ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

November 4, 2014

Contents

1 Overview

In this assignment we have attempted to recognize different English characters based on shape. All English characters are all originally included in training/test images. Thus, before we attempt to recognize different letters, we will first separate letters into different sub-images. **Each sub-image will only contain 1 character for letter recognition**. In order to do that, we will first apply Otsu's algorithm on the gray-scale images to find the threshold value T_k. Then, based on threshold value the image segmentation will be performed. **Multiple iterations of image segmentation based on Otsu's threshold might be performed if segmentation result could be optimized.**

Note that the image segmentation result based on Otsu's threshold is still the size of original image. We only separate the foreground (characters) from the background at this moment. Hence in the same images there are many characters and they still can not be separated into different sub-images.

Component labelling is the process that we can separate the pixel blob containing each letters. The component labelling process can be done by applying the **8 component neighbourhood segmentation** to the segmented images (binary image) obtained in previous step. As long as the segmented images based on Otsu's algorithm is great, component labelling would not be troublesome. However, if the binary image obtained from previous step is noisy, then component labelling would give awful result.**Example of accurate component labelling please refer to result section**.

Finally, after both the image segmentation and component labelling processes have been done, the pixel-blobs (that only contain a single character) extracted from both the training and testing image are stored separately. To classify those separate characters, we need to generate a feature vector.

Feature vectors will be based on N Harris corners detected. (Sample Harris Corners will be presented in the result session). Now, assume the center of the pixel-blob is the center of the sub-image. (**Obviously this is not complete accurate as the sub-image is a square shape well the pixel-blob is a circle. The inaccuracy in this step probably will impact the final recognition accuracy**) After center is picked from the pixel blob, we will radially project each of the N Harris corners to the unit-circle. Then, the arc lengths on the unit circle between consecutive projections of Harris corners will be calculated and the feature vector then will be a length N vector, denote as $\vec{v}_{feature} \in \mathbb{R}^{1\times N}$.

$$\sum_{i=1}^{N} \vec{v}_{feature}(i) = 2\pi$$

Character classification will then be based on minimum euclidean distance between the i^{th}

testing pixel-blob and the j^{th} training pixel-blob.

$$\hat{j} = \arg\min_{j} \sum_{i=1}^{N} \{\vec{v}_{feature}^{(test:\ i)}(i) - \vec{v}_{feature}^{(training:\ j)}(i)\}^2$$

Note that although theoretically the sharpest corners found by Harris Detector should be the same if the character are the same, this is typically **NOT** the case in real applications. As different font will return very different combinations of sharpest Harris corners, the recognition accuracy will decrease significantly if this issue could not be fixed.

`The literature of Harris Corner Detection and Otsu's method will be quoted from my previous homework.`

To conclude, the whole process is as followed:

1. Convert color image to gray-scale image.
2. Perform Otsu's algorithm to find the threshold value for image segmentation. Multiple iterations is required if result could be optimized.
3. Extract single characters from the entire image and save each of those extracted characters in a separate pixel-blob.
4. Formulate the feature vectors of both training and testing data based on arc length of consecutive interest points on the unit circle after we project the Harris corners radially to unit circle.
5. Match the English characters based on the training data based on Euclidean Distance between each element of feature vectors.

2 Otsu's Algorithm

`This part is based on my own homework 6`

Image segmentation based on Otsu's threshold would be performed on the converted **gray-scale** image. Thus, first of all, all images need to be converted to **single channel gray-scale** images.

Otsu's algorithms is designed so that the optimal threshold will be picked to distinguish foreground (characters in this assignment) from background.

Assume there are total of L different gray levels in a gray-scale image and the total number of pixel is N, and there are n_i pixels associated with each of the gray level. Then, easily we can calculate the probability p_i that a random sampled pixel is from i^{th} gray level. Denote as:

$$p_i = \frac{n_i}{N}$$

$$where\ p_i \geq 0,\ \sum_{i=1}^{L} p_i = 1\ and\ \sum_{i=1}^{L} n_i = N$$

Assume there are two classes $\mathcal{C}_0$ and $\mathcal{C}_1$ which denote background and foreground. Assume the threshold value in gray level to distinguish foreground and background is k. Thus, we have zero-order moment:

$$w_0 = Pr(\mathcal{C}_0) = \sum_{i=1}^{k} p_i = w(k)$$

$$w_0 = Pr(\mathcal{C}_1) = \sum_{i=k+1}^{L} p_i = 1 - w(k)$$

Accordingly, the conditional mean (first order moment) is:

$$\mu_0 = \sum_{i=1}^{k} i \times Pr(i|\mathcal{C}_0) = \sum_{i=1}^{k} \frac{ip_i}{w_0} = \frac{\mu(k)}{w(k)}$$

$$\mu_0 = \sum_{i=k+1}^{L} i \times Pr(i|\mathcal{C}_0) = \sum_{i=k+1}^{N} \frac{ip_i}{w_1} = \frac{\mu_T - \mu(k)}{1 - w(k)}$$

$$Where:\ w(k) = \sum_{i=1}^{k} p_i,\ \mu(k) = \sum_{i=1}^{k} ip_i,\ \mu_T = \mu(L) = \sum_{i=1}^{L} ip_i$$

Our task is to find the value k that maximize the **within-class variance**:

$$\hat{k} = \arg\max_k \sigma_B^2, \quad where\ \sigma_B^2 = w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2$$

Thus:

$$\hat{k} = \arg\max_k \{w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2\}$$

Simplify furthermore:

$$\hat{k} = \arg\max_k \{w_0 w_1(\mu_1 - \mu_0)^2\}$$

Note that in this implementation, σ_B^2 is required for each value k.

Now, after the threshold value has been obtained the image segmentation could be easily performed on the **gray-scale** image.

3 Component Labelling

For the component labelling, we will implement the **8 neighbourhood condition**. From the output of the previous image segmentation step, we know that at the pixel location of foreground (characters) we will have value $'1'$ and at the location of the background we will have value $'0'$. This is a binary image.

For example, we will start scanning the image following ratser order. After first $'1'$ pixel (i, j) was detected, we then check its 8 neighbourhood (the set $\partial(i, j)$ denote the 8 neighbourhood of pixel (i, j)):

$$\partial(i,j) = \{(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)\}$$

Now, check all the 8 neighbourhood pixel points to see if any of them are $'1'$. If for example (i', j') is, then put (i', j') into the set $\mathbb{C}_{connected}$ together with (i, j). We can consider the set $\mathbb{C}_{connected}$ is a stack or a vector in computer algorithms, we will check from the 1st element all the way till the very last element.

Now, after we check the very first pixel that belong to $\mathbb{C}_{connected}$, we might have dig up more pixels that are connected to it in the neighbourhood. Now our $\mathbb{C}_{connected}$ begin to rapidly expand. For example 4 of the neighbours are connected to the first element then the size of

$$size\ of\ \{\mathbb{C}_{connected}\}\ after\ first\ iteration\ = 5$$

Now, **iteratively we need to perform the same connect component detection on the next element of the set $\mathbb{C}_{connected}$.** Please note that in order to avoid the program fall into a infinity loop, **the pixels that have been checked will no longer be considered again.** This means, if a pixel is within a set $\mathbb{C}_{connected}$, it will appear once and only once. (Otherwise program will go to infinite looping)

We can observe the set $\mathbb{C}_{connected}$ increases rapidly at the initial stage (a lot of new points added at the beginning) and will converge after all the points in the set have been checked. **After the complete $\mathbb{C}_{connected}$ has been obtained, assign a class label to all the pixel in the set and then mask this entire set off (pixels included in $\mathbb{C}_{connected}$ will no longer be interest points for next iteration)**.

We can only detect one $\mathbb{C}_{connected}$ at one time, thus after entire set $\mathbb{C}_{connected}$ is obtained we will extract the set (and that will be a single character extracted).

Since this is an iterative (to some level it is recursive) approach, it would take some time to compute for the entire image. For an efficient implementation in C we could refer to `Prof. Bouman's ECE637 Laboratory: Neighborhoods and Connected Components.`

Please refer to result section for component labelling result.

4 Harris Corner Detection

`This part is based on my own homework 4`

Harris Corner Detector was used to find those corners of each letters and the corners detected will be as the interest points. This time, both `Sobel Operator` and `Harr Filter` will be implemented. The corners turned out to be accurate for both of the approach. (Please refer to results section for examples)

1. First of all, the gradient along x and y directions in the image need to be calculated. If scalable is not the goal, we can use `Sobel Operator`, which is defined as below:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

   However, if instead the scalability is the concern we will use `Haar Filter`. Haar Filter was implemented to replace Sobel Operator by finding the d_x and d_y. Below is an example of Haar filter with $\sigma = 1.2$.

$$Haar\ Filter\ for\ \frac{\partial}{\partial x},\ with\ \sigma = 1.2: \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

$$Haar\ Filter\ for\ \frac{\partial}{\partial y},\ with\ \sigma = 1.2: \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

   Please note the above form of Haar filter is based on the expansion of Haar wavelet at basic form. We have to make sure that the forms are scaled up to an M by M operator where M is the smallest even integer greater than $4 \times \sigma$. (Similarly we can easily prove that while $\sigma = 1.2$, M = 6. And when $\sigma = 1.4$, M = 8)

   As our segmented image from previous step is good and almost does not have noise, we will not implement the Gaussian smoothing filter.

2. After the d_x and d_y was obtained, we then create a neighbourhood window of size $5\sigma \times 5\sigma$. Note that the σ should be consistent of that used in the first part when we are filtering the image using Haar filter. The C matrix could then be constructed:

$$C = \begin{bmatrix} \sum d_x^2 & \sum d_x d_y \\ \sum d_x d_y & \sum d_y^2 \end{bmatrix}$$

3. While C in the previous step is a 2×2 matrix, we will first check the rank of $C_{i,j}$ at pixel location (i,j). As long as $rank(C) \neq 2$, we will remove the pixel locations from our candidates list of corners/interest points. **The computation efficiency could be improved significantly if we can first eliminate majority of candidates points.**

4. For the remaining corner candidates, we than need to determine the corner strength. Define

$$Conrner\ Response = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

While k is defined as a constant: 0.04, λ_1 and λ_2 is the eigenvalues of matrix C. Obviously if C is not rank 2 matrix the candidate point would not worth investigating. In order to simplify the calculation,

$$\det(C) = \lambda_1\lambda_2$$

$$trace(C) = \lambda_1 + \lambda_2$$

therefore, SVD of matrix C would then not be required.

5. After corner response at each candidates pixel has been calculated, we then set up a threshold to filter out those points whose corner responses are not strong enough. **Thus, we only pick up the N strongest corner response as our interest points. It is necessary to have local non-maxima suppression algorithm implemented because if all the interest points go to the same vertex then the interest points would be somehow useless.**

6. Now all the Harris corner detection technique has been performed and we have exact N interest points. Save the location of the interest points extracted from each images separately for feature vector formulation.

5 Feature Vector Formulation and Character Classification

Feature vector is constructed based on radially projecting the Harris Corners detected in previous step onto the unit circle. Then, find the arc lengths between the consecutive projections on the unit circle and we will form a feature vector of length N.

Now, as the arc length on the unit circle between two consecutive projections is exact the same as the scale of angle between two consecutive projection **radial line**, we need to detect the angle of each of the N projection radial lines with respect to the positive $x - axis$.

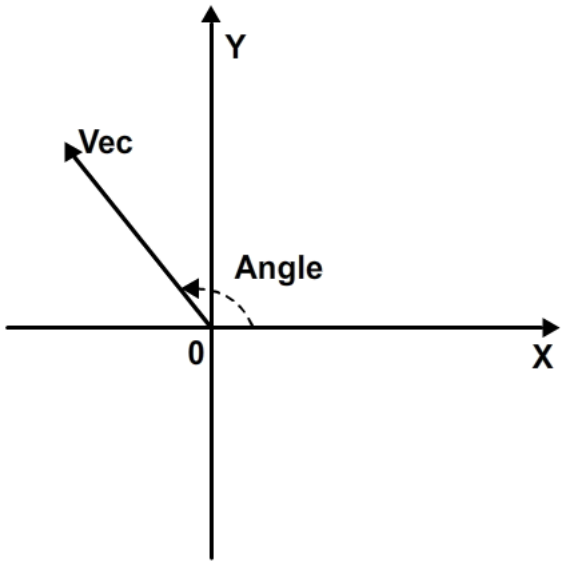


Figure 1. The angle of each of the N projection radial lines with respect to the positive $x - axis$.

We can simply find the angle by finding the

$$\theta = \arctan \frac{y - y_{center}}{x - x_{center}}$$

Now, clearly we have a issue here as arctan function can only return values in $[-\frac{\pi}{2}, \frac{\pi}{2}]$, as below:

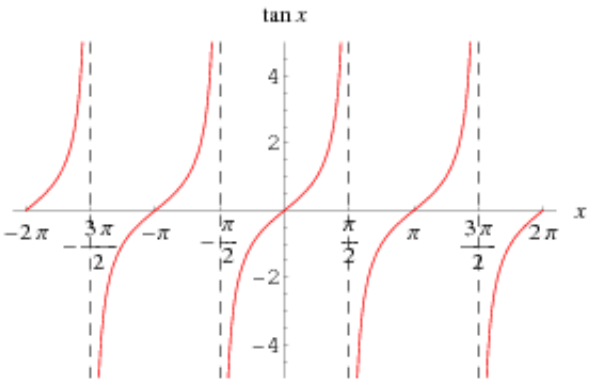


Figure 2. The Tangent function

What about those angles in **second quadrant** and **third quadrant**?

Simple. First we check if $x - x_{center} \geq 0$.

1. If $x - x_{center} \geq 0$, $\theta = \arctan \frac{y - y_{center}}{x - x_{center}}$
2. Else, $\theta = \arctan \frac{y - y_{center}}{x - x_{center}} + \pi$

Problem Solved!

Furthermore, if we want convert all angles θ into the region $[0, 2\pi]$, simply check if $\theta < 0$. For those $\theta < 0$, add 2π to the negative angle.

After convert all angles to the range $[0, 2\pi]$ range we simply sort all N angles in a ascending order and find the difference (actual arc length on unit circle) between two consecutive angles. This would go into a circular way thus a total of N arc lengths will be obtained. Certainly, sum of all arc lengths will be 2π. Then, classification will be based on Euclidean distances. Following is the Mathmatics expression.

Then, the arc lengths on the unit circle between consecutive projections of Harris corners will be calculated and the feature vector then will be a length N vector, denote as $\vec{v}_{feature} \in \mathbb{R}^{1 \times N}$.

$$\sum_{i=1}^{N} \vec{v}_{feature}(i) = 2\pi$$

Character classification will then be based on minimum euclidean distance between the i^{th} testing pixel-blob and the j^{th} training pixel-blob.

$$\hat{j} = \arg\min_{j} \sum_{i=1}^{N} \{\vec{v}_{feature}^{(test:\ i)}(i) - \vec{v}_{feature}^{(training:\ j)}(i)\}^2$$

6 Result

6.1 Result: Overall Conclusion

The results were obtained using different N. From the following section, we can observe the recognition accuracy does not necessarily increase nor decrease with the number N increasing. We have performed $N = 4, 5, 6, 7$ on images: `Image1.jpg, Image2.jpg, Image3.jpg, Image4.jpg, Image5.jpg`.

We only test `Imag6.jpg` with $N = 5$ because the segmentation in image6 is much worse than good enough. If we can not even segment the image correctly, there is nothing more we can do to recognize the individual letters.

As the accuracy would actually increase with N for test image `Image3.jpg, Image4.jpg, Imag5.jpg`, we will actually try to reach a higher $N = 8, 9$ to see if the trend would continue.

