**Rompimiento de CAPTCHA’s basados en texto mediante rotación o rectificación usando un OCR**

Edson Caqui Adriano, Josué Julcarima Calle, Elder Livisi Carbajal

Asignatura: Inteligencia Artificial

Facultad de Ingeniería Industrial y de Sistemas

Escuela Profesional de Ingeniería de Sistemas

Universidad Nacional de Ingeniería

concretos con ciertos tipos de CAPTCHA’s, después se describe el método de solución propuesto, sigue los resultados obtenidos y por último las conclusiones y recomendaciones de técnicas de optimización y estudios posteriores.

La librería Tesseract OCR fue inicialmente creada por HP y los últimos años ha sido mantenida por Google [5], se encuentra bajo licencia Apache y su código está publicado en github.

**PRESENTACIÓN DEL PROBLEMA**

El CAPTCHA se propuso como un paliativo a la utilización de bots. La comprobación de que los avances en Visión Computacional pueden ayudar a romper varios esquemas de CAPTCHA basados en texto es una tarea parecida a encontrar vulnerabilidades de día cero en el hecho de que el que lo logra primero puede aprovecharse del conocimiento.

La tendencia en CAPTCHA’s hoy en día ha decantado hacia el uso de esquemas basados en clasificación de imágenes, análisis de comportamiento del usuario (ej.: movimiento del mouse en la pantalla) y consistencia en el uso de las diversas páginas web que visita [3].

Sin embargo existe una minoría de páginas web que permanece usando esquemas basados en texto (SUNAT, sitios con recaptcha v1, etc.).

Los enfoques para comprobar la efectividad de los CAPTCHA’s basados en texto suelen basarse en segmentación de caracter por caracter y luego reconocimiento de cada caracter mediante un OCR. En [1] se aborda el problema hallando las posibles segmentaciones para el primer caracter y según el nivel de confidencia que se va obteniendo se va realizando nuevas segmentaciones y obteniendo un árbol en el cual cada nodo es una posible segmentación, al final se obtiene una confidencia en suma para cada hoja del árbol y se escoge la que tenga la mayor confidencia en suma.

**Resumen**

El completamente automatizado test de Turing público para diferenciar humanos de computadoras, también llamado CAPTCHA por sus siglas en inglés, es un método que se ha utilizado ampliamente sobre todo en su versión basada en texto. En el presente documento se describe una técnica para acelerar la resolución de CAPTCHA’s rotando la imagen hasta que el OCR obtenga los mejores resultados posibles. Se incluyen un pre-procesamiento para mejorar el reconocimiento incluyendo la binarización de la imagen.

**INTRODUCCIÓN**

La validación de que los servicios orientados a ser usados por humanos bajo una manera no indiscriminada es un trabajo que ha recaído en los CAPTCHA y estos han servido a su vez para alimentar algoritmos de Machine Learning especializados en OCR y reconocimiento de imágenes como en el caso de reCaptcha v1 y v2 respectivamente.

El costo que implica que una persona tenga el control de muchas cuentas o bots puede llegar a ser muy alto, desde el costo del almacenamiento gratuito en la nube que se brinda por cuenta (ej.: Google Drive, One Drive, etc.) hasta la manipulación de los medios de comunicación cercanos a nuestro día a día (Facebook, Twitter, etc.), pasando por el spam masivo, el hacer trampas en videojuegos y programas de recompensa por referencias a amigos.

El estado del arte respecto al rompimiento de CAPTCHAS basados en texto es muy amplio, se tienen hasta métodos genéricos que pregonan resolver cualquier CAPTCHA basado en texto [1] sin necesidad de reconfiguración del algoritmo dependiendo del esquema de CAPTCHA. Este último método logró una precisión superior a la mínima requerida para considerar un esquema de CAPTCHA como inseguro (umbral del 1%) según [2].

El presente documento empieza con la descripción del problema a atacar, luego se plantean objetivos

Búsqueda ternaria tiene una complejidad de donde:

* L: longitud del intervalo que se toma para realizar la búsqueda.
* e: precisión requerida.

A continuación se definen F y H:

* F(a):

Sea f(a) la función de confidencia en relación al ángulo de rotación ‘a’ de la imagen se puede deducir que f posee al menos un máximo local.

