distances = {}
        self.parent_nodes = {}
        self.visited_nodes = {}
        self.GRANULARITY = granularity
```

Imagen 6

Como se presencia en la anterior imagen tambien haremos uso del modulo Queue que es la respretancion de la estructura de datos.

```
def solveMaze(self,start_x,start_y,end_x,end_y):
    i = 0
    #setup
    self.start_x = start_x
    self.start_y = start_y
    self.end_x = end_x
    self.end_y = end_y
    curr_node = Node(start_x,start_y,None)
    end_node = Node(end_x,end_y,None)
    self.node_distances[curr_node.position] = 0
    self.parent_nodes[curr_node.position] = None
    self.nodes_to_visit.put((curr_node.get_priority(end_node,0),curr_node))
    while curr_node.position!=end_node.position and not self.nodes_to_visit.empty():
        if(i%10000 == 0):
            print "Iteracion {0} completa".format(i)
        curr_node = self.nodes_to_visit.get()[1]
        for node in self._get_new_nodes(curr_node):
            distance = self.node_distances[curr_node.position] +self.GRANULARITY;
            try:
                if distance < self.node_distances[node.position]:
                    priority = node.get_priority(end_node,distance)
                    self.node_distances[node.position] = distance
                    self.nodes_to_visit.put((priority,node))
                    self.parent_nodes[node.position] = curr_node
            except KeyError:
                priority = node.get_priority(end_node,distance)
                self.node_distances[node.position] = distance
                self.nodes_to_visit.put((priority,node))
                self.parent_nodes[node.position] = curr_node
        self.visited_nodes[curr_node.position] = True
        i+=1
    if curr_node.position == end_node.position:
        print("Algorithm completed. Total iterations: {0}".format(i))
        return self._get_solution_list(curr_node)
    else:
        print("No solution found")
        return False
```

Imagen 7

La anterior imagen muestra lo que es la funcion principal la encargada de realizar la solución del laberinto, esta va recorriendo la imagen por medio de los pixeles como si fuera una matriz teniendo en cuenta las zonas negras que serian los muros.

2. Setup.py

En este archivo es donde se unen los dos anteriores para conformar el programa completo para la resolución de laberinto.

```
import numpy as np
import cv2
import sys
from ImageProcessor import ImageProcessor
import MazeSolver

MAZE_NAME = "Maze Display Window"
point = (-1,-1)

def setupWindow():
    filename = "Maze1.png"
    imageProcessor = ImageProcessor(cv2.imread(filename,0))
    colourImage = cv2.imread(filename,1)
    image = imageProcessor.getThresholdedImage(False)
    granularity = imageProcessor.get_granularity(image, 100)
    print("Granularidad: {0}".format(granularity))
    start_x,start_y,end_x,end_y = get_start_points(image)
    image = imageProcessor.encloseMaze(image)
    mazerunner = MazeSolver.MazeSolver(image,granularity)
    solution = mazerunner.solveMaze(start_x,start_y,end_x,end_y)
```

Imagen 8

III. CONCLUSIONES

- Por medio de este proyecto se logro el aprender mas sobre estos modulos que tienen tantas herramientas y muchas utilidades, asi mismo el lenguaje de Python el que cual es muy simple pero de alto nivel.

- El aprender a ver las imagenes como una matriz y asi mismo poder aplicar diferentes operaciones a estas buscando obtener los resultados esperados

REFERENCIAS

http://opencv.org/

http://www.numpy.org/

http://sourceforge.net/projects/opencvlibrary/

https://pypi.python.org/pypi/opencv-python

http://acodigo.blogspot.com.co/p/tutorial-opencv.html