Now, before we proceed to test our algorithm on testing data, we first test it on training data. Obviously, if the algorithm is working correctly the accuracy should be 100% and indeed the **accuracy is** 100% **on training data**.

However, this is still not strong enough to claim the program is working as expected, thus we generated a new testing image: `Image0.jpg`, which is just 180 degree rotation of the training data. The results are recoded as followed:

1. Letters Recognized for N = 4: A,B,D,C,I,K,J,O,S,U,Q,Y,X (total: 13)
2. Letters Recognized for N = 5: A,D,G,I,K,N,S,U,Q,W,Y,X (total: 12)
3. Letters Recognized for N = 6: B,C,G,I,K,L,J,P,U,Q,W,Y,Z,X (total: 14)
4. Letters Recognized for N = 7: B,E,C,G,I,K,N,J,P,R,Q,V,W,Y,Z,X (total: 16)
5. Letters Recognized for N = 8: G,I,K,L,N,J,P,R,T,Q,V,W,Z,X (total: 14)
6. Letters Recognized for N = 9: G,I,K,N,J,R,T,Q,V,W,Y,Z,X (total: 13)

The accuracy is not 100% mainly because when the letter rotate, although we can segment the pixel blocks accurately the machine (**even the human**) can not locate the center of each letter correctly all the time. If the center of the letters are not estimated all the same all the time, obviously we will have some issue with arc-length which is our feature vector.

Even the result is not perfect, it can still conclude that given a not-so-accurate center the recognition result is still very good.

The reason that lower/higher value N does not always guarantee better result is very likely due to the different shapes of each characters. For example, letter **X**, we can easily detect 12 corner out of it while for letter **O**, there is no significant corner at all!

If the character itself has many corners and our N is not larger enough, we have insufficient feature hence detection is not accurate. Similarly, If the character itself has very few corners and our N is too larger, we have more than sufficient feature (very likely noise) hence detection is not accurate again.

The below table indicate something similar. As the letters in `Image1.jpg and Image2.jpg` have very different shapes compared to our training data, the more points we obtain, more likely we will recognize the wrong one (shape is different). On the other side, as the letters in `Image3.jpg and Image4.jpg and Image5.jpg` have very similar shapes compared to our training data, the more points we obtain (not too many obviously, according to previous statement), more likely we will recognize the correct one (shape is similar)!

$Test\ Image$	$N=4$	$N=5$	$N=6$	$N=7$
$Image1.jpg$	F, I, Y	I, J	U, I, X, J	X
$Image2.jpg$	X, Y	R, G, C, S	M, J, G	J, G
$Image3.jpg$	Y, Q	Z, X, B, I	Z, X, K, B, I	M, X, K, N, R, I
$Image4.jpg$	Y, L, E, I	Y, L, P, S, Y, I	L, P, Y, E, T, I, H	L, Y, N, T, G, W, I, H
$Image5.jpg$	E, S, K, D, Q	N, E, Y, K	N, L, X, Q, B, P	$N, G, Z, L, Y, K, X, Q, B, V$
$Image6.jpg$	$None$	$None$	$None$	$None$

Table 1. Letters Recognized Correctly For Each Test Images for $N = 4, 5, 6, 7$.

In order to verify our assumption that: 'too many corners would not necessarily help', we have further test $N = 8, N = 9$ on `Image3.jpg and Image4.jpg and Image5.jpg`

$Test\ Image$	$N=8$	$N=9$
$Image3.jpg$	K, R, T	F, X, K, L, J, I, T
$Image4.jpg$	E, Y, L, R, T, G, W, I	R, L, Y, N, G, I, H
$Image5.jpg$	$N, G, Z, T, Y, K, R, X, Q, V$	N, F, Z, Y, K, X, W, V

Table 2. Letters Recognized Correctly For `Image3.jpg and Image4.jpg and Image5.jpg` with $N = 8, 9$.

Best N value for each of the test image character recognition:

1. Image1.jpg: N = 6 (total 4)
2. Image2.jpg: N = 5 (total 4)
3. Image3.jpg: N = 9 (total 7)
4. Image4.jpg: N = 7 or 8 (total 8)

5. Image5.jpg: N = 7 or 8 (total 10)
6. Image6.jpg: Failed due to segmentation failure

Please note that: as some of the initial image quality is not good enough for the segmentation, we will perform necessary image pre-processing/enhancement in order to improve our segmentation result. **If our segmentation is not good enough than the letter extraction would be catastrophic not to mention the character recognition**.

6.2 Performance Measures

Since `Image6.img` did not segmented well, we will exclude `Image6.img` for performance measurement.

Test Image	$N = 4$	$N = 5$	$N = 6$	$N = 7$
Training.jpg	100%	100%	100%	100%
*Image*1.*jpg*	11.5%	7.69%	15.38%	3.84%
*Image*2.*jpg*	7.69%	15.38%	11.53%	7.69%
*Image*3.*jpg*	7.69%	15.38%	19.23%	23.07%
*Image*4.*jpg*	15.38%	23.07%	26.92%	30.76%
*Image*5.*jpg*	19.23%	15.38%	23.07%	38.46%
*Image*6.*jpg*	*None*	*None*	*None*	*None*

Table 3. Letters Recognition Accuracy For Each Test Images for $N = 4, 5, 6, 7$.

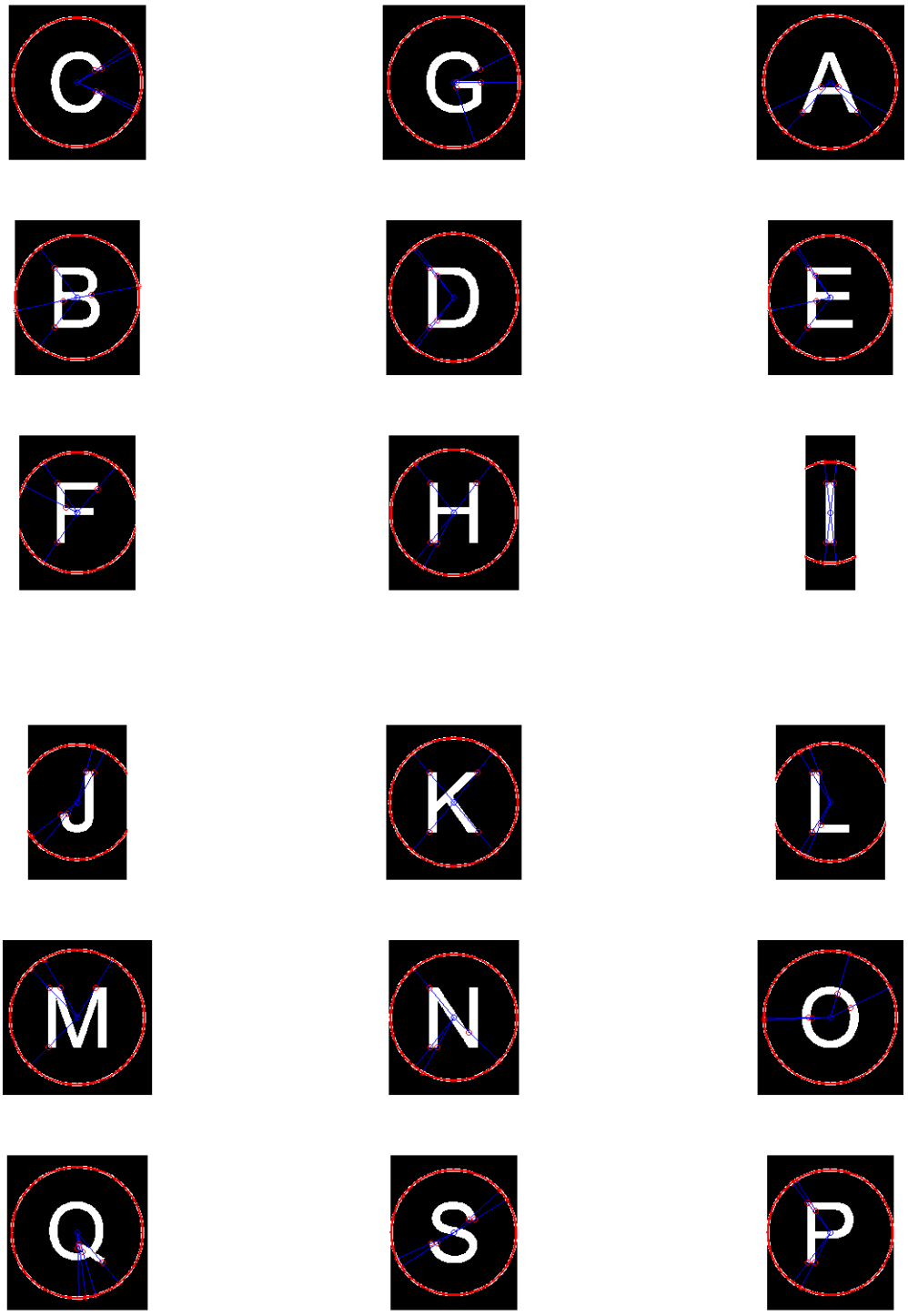
Test Image	$N = 8$	$N = 9$
*Image*3.*jpg*	11.53%	26.92%
*Image*4.*jpg*	30.76%	26.92%
*Image*5.*jpg*	38.46%	30.76%

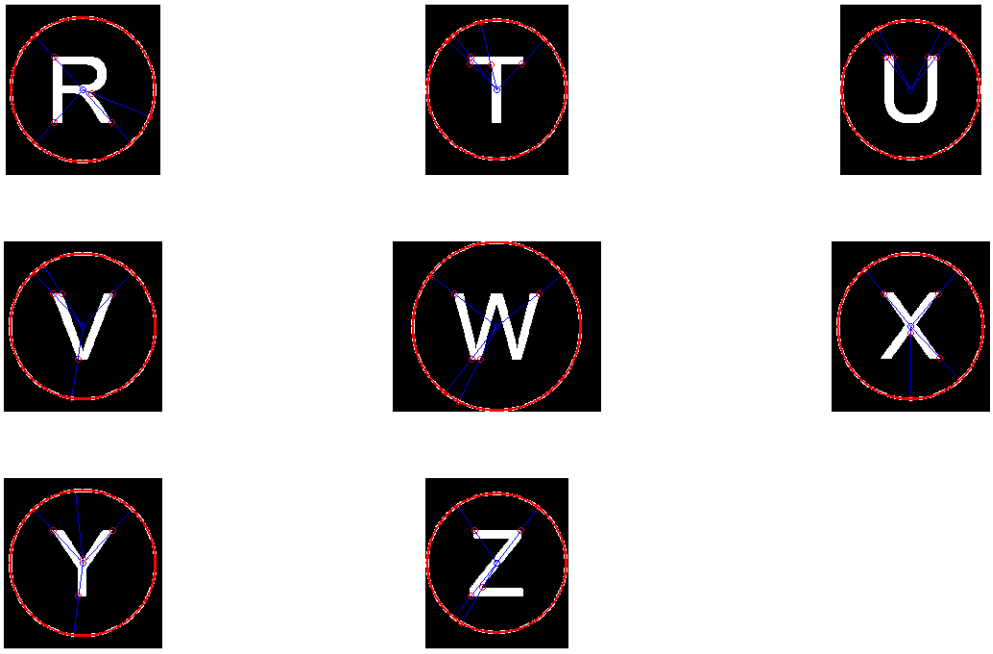
Table 4. Letters Recognition Accuracy For `Image3.jpg and Image4.jpg and Image5.jpg` with $N = 8, 9$.

Letter	$N = 4$	$N = 5$	$N = 6$	$N = 7$
A	0	0	0	0
B	0	0.2	0.4	0.2
C	0	0.2	0	0
D	0.2	0	0	0
E	0.4	0.2	0.2	0
F	0.2	0	0	0
G	0	0.2	0.2	0.6
H	0	0	0.2	0.2
I	0.4	0.6	0.6	0.4
J	0	0.2	0.4	0.2
K	0.2	0.2	0.2	0.4
L	0.2	0.2	0.4	0.4
M	0	0	0.2	0.2
N	0	0.2	0	0.6
O	0	0	0	0
P	0	0.2	0.2	0
Q	0.4	0	0.2	0.2
R	0	0.2	0	0.2
S	0.2	0.4	0	0
T	0	0	0	0.2
U	0	0	0.2	0
V	0	0	0	0.2
W	0	0	0	0.2
X	0.2	0.2	0.6	0.6
Y	0.8	0.4	0	0.4
Z	0	0.2	0.2	0.2
Overall	0.123	0.146	0.161	0.207

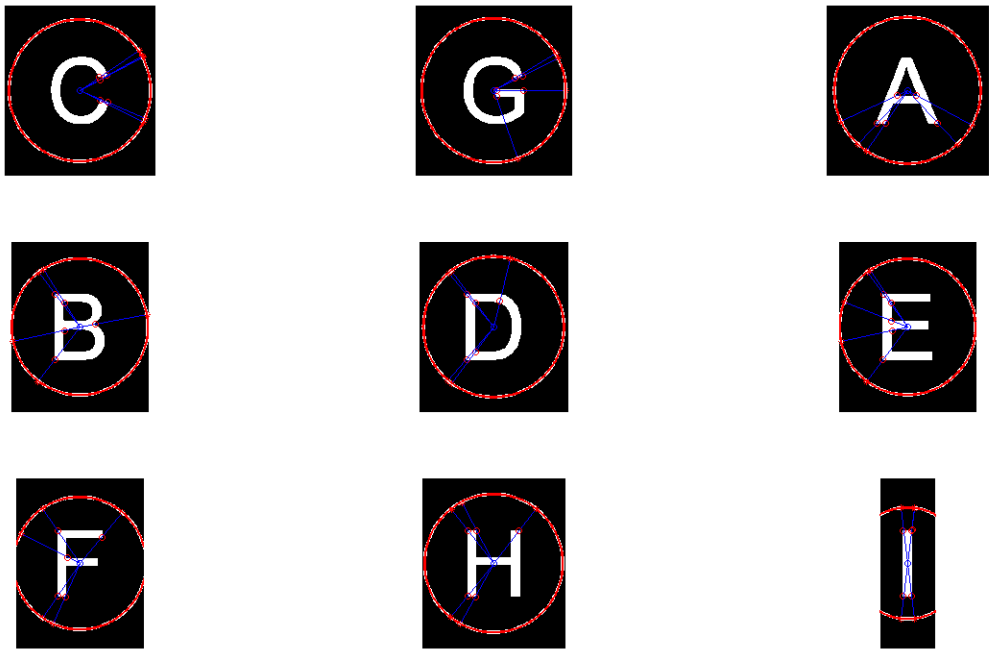
Table 5. Recognition Accuracy For Each Letter and Overall Accuracy for: $N = 4, 5, 6, 7$.