Dado que al ejecutar la búsqueda ternaria en la función f no necesariamente se obtiene el máximo absoluto dado que se pueden tener varios máximos relativos se define la siguiente función F con la que nos aseguramos de hallar picos con al menos un umbral “u” de confidencia:

* H(a):

Una manera más heurística de encontrar el mejor ángulo de rotación que no requiere de ejecutar el OCR para cada ángulo a probar es maximizando la entropía, la cual se define como sigue:

Donde:

* + h(i, a): frecuencia relativa de la cantidad de pixeles pertenecientes al texto cuya proyección sobre la recta *L* rotada “a” grados que pasa por el centro de masa pertenece al intervalo “i”. Todos los intervalos tienen una longitud constante.

En resumen el primer método propuesto es el siguiente:

1. *Leer la imagen del CAPTCHA y guardarla en “im”*

*(295)*

1. *Aplicar refinamientos y filtros a “im”*
2. *Binarizar “im”*

**OBJETIVOS**

**Objetivo general**:

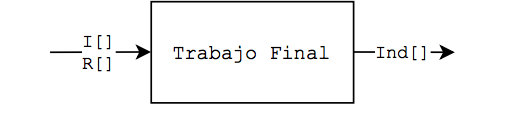
Obtener una precisión de reconocimiento mayor al 1% para imágenes de CAPTCHA’s obtenidas de manera aleatoria.

**Objetivos específicos:**

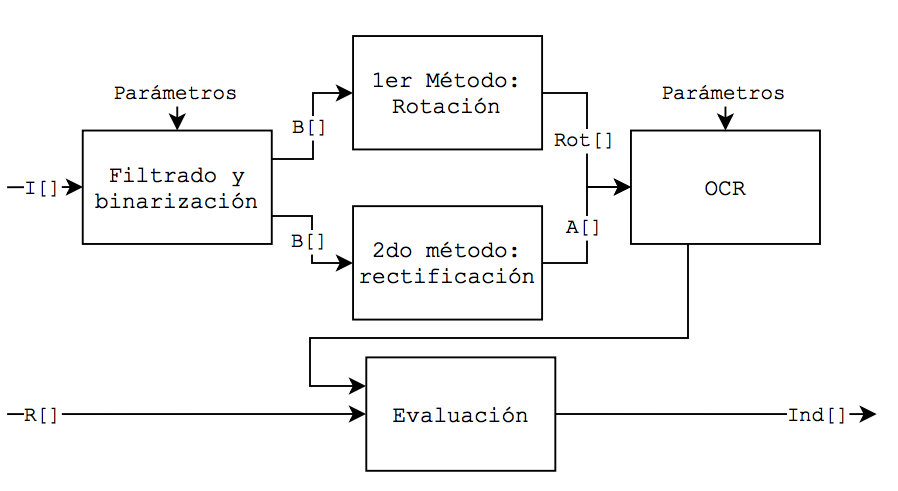
1. Reconocer CAPTCHA’s usando Tesseract.
2. Encontrar el ángulo de rotación de la imagen con el que se obtiene la mayor entropía.
3. Encontrar el ángulo de rotación de la imagen con el que se obtiene la mayor confidencia.
4. Rectificar la imagen encontrando la función de la curva mediante regresión.

**Descripción de la solución**

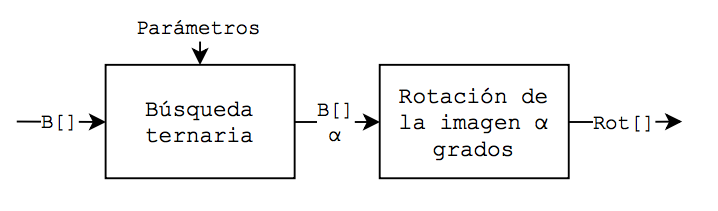
El esquema seguido para desarrollar los distintos métodos de solución propuestos es el siguiente:



Donde el bloque “Trabajo Final” se compone así:



**Primer método: Rotación**

****

La primera solución propuesta ataca el problema de encontrar el ángulo de rotación que maximice una función (dada como parámetro).

Dicha función es un parámetro de la búsqueda ternaria que podrá tomar el valor de F(α) o bien de H(α), ambas funciones definidas más adelante.

Es por ello que el algoritmo que se propone para optimizar la búsqueda de el/los ángulos de rotación óptimos es una trivial búsqueda ternaria.

