**BREAKING A VISUAL CAPTCHA: A NOVEL APPROACH USING HMM**

**Abhay Bansal, Divye Garg, Anup Gupta, Anand Gupta**  
Department of Computer Engineering, NSIT, New Delhi, India  
abhaybansal_371@yahoo.co.in, divyegarg@gmail.com, anuprocks@gmail.com, anand@coe.nsit.ac.in

**ABSTRACT**

Completely Automated Public Turing Tests to Tell Computers and Humans Apart (CAPTCHAs) are the automatic filters that are widely used these days to disallow any automated script that can perform the work of a human. CAPTCHAs are built in such a way that it is very difficult for any automated script to break them. In this paper, a novel approach for EZ-Gimpy CAPTCHAs has been proposed that first preprocesses the given CAPTCHA, segments its characters, matches the character’s features with the features in a characteristic table and then uses HMM to recognize the characters of the CAPTCHA. Results verify the effectiveness of the approach.

**Index Terms**— CAPTCHA, HIP, EZ-Gimpy, HMM.

1. **INTRODUCTION**

CAPTCHAs are also known by the name Human Interactive Proofs (HIPs) [1]. They are widely used these days to prevent any script that is automated to perform an activity that is meant for a human. They are basically used to prevent bots which use these scripts to make false accounts at the websites. CAPTCHAs can be categorized into two types mainly: - (i) Visual CAPTCHA and (ii) Audio CAPTCHA. Audio CAPTCHAs have characters written on a complex background like Visual CAPTCHAs but they also read out the characters which is helpful for a blind person or in case some character cannot be recognized. An example of Audio CAPTCHA is shown in figure 1.

Figure 1. An example of Audio CAPTCHA of MSN/Hotmail.

1.1. Related Work

These days, CAPTCHAs are designed in the toughest possible way that prevents any algorithm to break them but still significant work has been done to break these CAPTCHAs and can be find out at http://www.captcha.net [2]. In 2003, Mori and Malik [3] proposed a shape matching algorithm to break EZ-Gimpy and Gimpy CAPTCHAs. They achieved a success rate of 92% in case of EZ-Gimpy and 33% in case of Gimpy. In 2004, Moy et. al. [4] used distortion estimation technique to break EZ-Gimpy CAPTCHAs and achieved 99% success rate.

In this paper only EZ-Gimpy CAPTCHAs has been focused. An example of EZ-Gimpy CAPTCHA has been shown in the figures below:

![EZ-Gimpy CAPTCHA](image)

Figure 2. An example of EZ-Gimpy CAPTCHA.

This paper follows an unprecedented approach which first preprocesses the given EZ-Gimpy CAPTCHA, segments the characters, matches the features of character with features in a characteristic table and then uses HMM to recognize the characters of the given CAPTCHA. This approach works on the same 191 EZ-Gimpy CAPTCHA data set which were used by Mori et. al. [3] and is provided at [5].

The paper is divided into various sections. Section 2 discusses about the EZ-Gimpy CAPTCHAs. Section 3 gives the methodology that is adopted for different types of CAPTCHAs. In section 4 results have been discussed.

2. **EZ-GIMPY CAPTCHAS**

EZ-Gimpy CAPTCHAs were used by Yahoo before September 2004. In EZ-Gimpy CAPTCHAs, distant characters (24 letters excepting ‘q’ and ‘z’) which form a dictionary word are written on a complex background. The characters are black in color whereas background is colored. Characters are vertical oriented. EZ-Gimpy CAPTCHAs can be divided into four categories namely (i) Simple (No mesh) Background, (ii) Black Mesh Background, (iii) White Mesh Background and (iv) Loosely Connected Characters.

In Simple (No Mesh) EZ-Gimpy, the characters are written on a simple background whereas in Black and White Mesh EZ-Gimpy the characters are written on black and white mesh respectively. In Loosely Connected Characters type, the pixels of the characters are loosely connected.
3. OUR APPROACH

The flow process chart of our approach is shown in the figure below:

![Flow Process Chart](image)

**3.1. Preprocessing**

Preprocessing Stage is the most important stage of all stages as it prepares the given CAPTCHA for further operations like character recognition etc. Since there are four types of EZ-Gimpy CAPTCHA, each one has different preprocessing operations which are as follows:

### 3.1.1. Simple (No Mesh) Background

(a) The given CAPTCHA is first binarized.