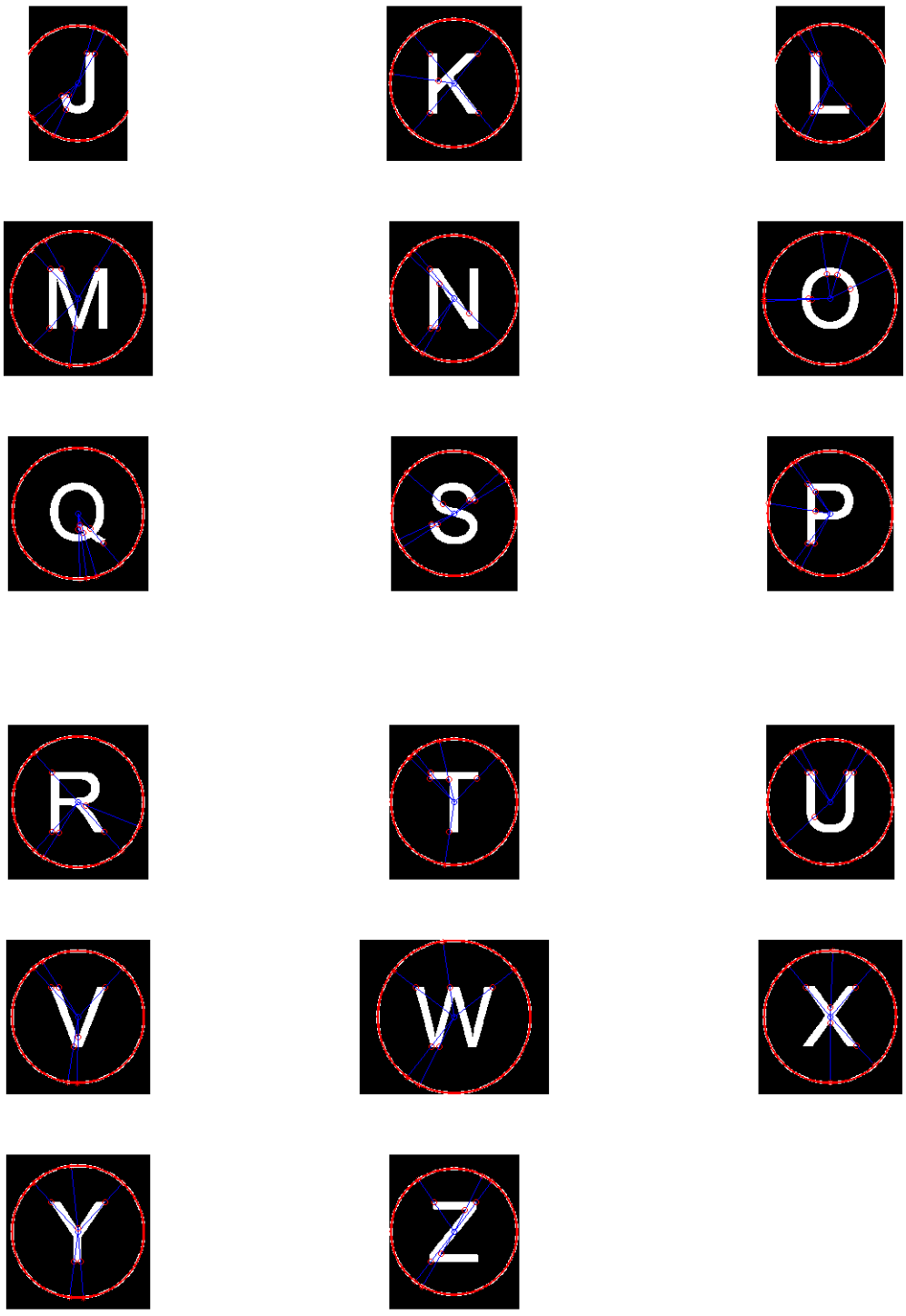
From the following result we can conclude that **Image Segmentation, Component Labelling, Harris Corner Detection and Angle/Arc Length Feature Vectors Formulation are all very accurate.** All images are obtained based on training data.


C
G
A
B
D
E
F
H
I
J
K
L
M
N
O
Q
S
P

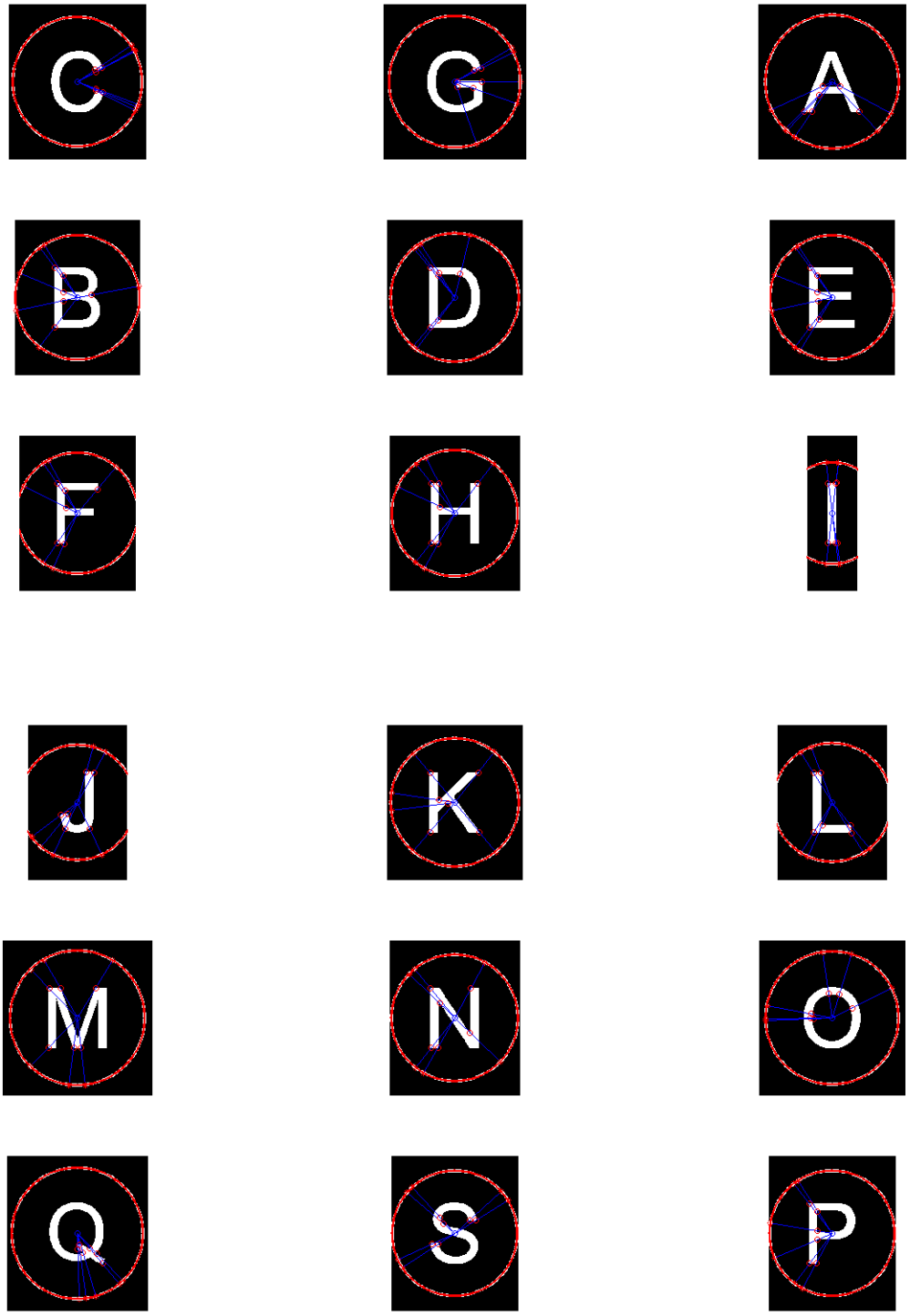


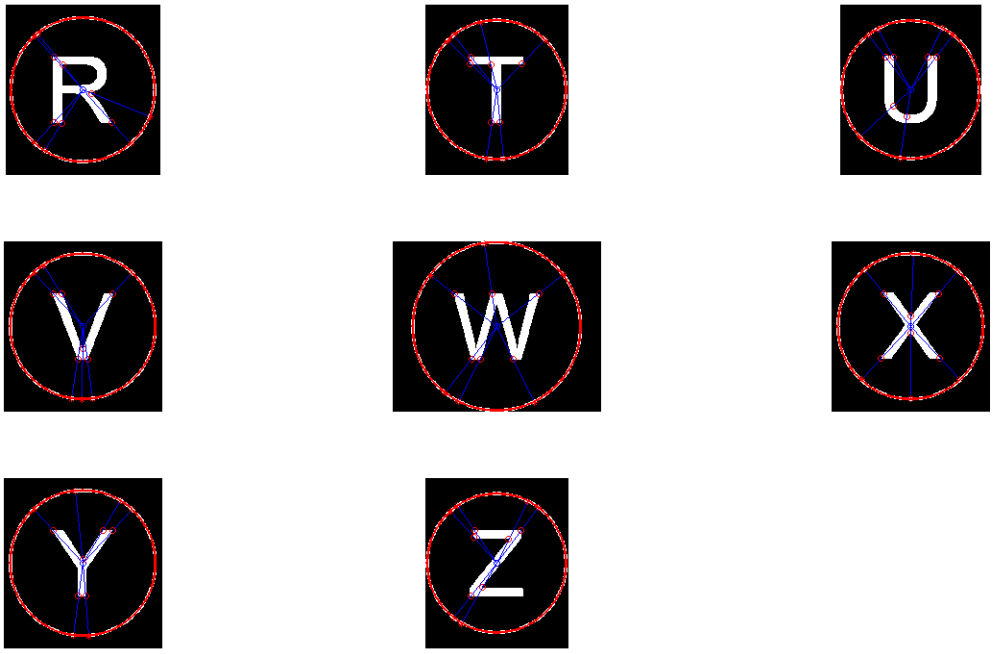
Appendix I. Above Three
Plots Are Projection of Corners To Unit Circle. `Training.jpg`, $N = 4$ (Zoom In For Detail).



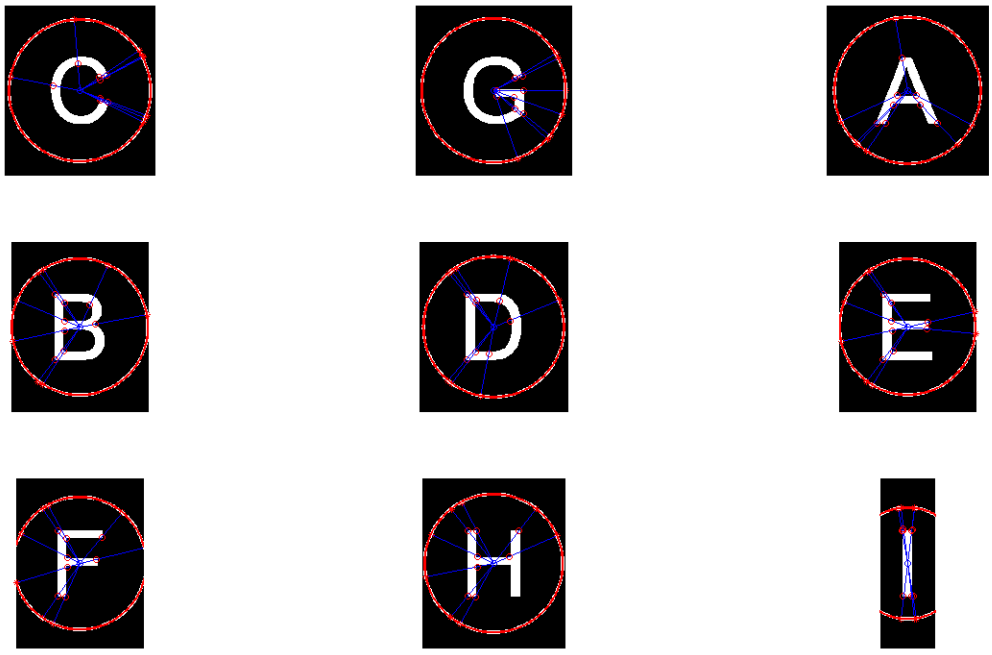


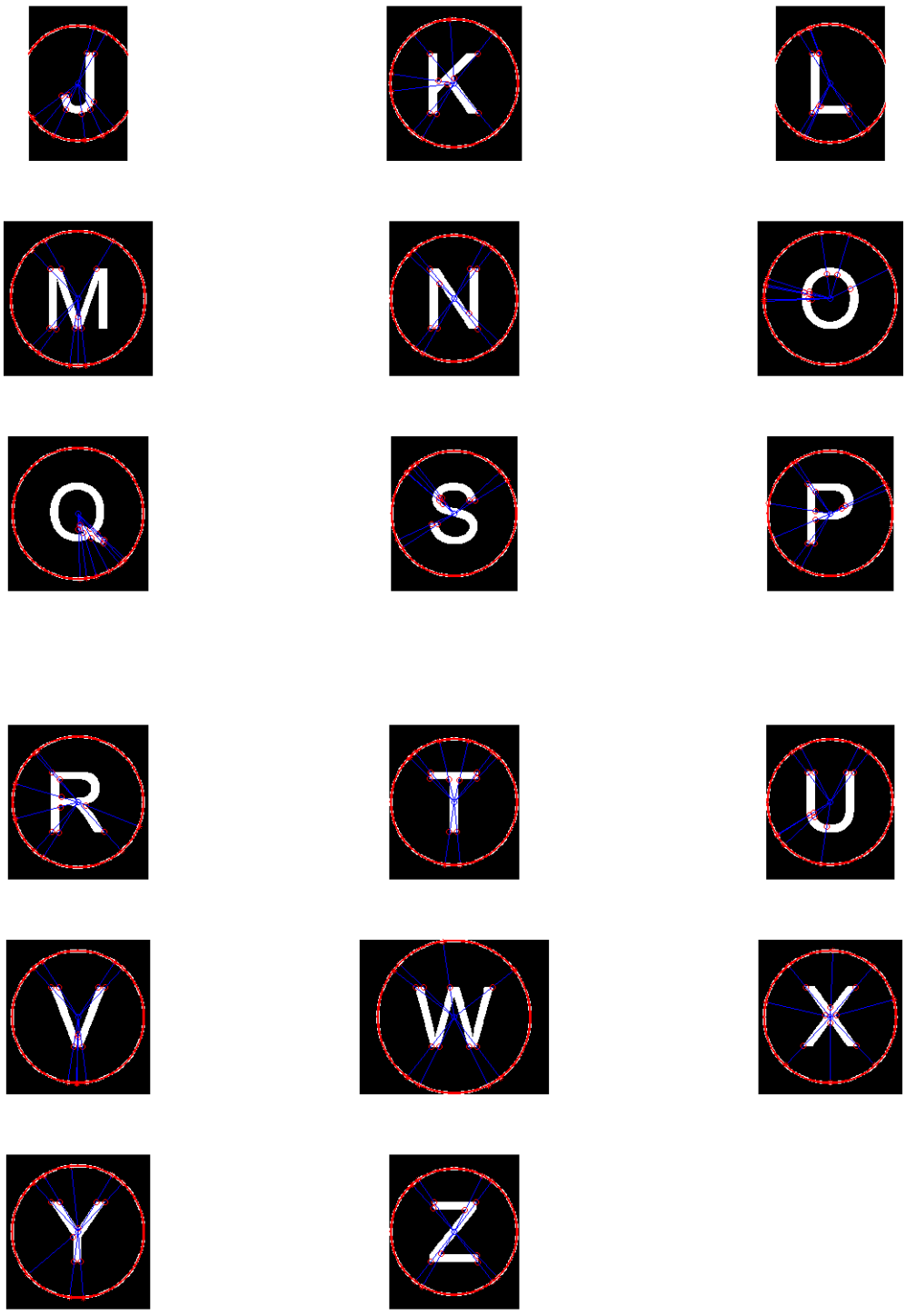
Appendix II. Above Three
Plots Are Projection of Corners To Unit Circle. `Training.jpg`, $N = 5$ (Zoom In For Detail).


C
G
A
B
D
E
F
H
I
J
K
L
M
N
O
Q
S
P



Appendix III. Above Three
Plots Are Projection of Corners To Unit Circle. `Training.jpg`, $N = 6$ (Zoom In For Detail).