Al inicio un ejemplo de las imágenes binarizadas que se tienen es:

D:\Google Drive\2015-2\Inteligencia artificial\Grupo Inteligencia Artificial\Captcha breaking\scripts\newJob\0afterBinarization.png

1. Cierre y erosión.- Se aplica la transformación morfológica del cierre previamente ampliando los bordes de la imagen para no restringirnos cuando la longitud del cierre es mayor que la longitud del margen de las letras de la imagen.

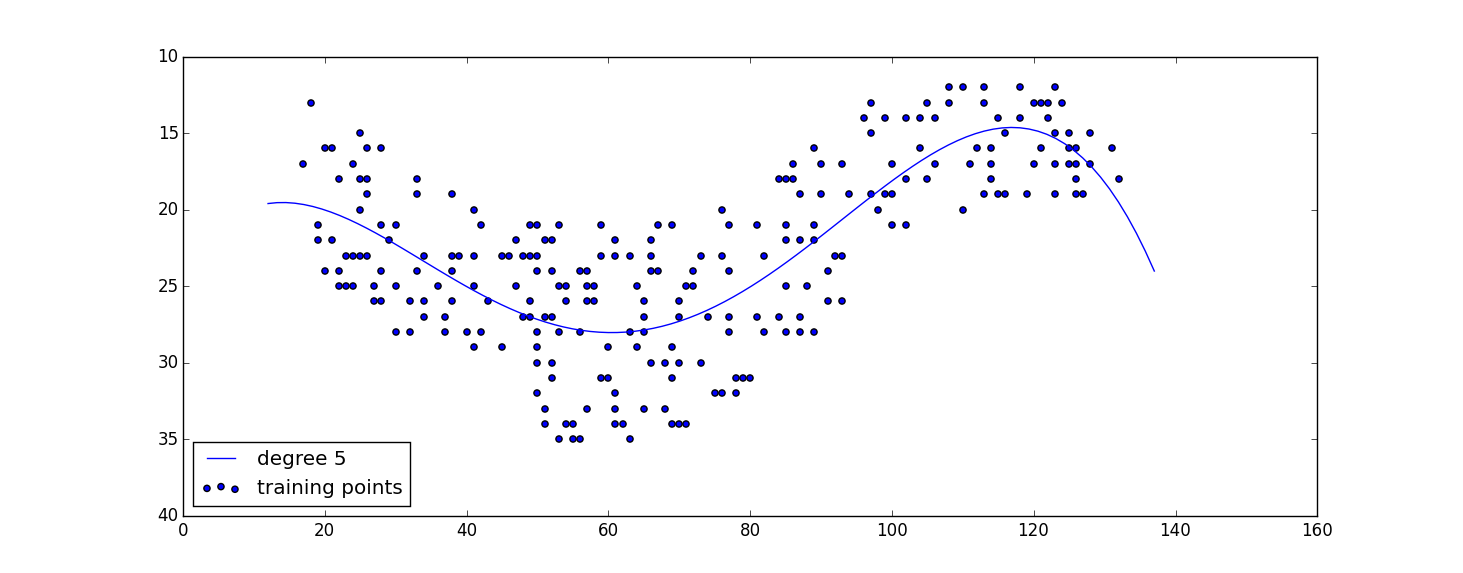
Después del cierre:

D:\Google Drive\2015-2\Inteligencia artificial\Grupo Inteligencia Artificial\Captcha breaking\scripts\newJob\1afterClosing.png

Después de la erosión:

D:\Google Drive\2015-2\Inteligencia artificial\Grupo Inteligencia Artificial\Captcha breaking\scripts\newJob\2afterEroding.png

1. Regresión Ridge.- Se escoge una muestra de 250 puntos al azar para entrenar un polinomio de grado 5 mediante el método de Ridge.



1. Alineación.- Se calcula la media de las alturas de los puntos y se trasladan todas las columnas de puntos para alinearse con la media.

D:\Google Drive\2015-2\Inteligencia artificial\Grupo Inteligencia Artificial\Captcha breaking\scripts\newJob\3afterAlignment.png

El paso del OCR Tesseract sobre esta última imagen obtiene el siguiente resultado: “DWSBHR”.