(b) Binarization of an image can lead to noise in image which is removed using a simple 3x3 Median Filter.

(c) The image is then dilated.

(d) Now a filter (Filter A) is applied which sums the adjacent pixels of a test pixel. If sum is less than 5 the test pixels are removed and otherwise it is kept as it is. The algorithm for Filter A is given below:

**Algorithm 1. Filter A**

**Input:** Image with pixels whose sum of adjacent pixels is less than 5.

**Output:** Image without pixels whose sum of adjacent pixels is less than 5.

```
for a1 in 2 to m-1 do
  for a2 in 2 to n-1 do
    if f(a1,a2) is equal to 0 then
      k ← f(a1-1:a1+1,a2-1:a2+1)
      sumxy ← sum(sum(~k)) //sum of adjacent pixels of test pixels
      end
    if sumxy is less than or equal to 5 then
      g(a1,a2) ← 1 //removing the pixel
      sumxy ← 0
    end
  end
k ← zeros(3,3)
end
```

(e) We find the connected components of a binary image and label them. All the pixels in connected components are given a level. The searching of the connected components is done in the column-wise fashion, in other words, in top-to-bottom scan order. Then a filter (Filter B) with threshold of 80 is applied which removes all connected component pixels labeled less than 80. The algorithm for Filter B is given below:

**Algorithm 2. Filter B**

**Input:** Image with connected component pixels labeled less than the threshold.

**Output:** Image without connected component pixels labeled less than the threshold.

```
for a4 in 1 to temp do
  k ← find(ma==a4)
  if size(k,1) is greater than threshold then //checking for connected component pixels labeled greater than threshold.
    end
  end
for a4 in1 to size(ma,1) do
  for a2 in 1 to size(ma,2) do
    if ma(a1,a2) is equal to a4 then
      ma(a1,a2) ← temp+1
    end
  end
end
for a1 in 1 to size(ma,1) do
  for a2 in 1 to size(ma,2) do
    if ma(a1,a2) is not equal to temp+1 then
      ma(a1,a2) ← 1
    else
```
ma(a_1, a_2) \leftarrow 0
end
end
end

(f) Logical AND operation of the resultant and the original image is taken.
(g) Again, Filter B with threshold of 60 and logical AND operation are applied in sequence.

The sequences of images obtained at each stage are shown below:

![Image](image1)

Figure 5. (i) Original image. (ii) Binarized image. (iii) Noise Removed and Dilated image. (iv) Resultant Image after applying Filter A. (v) Resultant Image after applying Filter B. (vi) Resultant image after logical AND operation with original image. (vii) Final image (after applying Filter A, Filter B and logical AND operation).

### 3.1.2. Black Mesh Background
Preprocessing stage for Black Mesh Background CAPTCHAs is same as that of Simple (No Mesh) Background but Filter A and Filter B with threshold of 60 are once again applied in sequence.

The sequences of images obtained at each stage are shown below:

![Image](image2)

Figure 6. (i) Original image. (ii) Binarized image. (iii) Noise Removed and Dilated image. (iv) Resultant Image after applying Filter A. (v) Resultant Image after applying Filter B. (vi) Resultant image after logical AND operation with original image. (vii) Resultant Image after applying Filter A again. (viii) Final image (Resultant Image after applying Filter B again).

### 3.1.3. White Mesh Background
(a) The given CAPTCHA is first binarized.
(b) Binarization of an image can lead to noise in image which is removed using a simple 3x3 Median Filter.
(c) The image is now inverted. Now the same black mesh background as stated above is applied.
(d) The image is now inverted to get back the original image.

The sequences of images obtained at each stage are shown below:

![Image](image3)

Figure 7. (i) Original image. (ii) Binarized and Noise Removed image. (iii) Resultant Image after inverting the image (iv) Resultant image after applying black mesh background algorithm. (v) Final image after inverting again.
3.1.4. Loosely Connected Characters
(a) The given CAPTCHA is first binarized.
(b) Binarization of an image can lead to noise in image which is removed using a simple 3x3 Median Filter.
(c) The image is then dilated to get the final image.