Appendix IV. Above Three Plots Are Projection of Corners To Unit Circle. `Training.jpg`, $N = 8$ (Zoom In For Detail).

6.3 Training Image

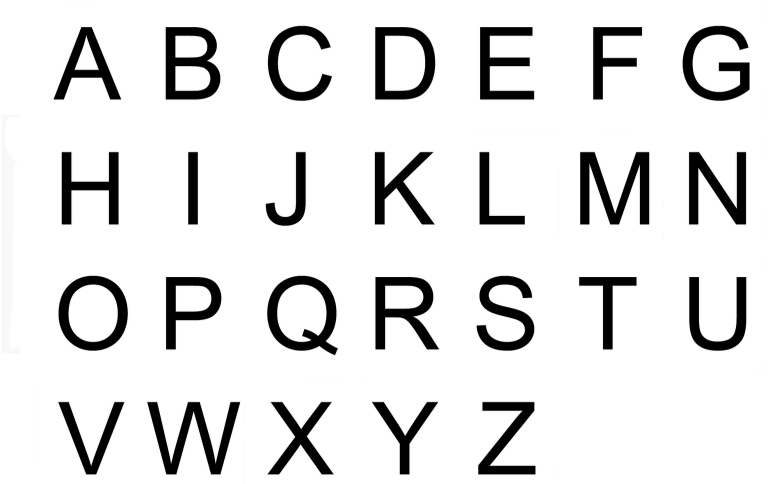


Figure 3. The Original Training Image

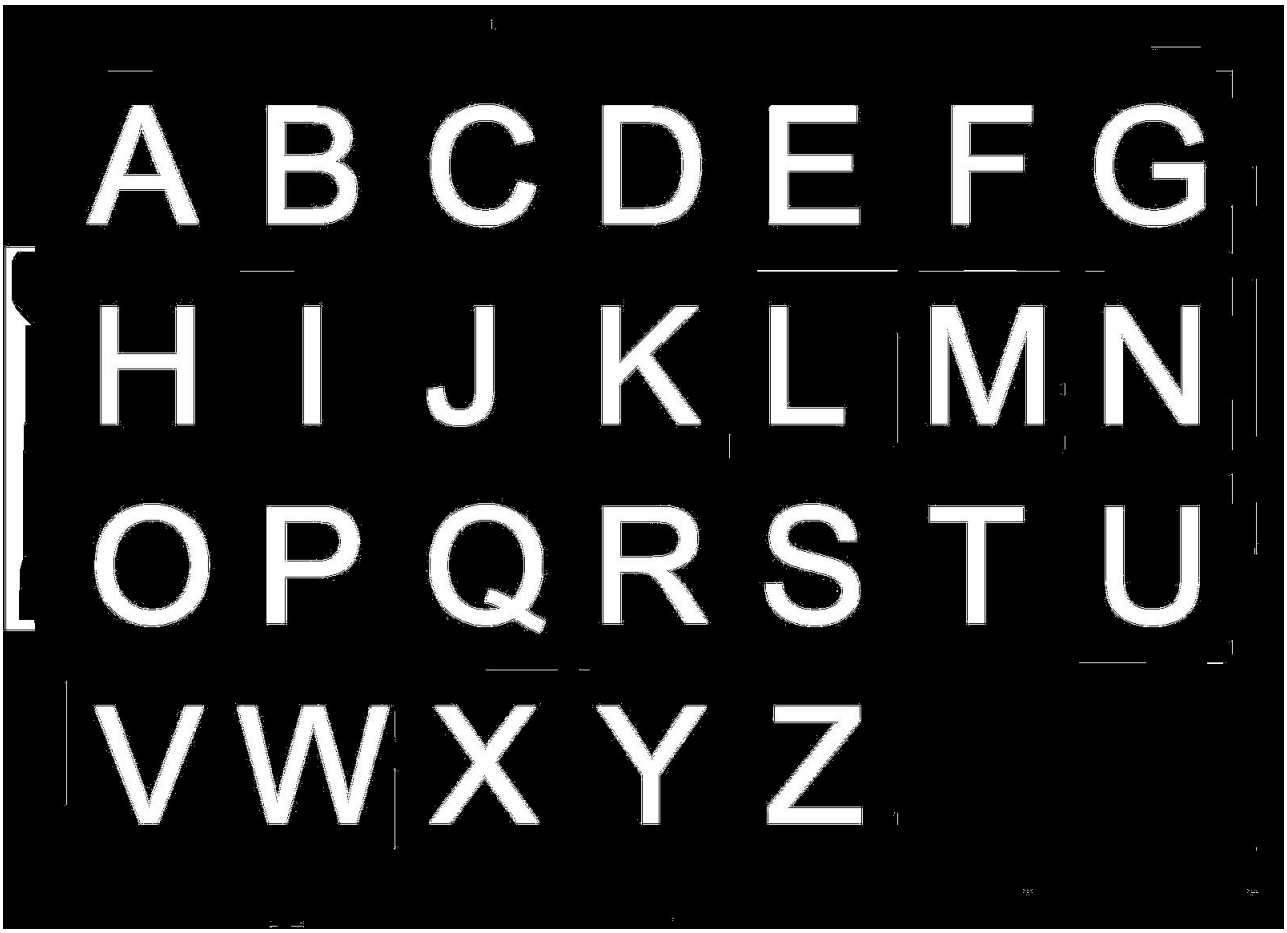


Figure 4. First Iteration of Image Segmentation of Training Image Based On Otsu's Threshold.

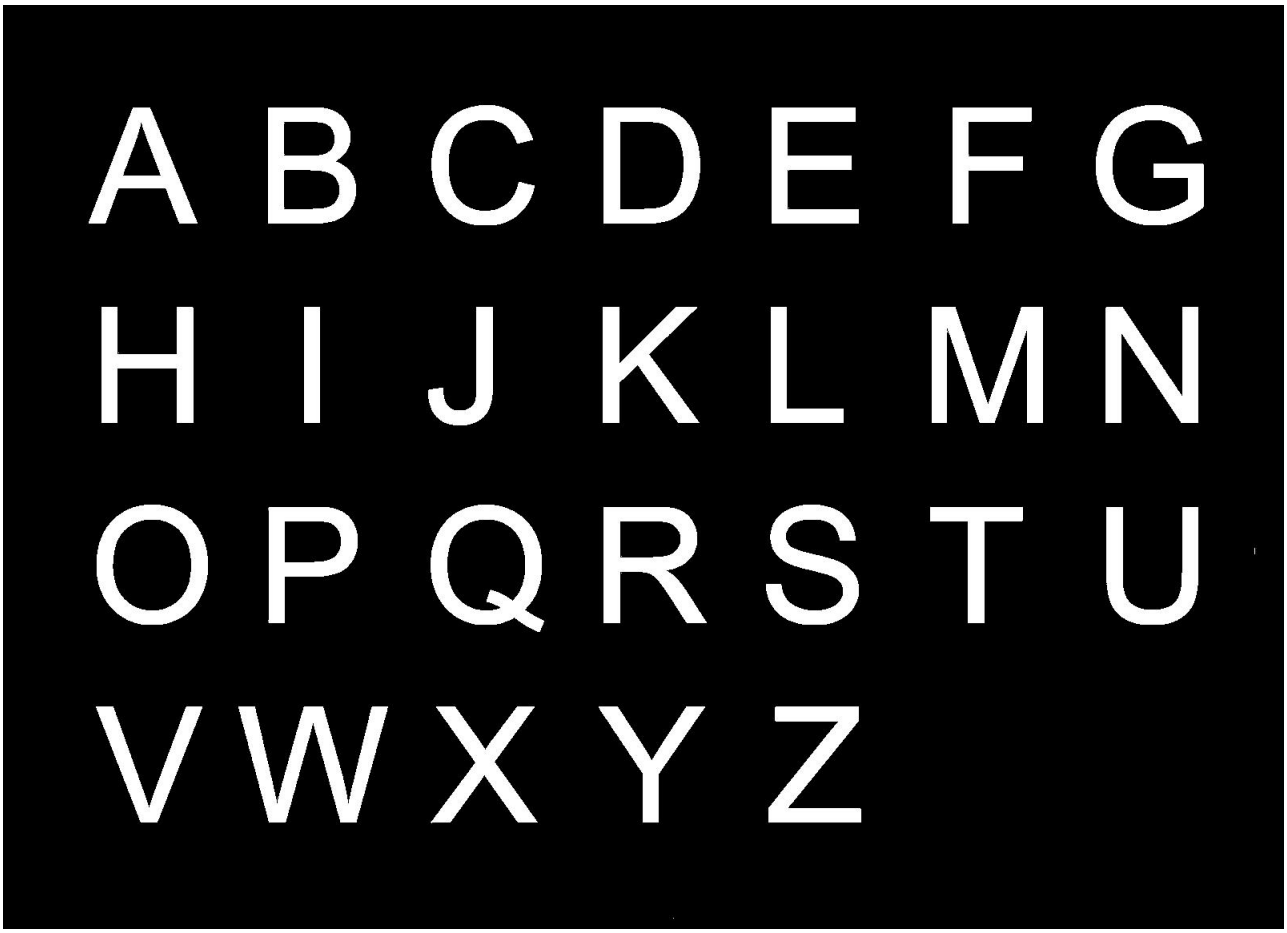


Figure 5. Second Iteration of Image Segmentation of Training Image Based On Otsu's Threshold. (result improved)