1. *Realizar una búsqueda ternaria para encontrar el ángulo en el que la función H sea máxima y luego volver a realizar la búsqueda en el intervalo a la derecha y luego a la izquierda del punto encontrado para cubrir posibles picos no tomados en cuenta.*

F:\Google drive\2015-2\Inteligencia artificial\Grupo Inteligencia Artificial\Captcha breaking\scripts\rotated\295.png*(295)*

*Resultado del OCR: “I41ARN”*

1. *Realizar búsqueda ternaria para encontrar el ángulo en el que la función F sea máxima*

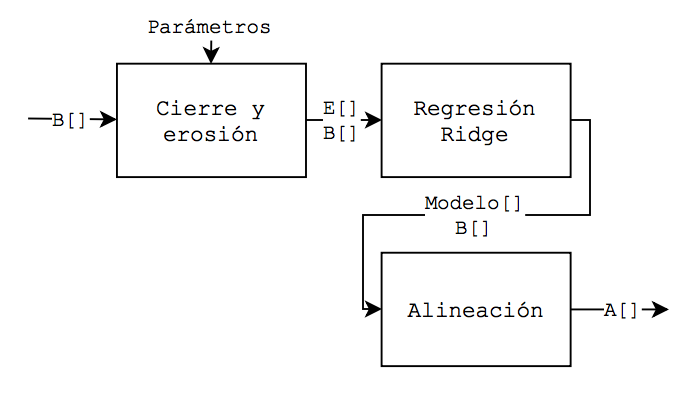
*F:\Google drive\2015-2\Inteligencia artificial\Grupo Inteligencia Artificial\Captcha breaking\scripts\r3_295.png*

*Resultado del OCR: “I41ARN”*

Detalle del algoritmo de búsqueda ternaria [4]:

1. *left = 0°*
2. *right = 360°*
3. *Mientras True:*
4. *Si abs(right – left) < e:*
5. *Retornar (left + right)/2*
6. *leftThird = left + (right – left)/3*
7. *rightThird = right – (right – left)/3*
9. *Si(F(leftThird)>F(rightThird)):*
10. *right = rightThird*
11. *sino:*
12. *left = leftThird*

**Segundo método: Rectificación**



**Segundo método: Alineación**

Confianza promedio 65.46%

Precisión 26.75%

Los resultados completos de la aplicación de todos los métodos expuestos se encuentra en el archivo resultados.xlsx.

**CONCLUSIONES**

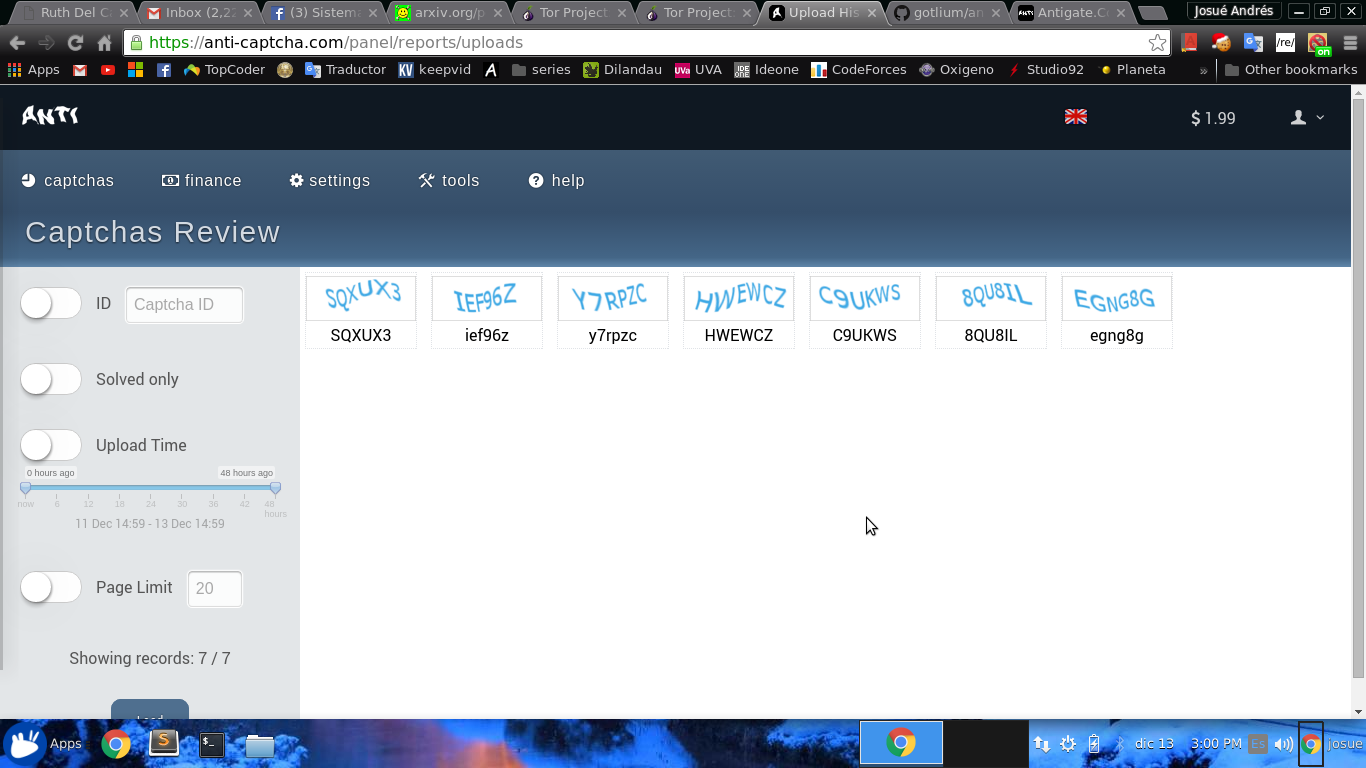
* Se observa que para la muestra tomada de captchas con características ondulaciones el método que logra mejor resultado es el de rotación (maximización de confianza).
* Para no sobrealimentar el modelo se tomaron puntos al azar lo cual permitió mejorar la curva generada.
* Los captchas basados en texto en general ya cayeron en desuso debido a la molestia que presenta al usuario y a la existencia de métodos más precisos como la clasificación de imágenes e reconocimiento de patrones del usuario. Incluso existen métodos que aprovechan el poder computacional del usuario pidiéndole resolver un problema matemático (incluso ayudando a minar bitcoins, quien sabe).