The sequences of images obtained at each stage are shown below:

Figure 8. (i) Original image. (ii) Binarized and Noise Removed image. (iii) Resultant Image after dilation.

3.2. Character Segmentation

After the image passes the preprocessing stage, characters need to be segmented because if characters are joined then it is very difficult to recognize them. In EZ-Gimpy CAPTCHA all the characters are distant so it is not very difficult to segment them and so are segmented by checking the connected component pixels of each character. An example of segmentation of characters is shown in figure below.

Figure 9. (i) Preprocessed image. (ii) Characters being segmented into c, o, l, l, a, r.

3.3. Feature Extraction

Once the characters are segmented, unique features like number of holes, the height of character etc. of each character are extracted so that they can be matched with the features of a particular character in the characteristics table shown in Table 1.

3.4. Feature Matching

Once the features of a particular character are obtained they are matched with the features of the characters listed in the characteristics table (Table 1). The best matched characters are taken from the table. The table lists the features of 26 characters (‘a’ to ‘z’). The features which were used are as follows:
(i) Number of Holes: - Each character is checked whether it has a hole or not. Characters a, b, d, e, g, o, p, q each have 1 hole and rest of the characters do not have a hole.
(ii) Height of Character: - Each character is categorized into small or big on the basis of its height. A threshold is taken which categorizes the given character.
(iii) Maximum Number of White-Black Transitions: - A line cutting the character is drawn as shown in table and the maximum numbers of white-black transitions that are possible are noted. Example: - When a line is drawn through character ‘a’, maximum 3 transitions are possible. Similarly transitions for other characters were also noted.
(iv) Light Fall: - The light is made to fall from both sides i.e. upwards and downwards on a character and pixel area of character is noted. Two flags (Flag1 and Flag2) are used. In case, the final pixel area (pix.f) of a character is greater than the initial pixel area (pix.i) of a character flag1 is set to 1, otherwise to 0. And in case, the final pixel area (pix.f) of a character is greater than the initial pixel area of character’s hole (pix.i-holearea) flag2 is set to 1, otherwise to 0. The different characters are shown in the table and green portion is the pixel area when light is made to fall on it.
(v) Nature of Vertical Stroke: - A vertical stroke (Blue in color) is drawn for character with ‘big’ height except ‘y’ (no vertical stroke) along the vertical stroke of the character. If the character is of ‘small’ height ‘-‘ is noted. If there is no hole in character ‘1’ is noted. And if there is a hole in character then the position of vertical stroke relative to the hole i.e. either left or right is noted.

The features used for each character are listed in the table shown below:

<table>
<thead>
<tr>
<th>Character</th>
<th>No. of Holes</th>
<th>Character Height</th>
<th>Max. no. of White-Black Transitions</th>
<th>Light Fall Flag1 (pix.f&gt;pix.i), Flag2 (pix.f&gt;pix.i-holearea).</th>
<th>Nature of Vertical stroke (position wrt hole if any).</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>Small</td>
<td>3</td>
<td>1,1</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>Big</td>
<td>3</td>
<td>1,0</td>
<td>1 to the left</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>Small</td>
<td>2</td>
<td>1,1</td>
<td>-</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>Big</td>
<td>3</td>
<td>1,0</td>
<td>1 to the right</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>Small</td>
<td>3</td>
<td>1,1</td>
<td>-</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>Big</td>
<td>3</td>
<td>1,1</td>
<td>1</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>Big</td>
<td>3</td>
<td>1,1</td>
<td>1 to the right</td>
</tr>
</tbody>
</table>
Table 1. Characteristics Table of 26 Characters.