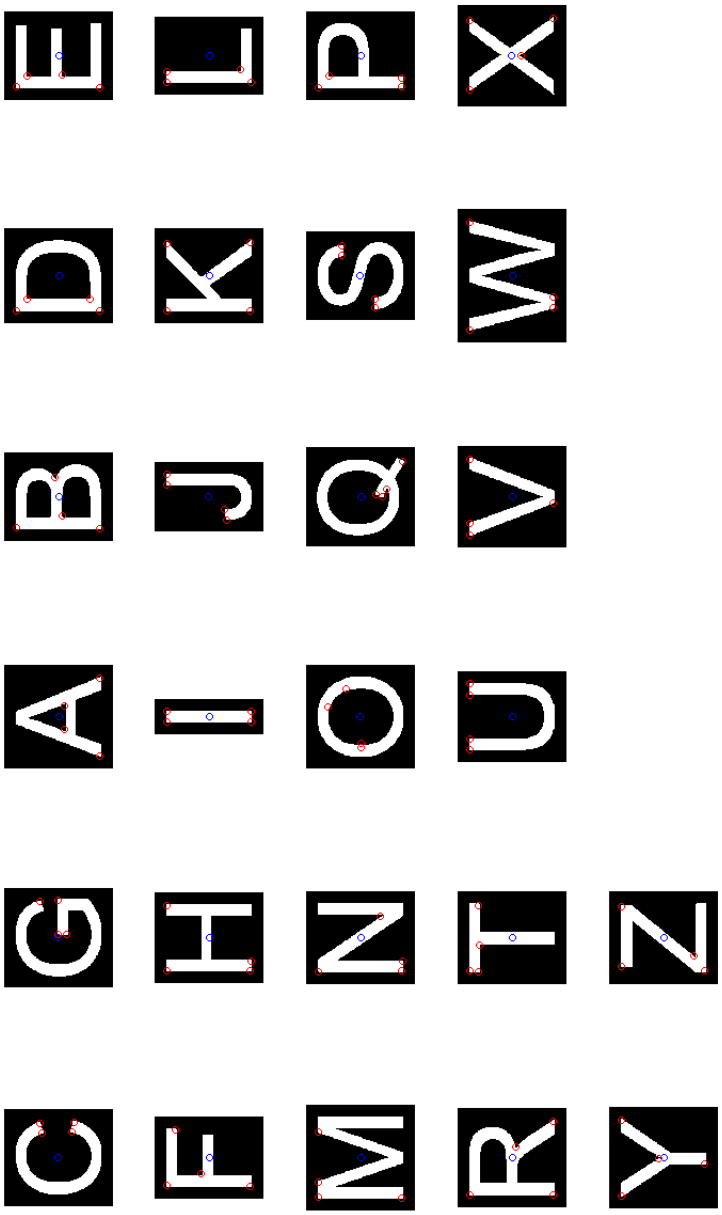


Figure 6. Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

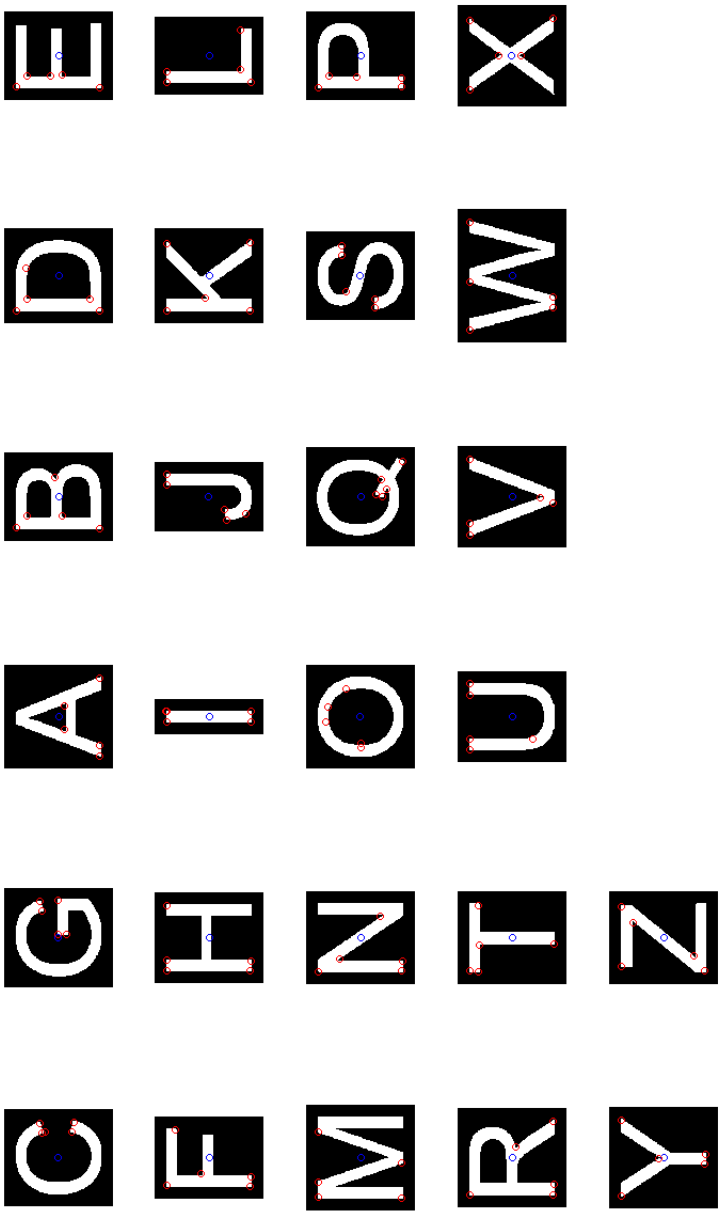


Figure 7. Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

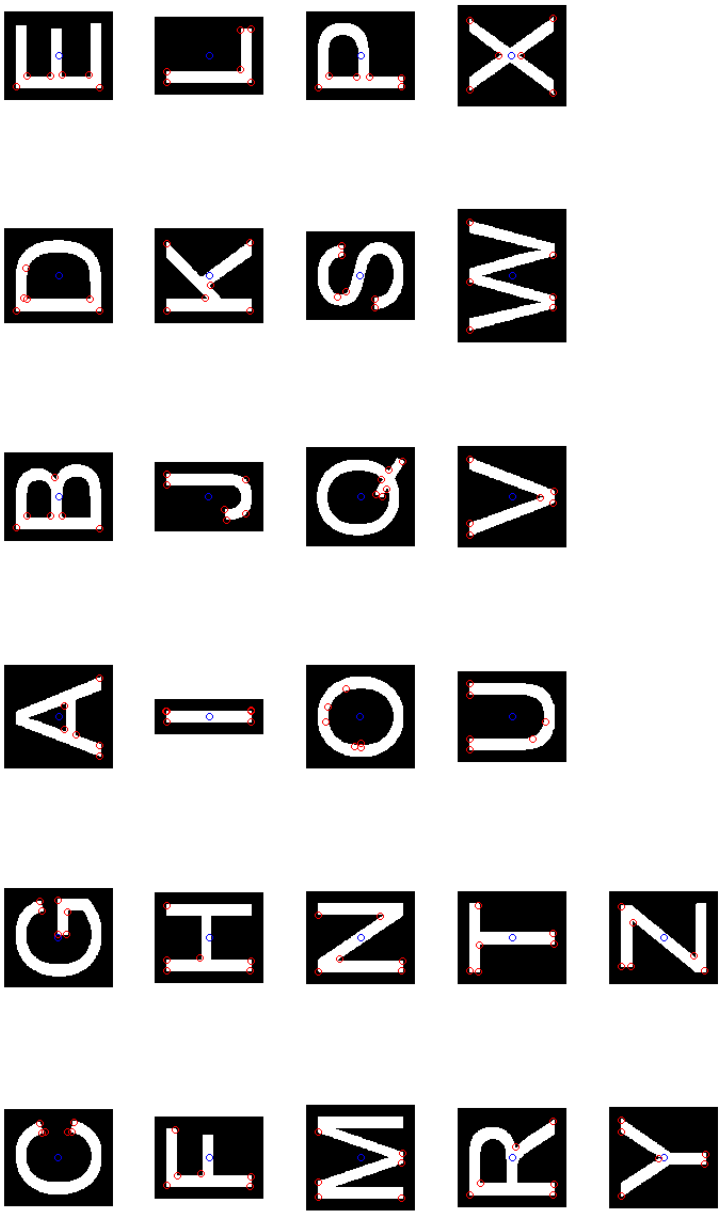


Figure 8. Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

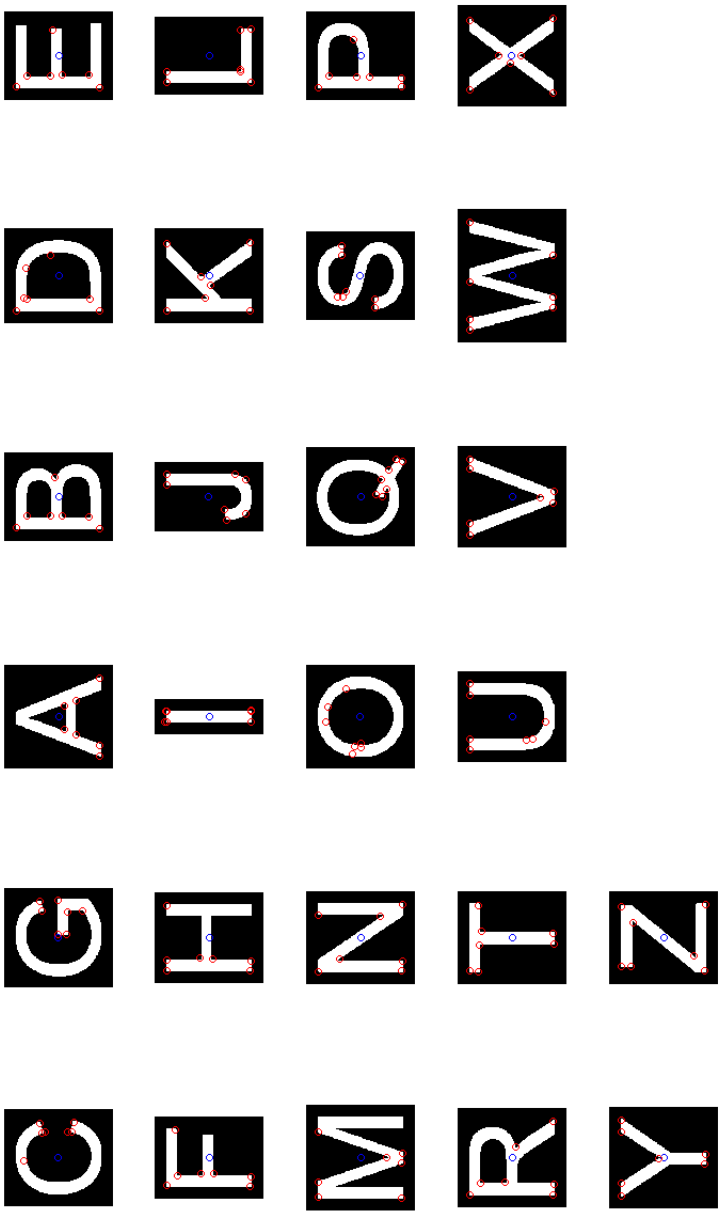


Figure 9. Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

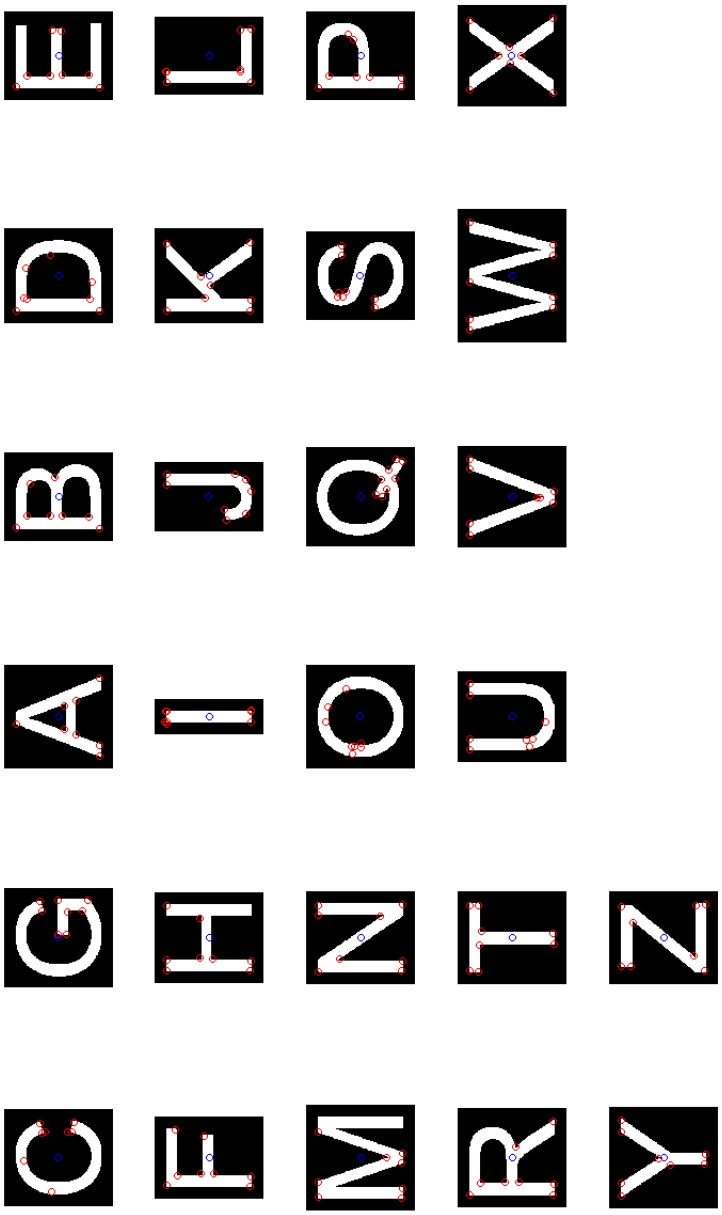


Figure 10. Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 8$

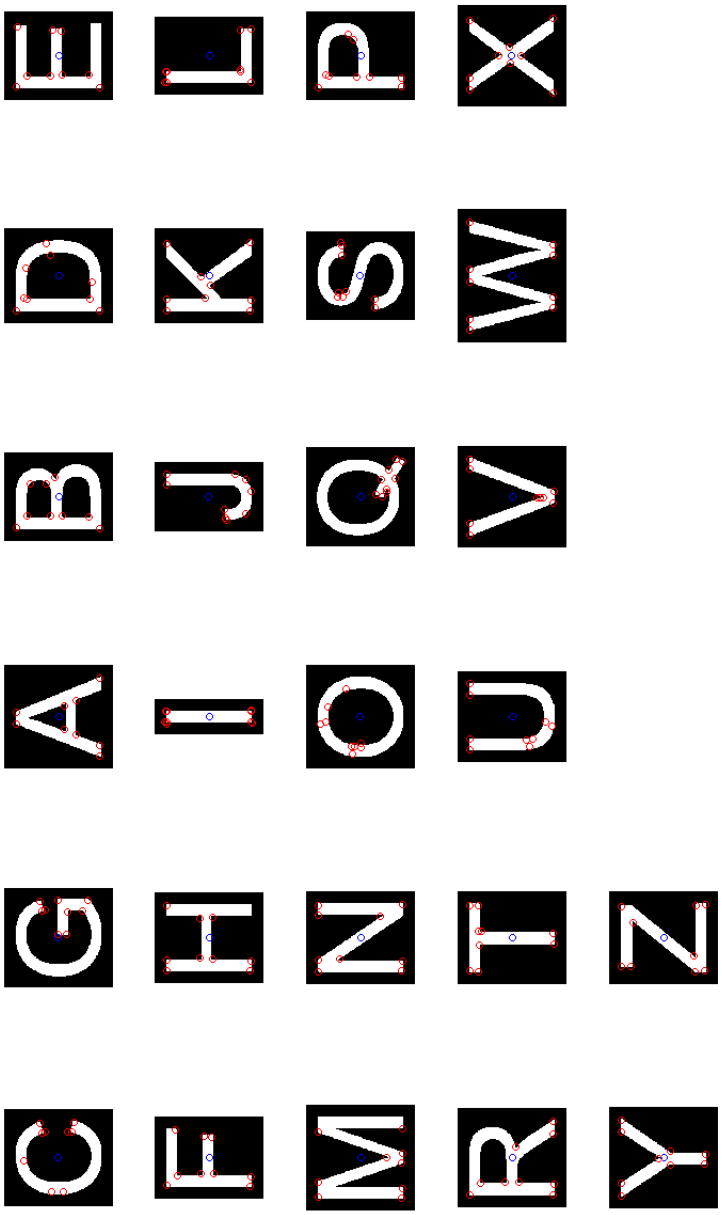


Figure 11. Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 9$

6.4 `Image0.jpg`: 180 Degree Rotation of the `Training.jpg`

Figure 12. The Original `Image0.jpg`

Figure 13. First Iteration of Image Segmentation of `Image0.jpg` Based On Otsu's Threshold.

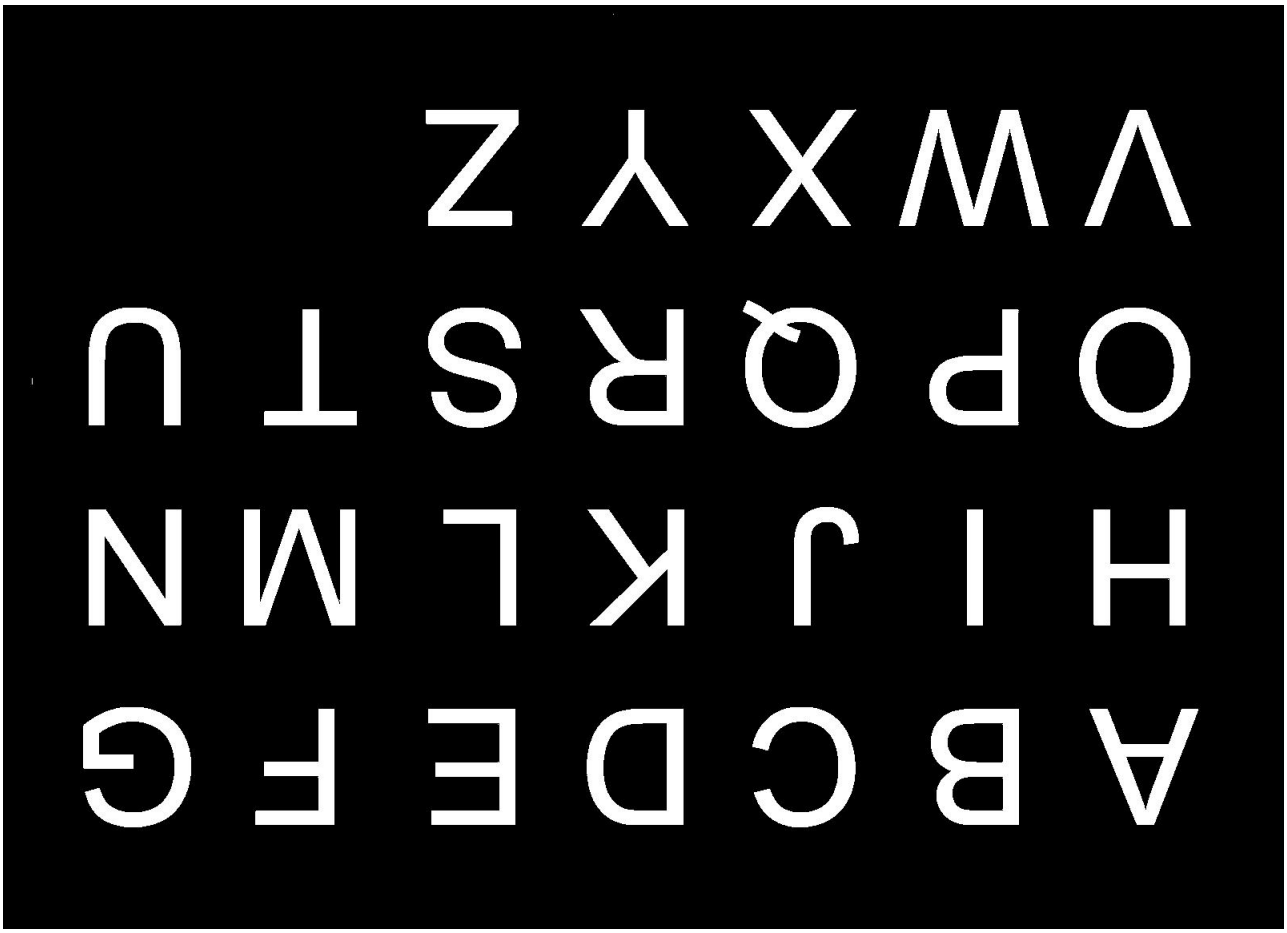


Figure 14. Second Iteration of Image Segmentation of `Image0.jpg` Based On Otsu's Threshold. (result improved)