**RECOMENDACIONES**

* Para mejorar el segundo método (rectificación) se recomienda aplicar longitud de arco y entrenar no solo un polinomio sino hacer que el primer polinomio divida la imagen y entrenar un polinomio por cada una de las dos imágenes generadas.

**RESULTADOS**

Para obtener los resultados mediante seres humanos se utilizó el servicio de anti-captcha.com.



Estos resultados se encuentran en la pestaña “usando seres humanos” del archivo resultados.xlsx.

**Primer método: Rotación**

* Maximizando la entropía (H):

Confianza promedio 49.69%

Precisión 10.25%

* Maximizando la confianza (F):

Confianza promedio 71.36%

Precisión 32.40%

**BIBLIOGRAFÍA**

[1] Bursztein, E., Aigrain, J., Moscicki, A., & Mitchell, J. C. (2014, August). The end is nigh: Generic solving of text-based captchas. In 8th USENIX Workshop on Offensive Technologies (WOOT 14). USENIX Association.

[2] E. Bursztein, M. Martin, and J. Mitchell. Text-based captcha strengths and weaknesses. In Proceedings of the 18th ACM conference on Computer and communications security, CCS ’11, pages 125–138, New York, NY, USA, 2011. ACM

[3] CAPTCHA technologies Web Usage Statistics. (2015, November 28). Retrieved November 28, 2015, from <http://trends.builtwith.com/widgets/captcha>

[4] Ternary search. (n.d.). Retrieved November 28, 2015, from <https://en.wikipedia.org/wiki/Ternary_search>

[5] Smith, R. (2007, September). An overview of the Tesseract OCR engine. In icdar (pp. 629-633). IEEE.

**ANEXOS**

El código del primer método se encuentra en los archivos process1.py y process3.py (maximizando H y F respectivamente).

El código del Segundo método realizado en Python (los nombres de variables se pusieron en inglés debido a que se piensa publicar el código en github y es la mejor forma de lograr más impacto):

main.py

# importing needed libraries and functions

import os

from skimage import io

# importing project files

import preparation

import rectification

import runParameters

showProcess = runParameters.showProcess

for i in range(runParameters.start, runParameters.end + 1):

runParameters.stepNumber = 0

filename = os.path.join("../samples/"+str(i)+'.png')

if showProcess:

print(filename)

B = preparation.cleanAndBin(filename)

R = rectification.rectificate(B)

resFile = os.path.join("results/"+str(i)+'.png')

io.imsave(resFile, R\*255)

preparation.py

from skimage import io

from skimage import filters

from skimage.color import rgb2gray

from skimage.viewer import ImageViewer

import runParameters

showProcess = runParameters.showProcess

def setShowProcess(val):

global showProcess

showProcess = val

def cleanAndBin(filename):

im = io.imread(filename)