<table>
<thead>
<tr>
<th>Character</th>
<th>Size</th>
<th>Feature 1</th>
<th>Feature 2</th>
<th>Feature 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Big</td>
<td>3</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>Small</td>
<td>3</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>j</td>
<td>Big</td>
<td>2</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>Big</td>
<td>2</td>
<td>1.1</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>Big</td>
<td>3</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>m</td>
<td>Small</td>
<td>3</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>n</td>
<td>Small</td>
<td>2</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>o</td>
<td>Small</td>
<td>2</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td>p</td>
<td>Big</td>
<td>3</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td>q</td>
<td>Big</td>
<td>2</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td>r</td>
<td>Small</td>
<td>2</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>s</td>
<td>Small</td>
<td>3</td>
<td>1.1</td>
<td>-</td>
</tr>
<tr>
<td>t</td>
<td>Big</td>
<td>2</td>
<td>1.1</td>
<td>1</td>
</tr>
<tr>
<td>u</td>
<td>Small</td>
<td>2</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>v</td>
<td>Small</td>
<td>2</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>w</td>
<td>Small</td>
<td>4</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>x</td>
<td>Small</td>
<td>2</td>
<td>1.1</td>
<td>-</td>
</tr>
<tr>
<td>y</td>
<td>Big</td>
<td>2</td>
<td>1.1</td>
<td>-</td>
</tr>
<tr>
<td>z</td>
<td>Small</td>
<td>3</td>
<td>1.1</td>
<td>-</td>
</tr>
</tbody>
</table>

3.5. HMM Training

After the extraction of the above listed 5 features the hmm model is trained with a set of tentative solutions. Three samples for each of the character are taken and the hmm model is trained with the chain code of their respective contours using the algorithm below.

**Algorithm 3. HMM**

**Input:** Sequence of Characters.

**Output:** Probabilities of Characters.

```plaintext
datach1{1} Å [sample.ch1_1]
datach1{2} Å [sample.ch1_2]
datach1{3} Å [sample.ch1_3]
datach2{1} Å [sample.ch2_1]
datach2{2} Å [sample.ch2_2]
datach2{3} Å [sample.ch2_3]

hmmch1.prior Å [1 0 0]
hmmch1.transmat Å rand(3,3) % 3 by 3 transition matrix
hmmch1.transmat(2,1) Å 0
hmmch1.transmat(3,1) Å 0
hmmch1.transmat(3,2) Å 0
hmmch1.transmat Å mk_stochastic(hmmch1.transmat)
hmmch1.obsmat Å rand(3, 16) % # of states * # of observation
hmmch1.obsmat Å mk_stochastic(hmmch1.obsmat)

hmmch2.prior = [1 0]
hmmch2.transmat Å rand(2,2) % 2 by 2 transition matrix
hmmch2.transmat(2,1) Å 0
hmmch2.transmat Å mk_stochastic(hmmch2.transmat)
hmmch2.obsmat Å rand(2, 16) % # of states * # of observation
hmmch2.obsmat Å mk_stochastic(hmmch2.obsmat)

[hmmch1.transmat, hmmch1.obsmat] Å hmmtrain(data0, hmmch1.transmat, hmmch1.obsmat)
[hmmch2.transmat, hmmch2.obsmat] Å hmmtrain(data1, hmmch2.transmat, hmmch2.obsmat)

for dt in 1 to length(data1)
    loglikech1 Å dhmm_logprob(data1{dt}, hmmch1.prior, hmmch1.transmat, hmmch1.obsmat)
    loglikech2 Å dhmm_logprob(data1{dt}, hmmch2.prior, hmmch2.transmat, hmmch2.obsmat)
end

for dt in 1 to length(data2)
    loglikech1 Å dhmm_logprob(data2{dt}, hmmch1.prior, hmmch1.transmat, hmmch1.obsmat)
    loglikech2 Å dhmm_logprob(data2{dt}, hmmch2.prior, hmmch2.transmat, hmmch2.obsmat)
end

temp Å [test_character]
    loglikech1 Å dhmm_logprob(temp, hmmch1.prior, hmmch1.transmat, hmmch1.obsmat)
    loglikech2 Å dhmm_logprob(temp, hmmch2.prior, hmmch2.transmat, hmmch2.obsmat)
```
Once the trained model is obtained the matched probability of the test sample is calculated. The higher is the probability the better is the match.

4. RESULTS

191 samples were taken, out of which 187 samples were being identified. The accuracy is much dependent on the preprocessing done on the samples particularly on the dilation part. The accuracy in case of Loosely Connected Characters EZ-Gimpy type CAPTCHA is low. Improving the same can lead to betterment of the results.

5. REFERENCES