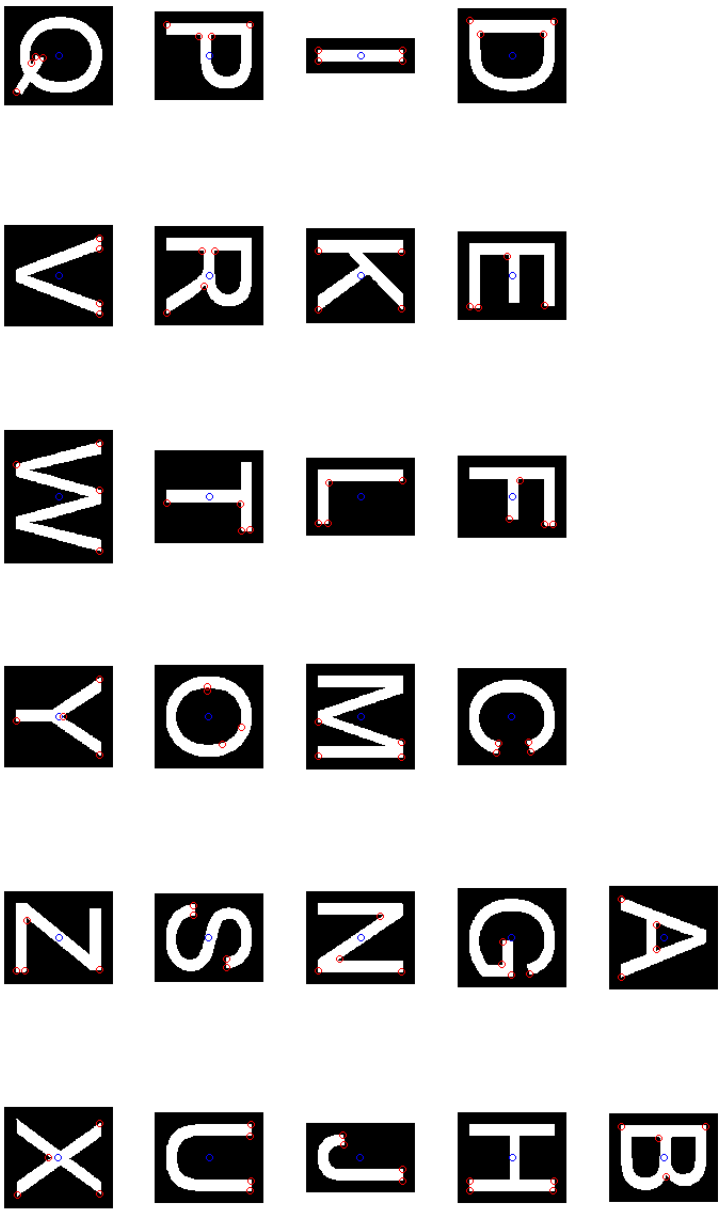


Figure 15. `Image0.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

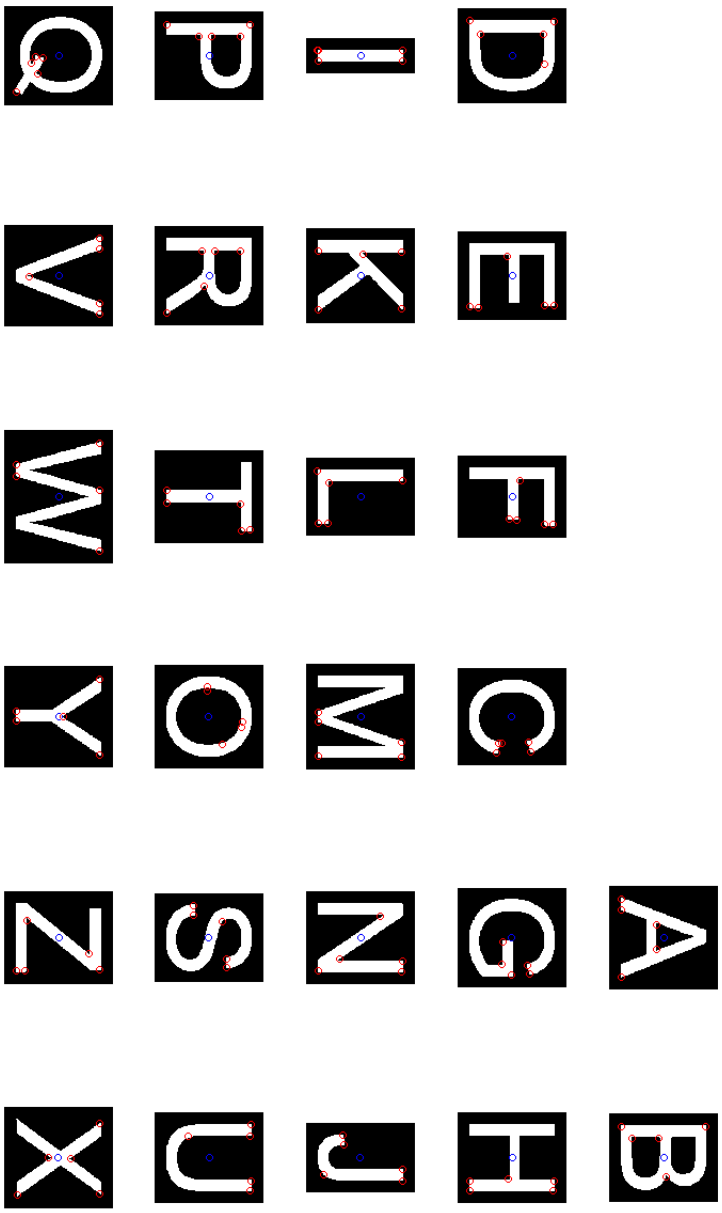


Figure 16. `Image0.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

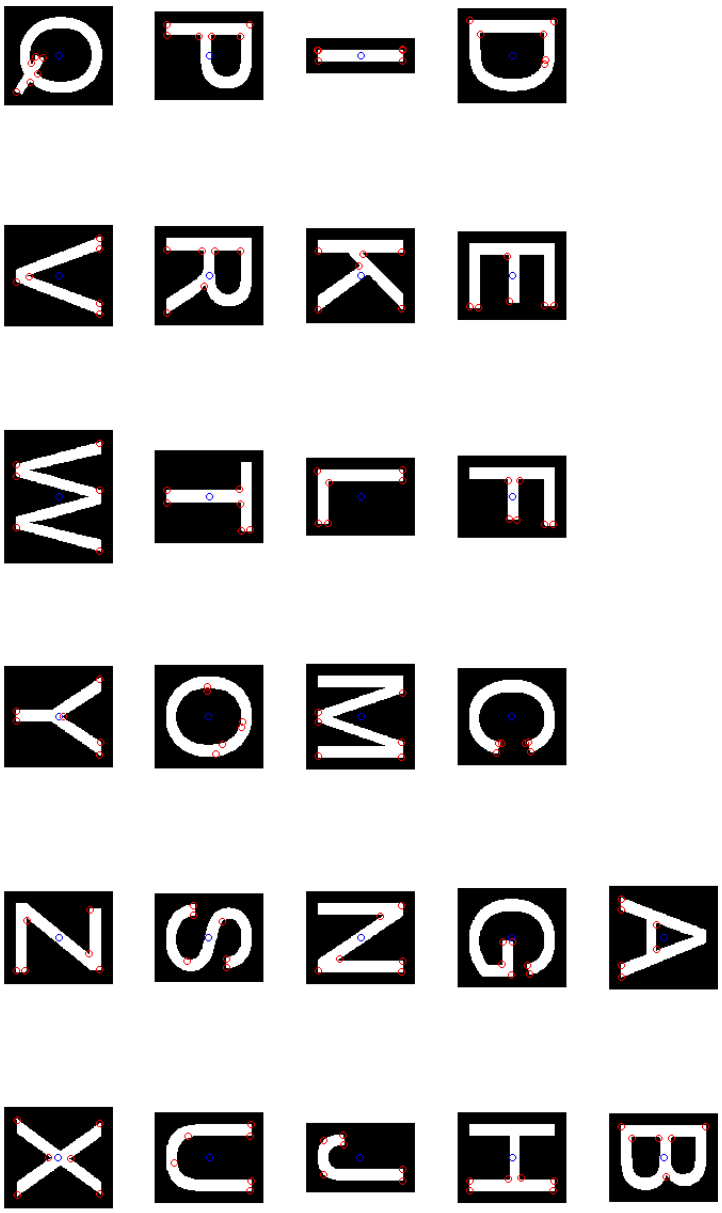


Figure 17. `Image0.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

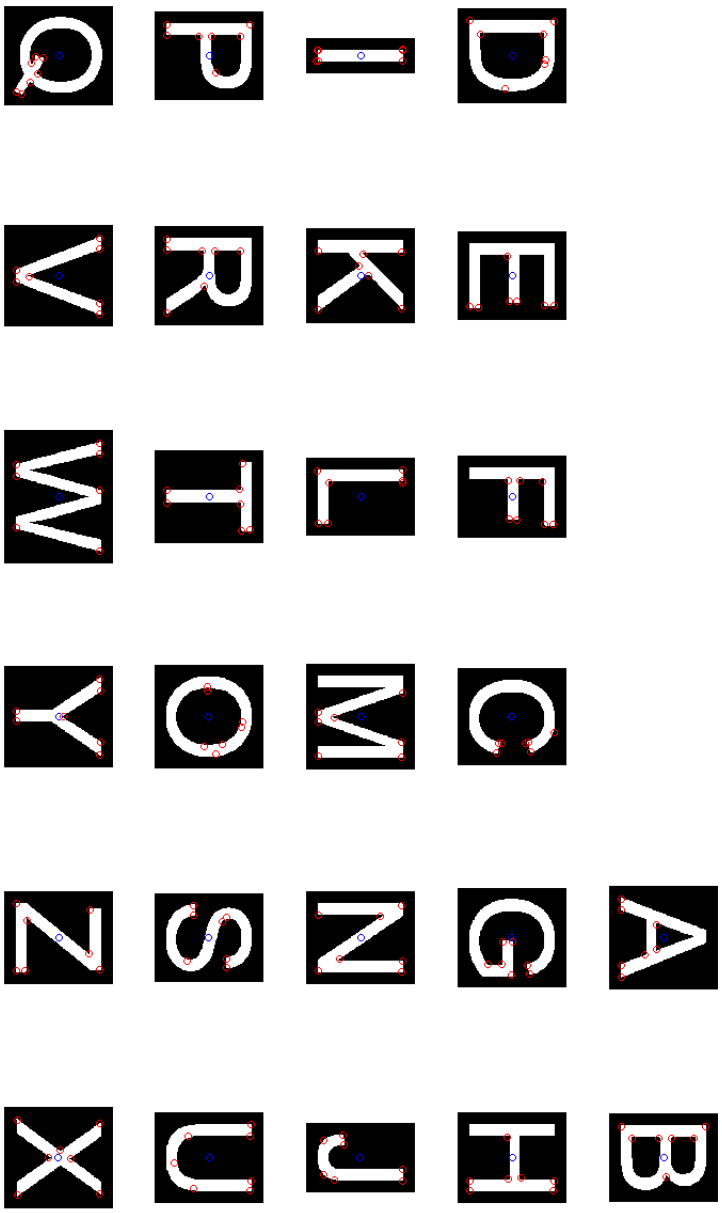


Figure 18. `Image0.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

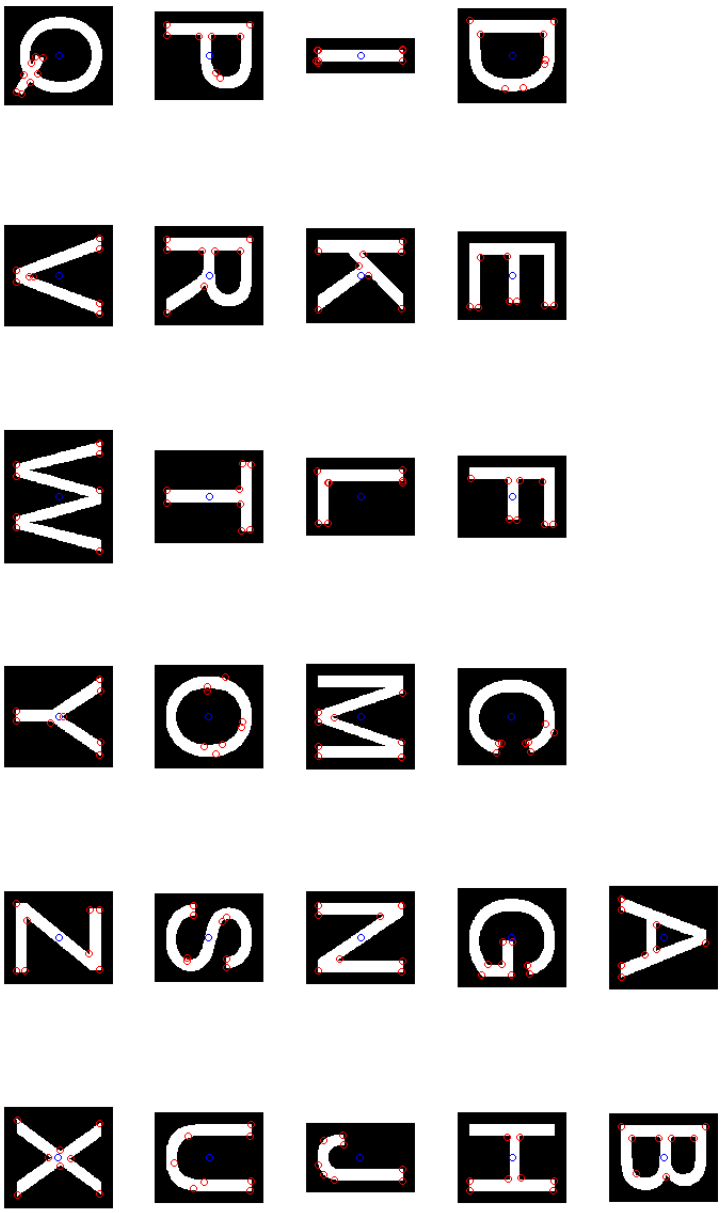


Figure 19. `Image0.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 8$

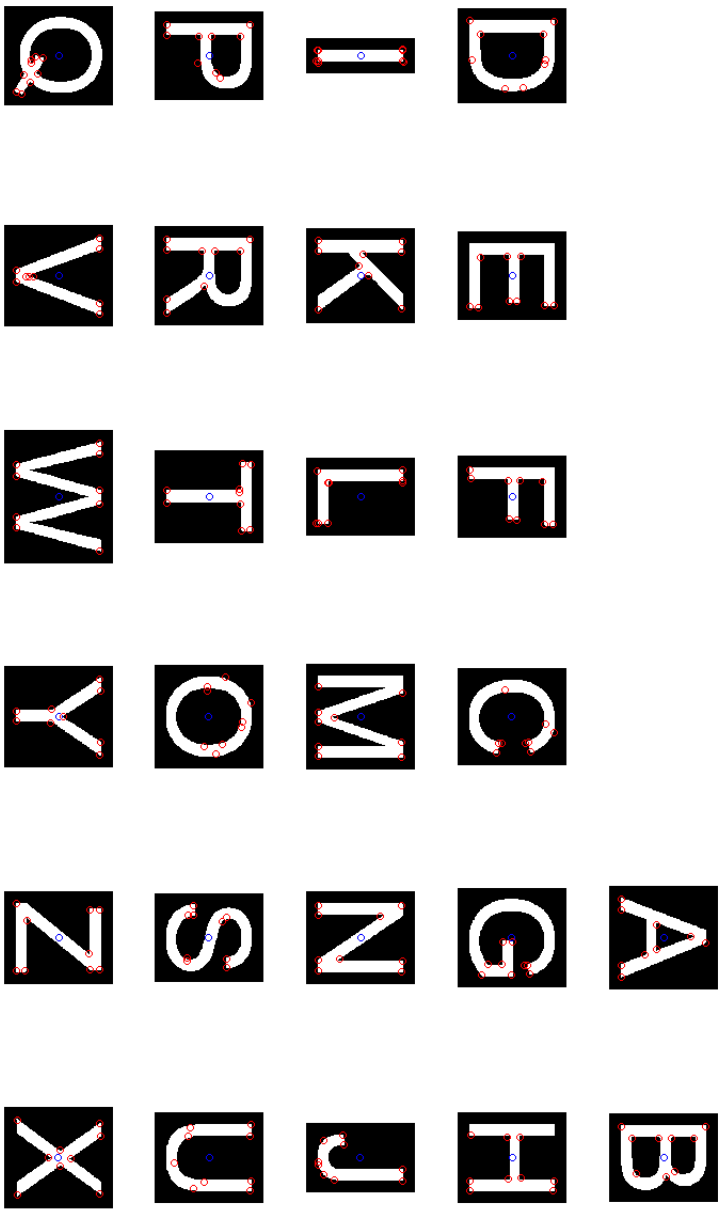


Figure 20. `Image0.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 9$

6.5 `Image1.jpg`: With Pre-Processing

Figure 21. The Original `Image1.jpg`

Figure 22. The processed `Image1.jpg`

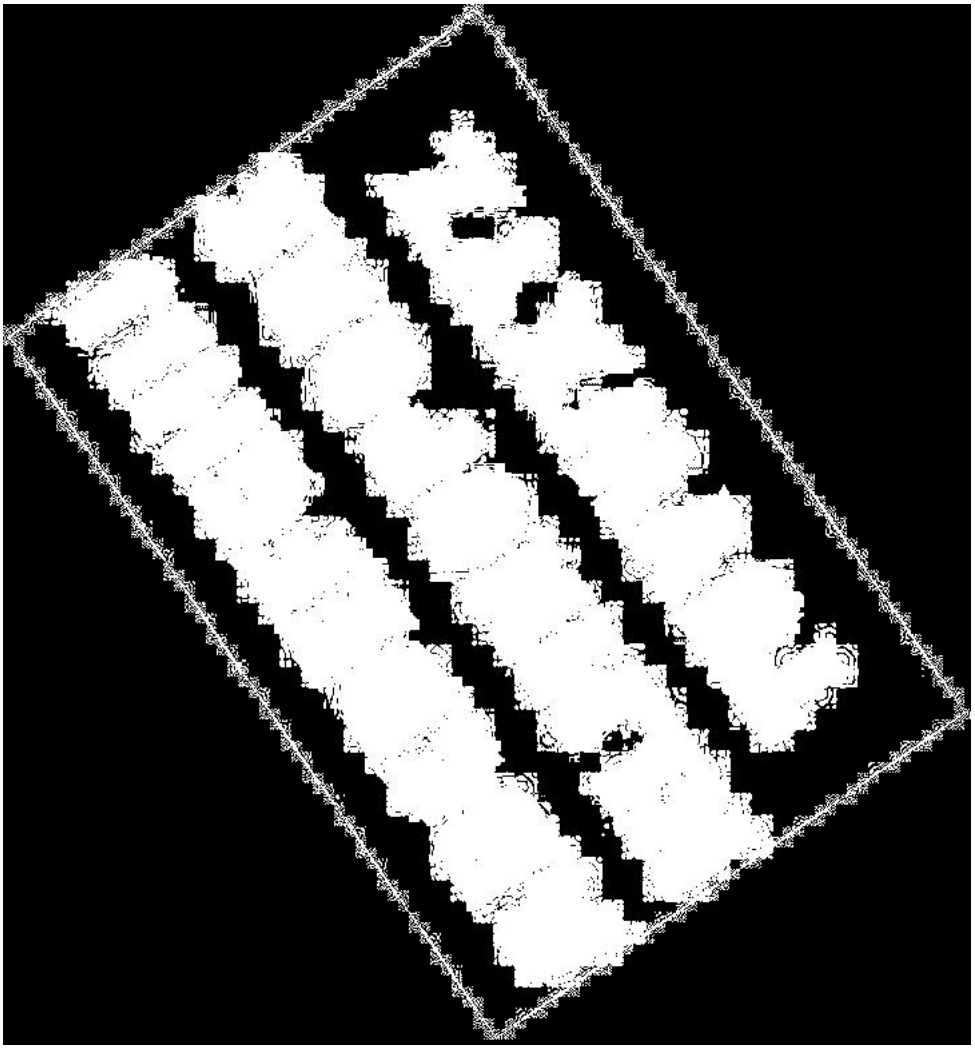


Figure 23. First Iteration of Image Segmentation of `Image1.jpg` Based On Otsu's Threshold.