# remove borders to mask

im = im[4:-4,4:-4]

im = rgb2gray(im)

val = filters.threshold\_otsu(im)

mask = im < val

if showProcess:

#io.imshow(mask)

viewer = ImageViewer(mask)

viewer.show()

io.imsave(str(runParameters.stepNumber) + "afterBinarization.png", mask\*255)

runParameters.stepNumber += 1

return mask

rectification.py

from skimage import io

from skimage.viewer import ImageViewer

import numpy as np

from skimage.morphology import binary\_closing

from skimage.morphology import binary\_erosion

import matplotlib.pyplot as plt

from sklearn.linear\_model import Ridge

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make\_pipeline

import runParameters

showProcess = runParameters.showProcess

def setShowProcess(val):

global showProcess

showProcess = val

def close(B):

neiDia = 20 # neighborhood square side

# expanding image

C = [0] \* (2 \* neiDia + len(B[0]))

C = [C] \* (2 \* neiDia + len(B))

C = np.array(C)

C[neiDia:-neiDia, neiDia:-neiDia] = B[:, :]

# creating a 2d array of ones

ones = [1] \* neiDia

ones = [ones] \* neiDia

C = binary\_closing(C, ones)

# returning to original size

C = C[neiDia:-neiDia, neiDia:-neiDia]

if showProcess:

viewer = ImageViewer(C)

viewer.show()

io.imsave(str(runParameters.stepNumber) + "afterClosing.png", C\*255)

runParameters.stepNumber += 1

return C

def erode(C):

eroDia = 10

# creating a 2d array of ones

ones = [1] \* 1

ones = [ones] \* eroDia

E = binary\_erosion(C, ones)

if showProcess:

viewer = ImageViewer(E)

viewer.show()

io.imsave(str(runParameters.stepNumber) + "afterEroding.png", E\*255)

runParameters.stepNumber += 1

return E

def getPoints(E):

vals = []

for x in range(len(E)):

for y in range(len(E[x])):

if E[x, y]:

vals.append([x, y])

points = np.array(vals, dtype='d')

if showProcess:

print( "# of points: " + str(len(points)))

print( "max row: " + str(np.max(points[:,0])) )

print( "min row: " + str(np.min(points[:,0])) )

print( "max col: " + str(np.max(points[:,1])) )

print( "min col: " + str(np.min(points[:,1])) )

return points

def getTrainedModel(points):

degree = 5 # degree of function to fit

maxSampleSize = 250

sampleSize = min(maxSampleSize, len(points))

# getting train data

rng = np.random.RandomState()

rng.shuffle(points)

y = points[:sampleSize,0]

x = points[:sampleSize,1]

# create matrix versions of these arrays

X = x[:, np.newaxis]

# ridge interpolation:

# http://scikit-learn.org/stable/auto\_examples/linear\_model/plot\_polynomial\_interpolation.html

model = make\_pipeline(PolynomialFeatures(degree), Ridge())

model.fit(X, y)

if showProcess:

# generate points used to plot

x\_plot = np.linspace(np.min(x) - 5, np.max(x) + 5, 100) # de 0 a 200 inclusive tomando 100 puntos

X\_plot = x\_plot[:, np.newaxis]

y\_plot = model.predict(X\_plot)

plt.scatter(x, y, label="training points")

plt.plot(x\_plot, y\_plot, label="degree %d" % degree)

plt.legend(loc='lower left')

plt.gca().invert\_yaxis()

plt.show()

return model

def align(B, me, model):

A = [0] \* (len(B[0]))

A = [A] \* (len(B))

A = np.array(A)

offset = me[0]

for i in range(len(B[0])):

y = model.predict([[i]])

for j in range(len(B)):

pj = j + offset - y # projection

pj = (int)(pj)

if pj >= 0 and pj < len(B):

A[pj, i] = B[j, i]

if showProcess:

viewer = ImageViewer(A.astype(bool))

viewer.show()

io.imsave(str(runParameters.stepNumber) + "afterAlignment.png", A\*255)

runParameters.stepNumber += 1

return A

def rectificate(B):

C = close(B)

E = erode(C)

points = getPoints(E)

model = getTrainedModel(points)

A = align(B, np.mean(points, axis = 0), model)

return A

runParameters.py

showProcess = True # show and write intermediate processes images to files

start = 1924 # start index to process images

end = 1924 # last index to process images

# used for numbering process images

stepNumber = 0