Figure 24. Second Iteration of Image Segmentation of `Image1.jpg` Based On Otsu's Threshold. We can see that several characters were missed because their colors are too close to white. The frame was also mistakenly taken into account, while letter 'V' and letter 'W' is too close to each other and they were mistakenly segmented into one letter.

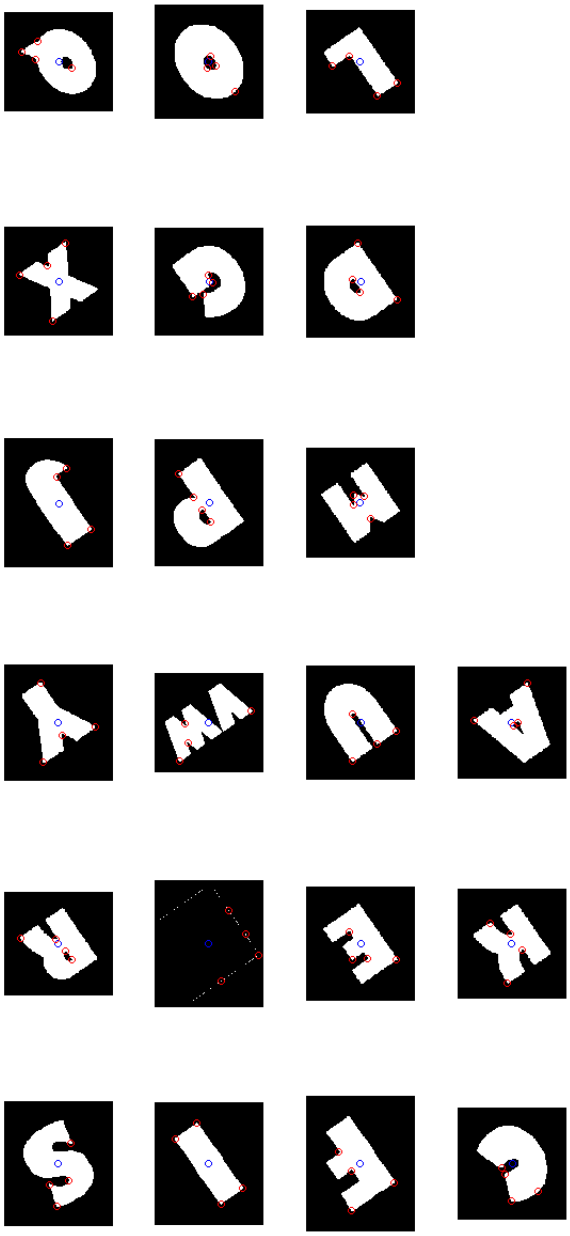


Figure 25. `Image1.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

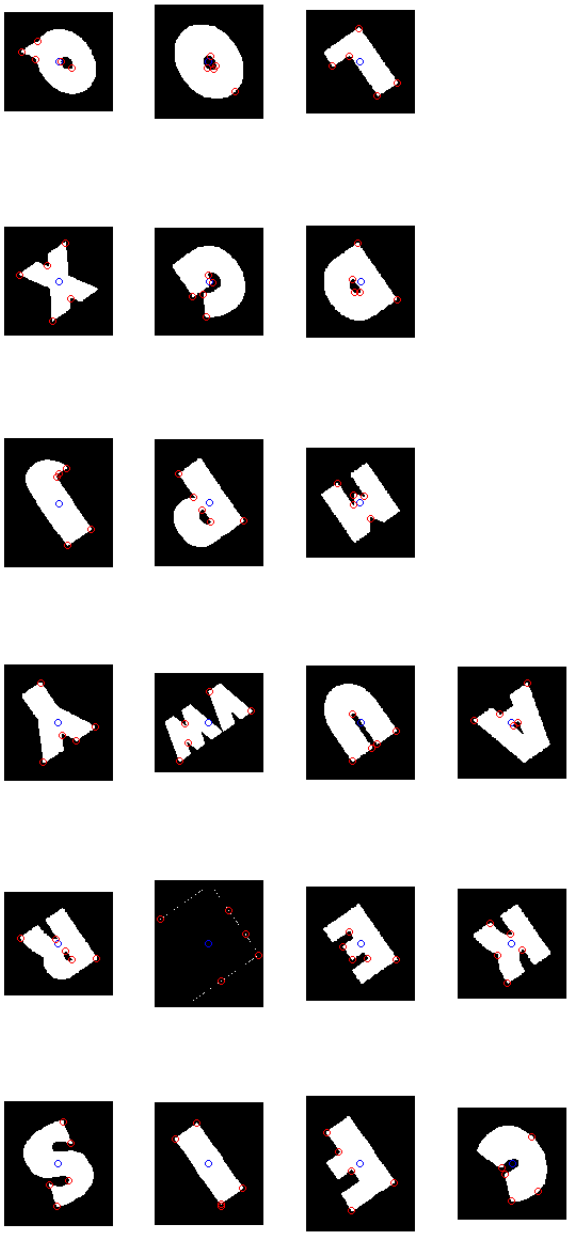


Figure 26. `Image1.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

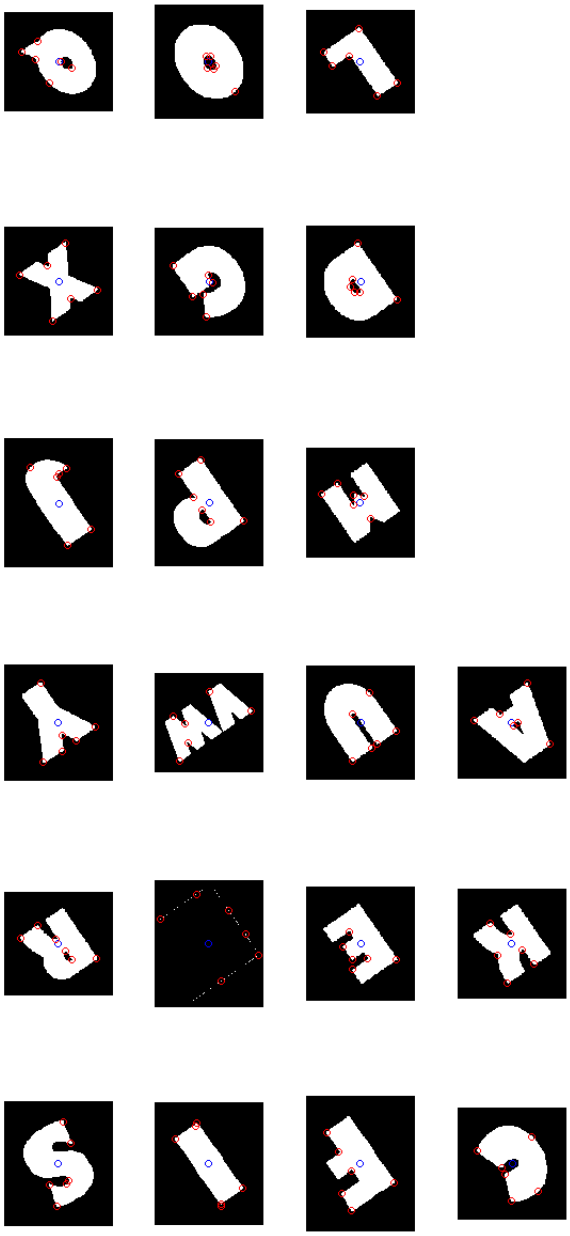


Figure 27. `Image1.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

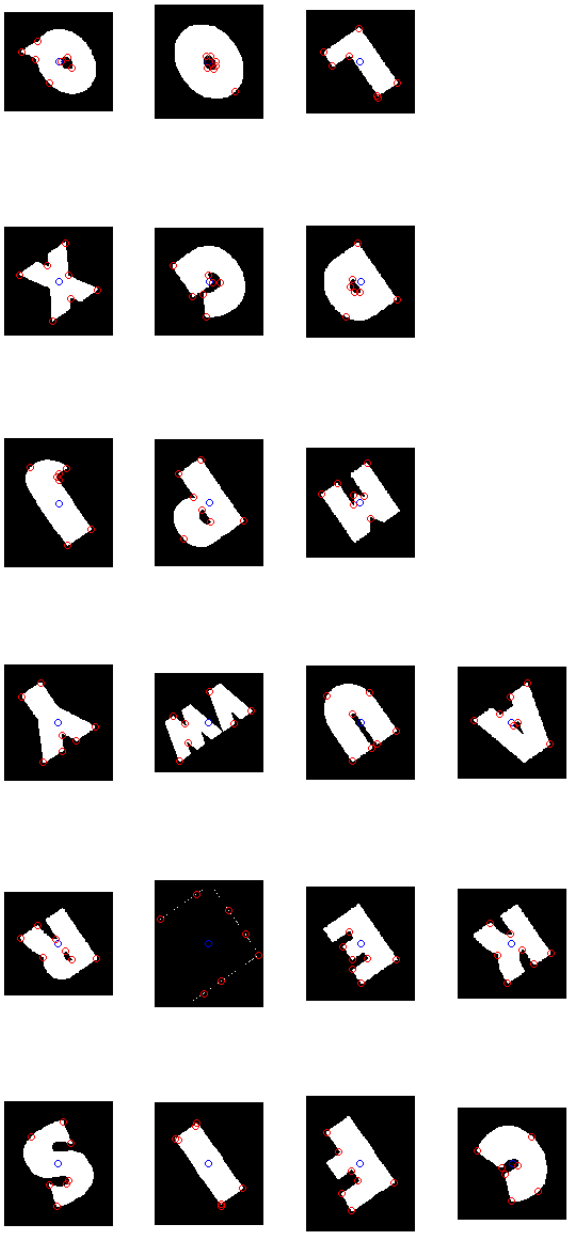


Figure 28. `Image1.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

6.6 Image2.jpg

Figure 29. The Original Image2.jpg

Figure 30. First Iteration of Image Segmentation of `Image2.jpg` Based On Otsu's Threshold.

Figure 31. Second Iteration of Image Segmentation of `Image2.jpg` Based On Otsu's Threshold. (result improved)

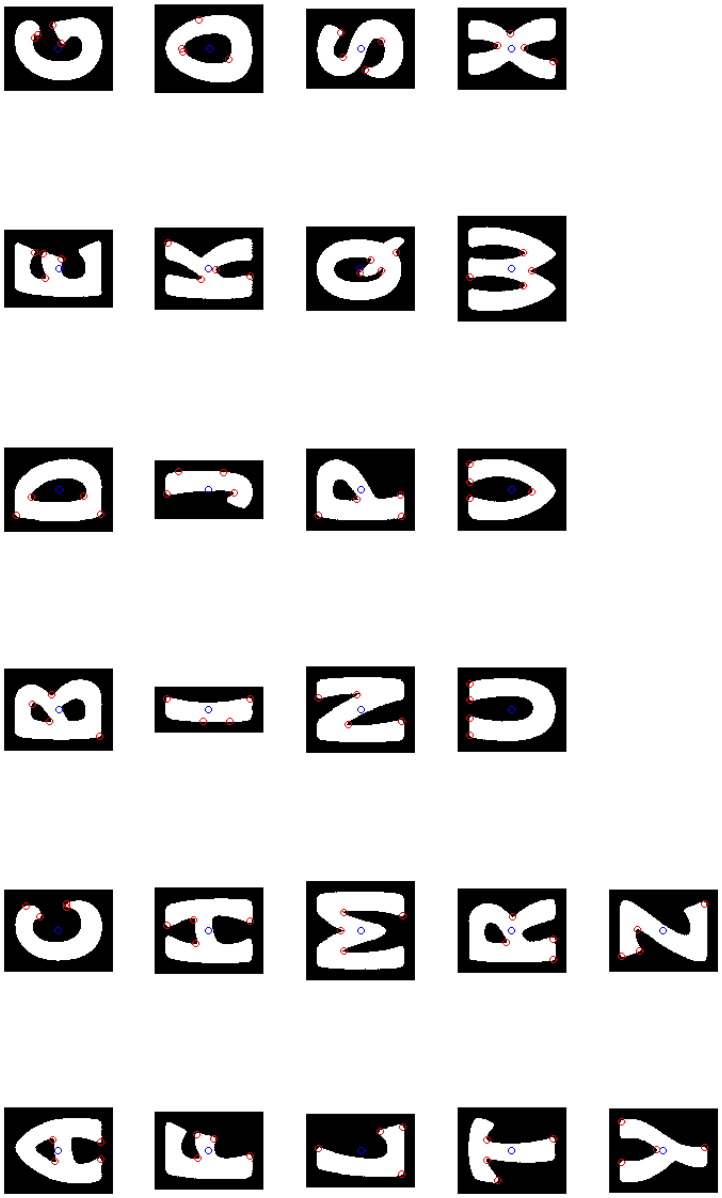


Figure 32. `Image2.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

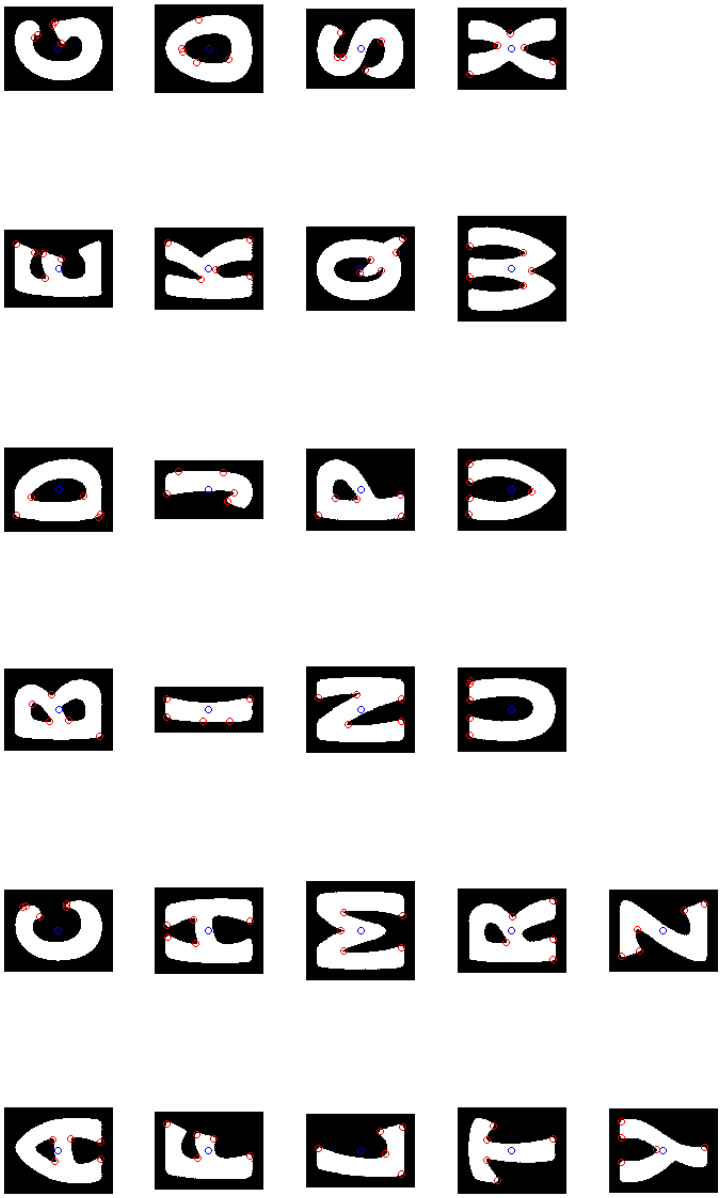


Figure 33. `Image2.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

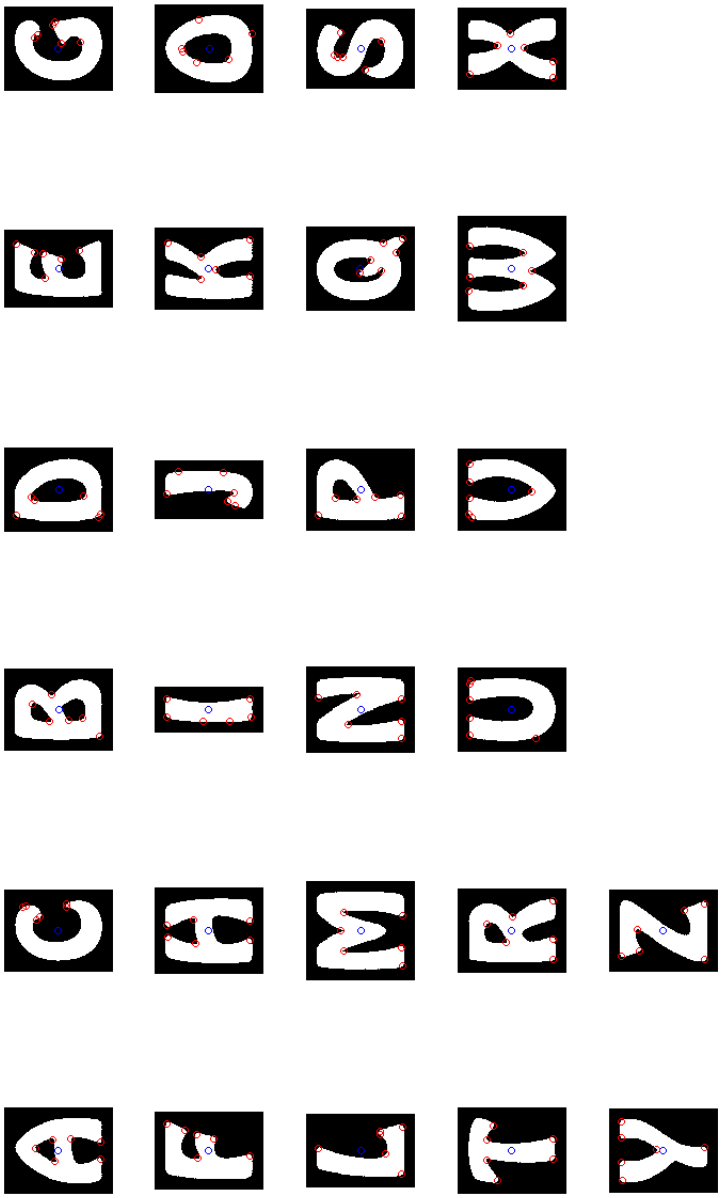


Figure 34. `Image2.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

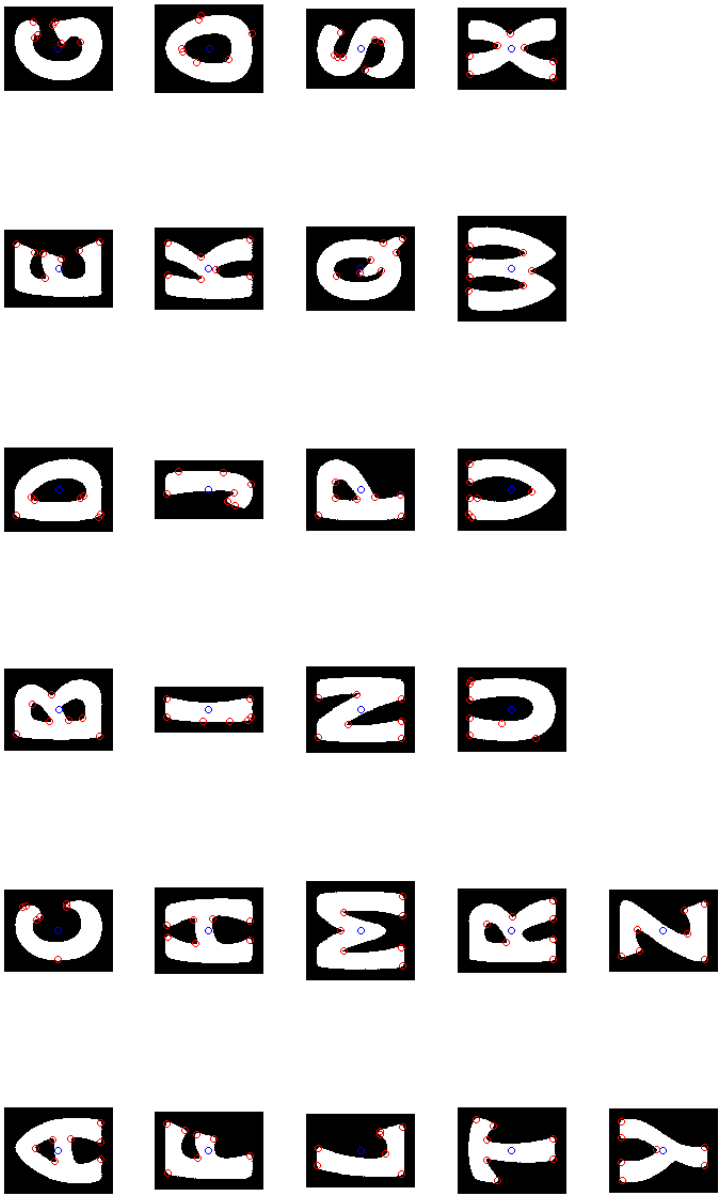


Figure 35. `Image2.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

6.7 Image3.jpg

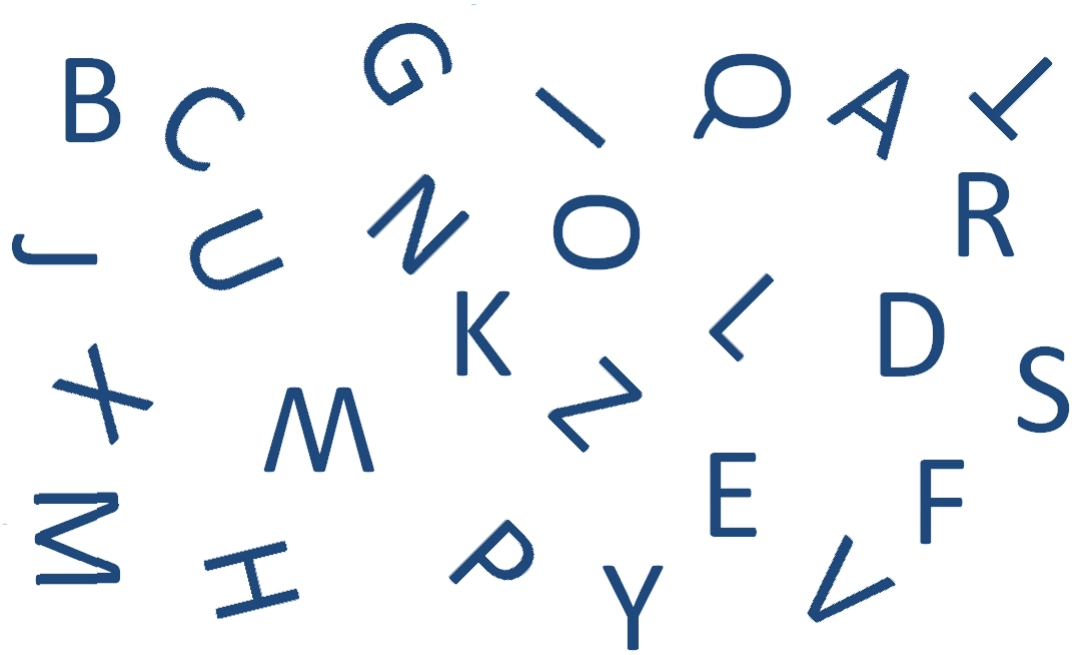


Figure 36. The Original Image3.jpg

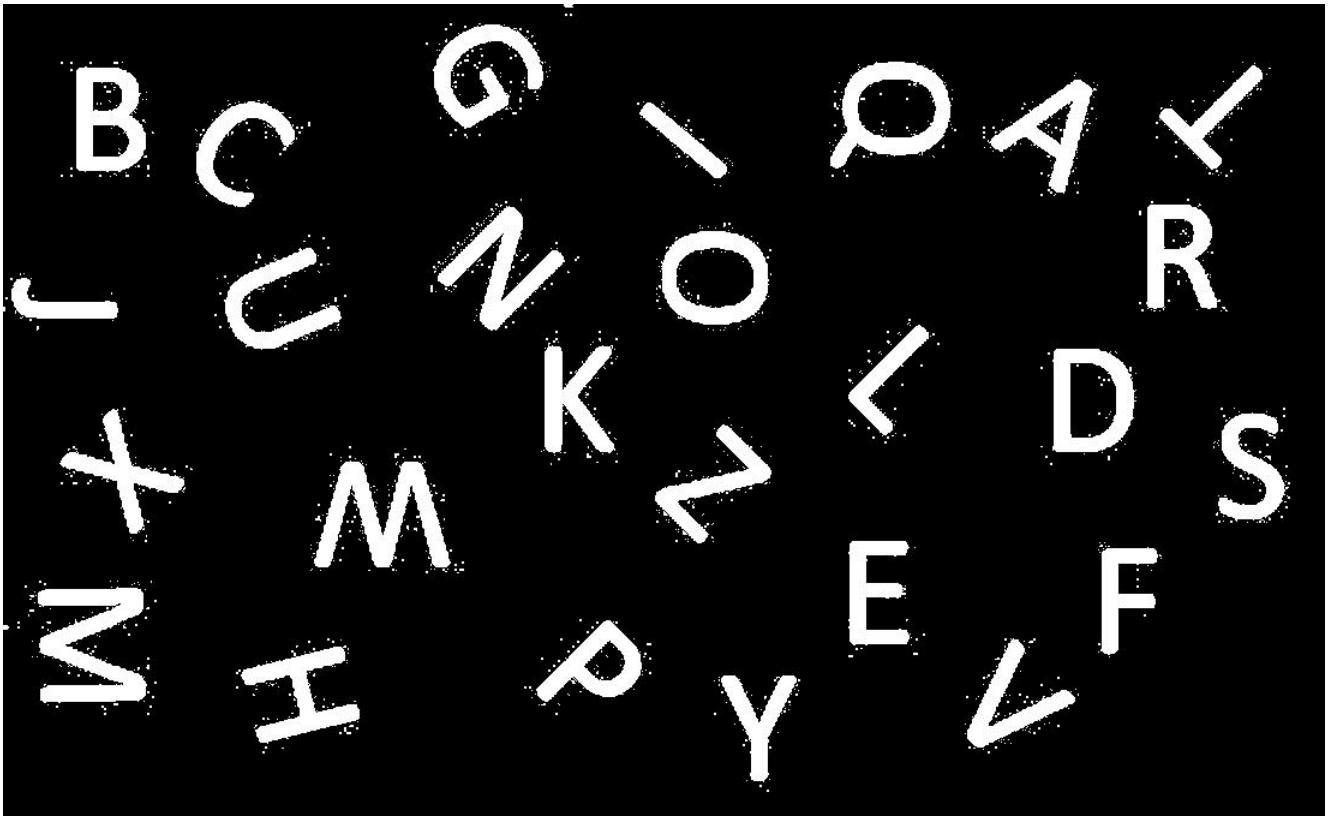


Figure 37. First Iteration of Image Segmentation of Image3.jpg Based On Otsu's Threshold.

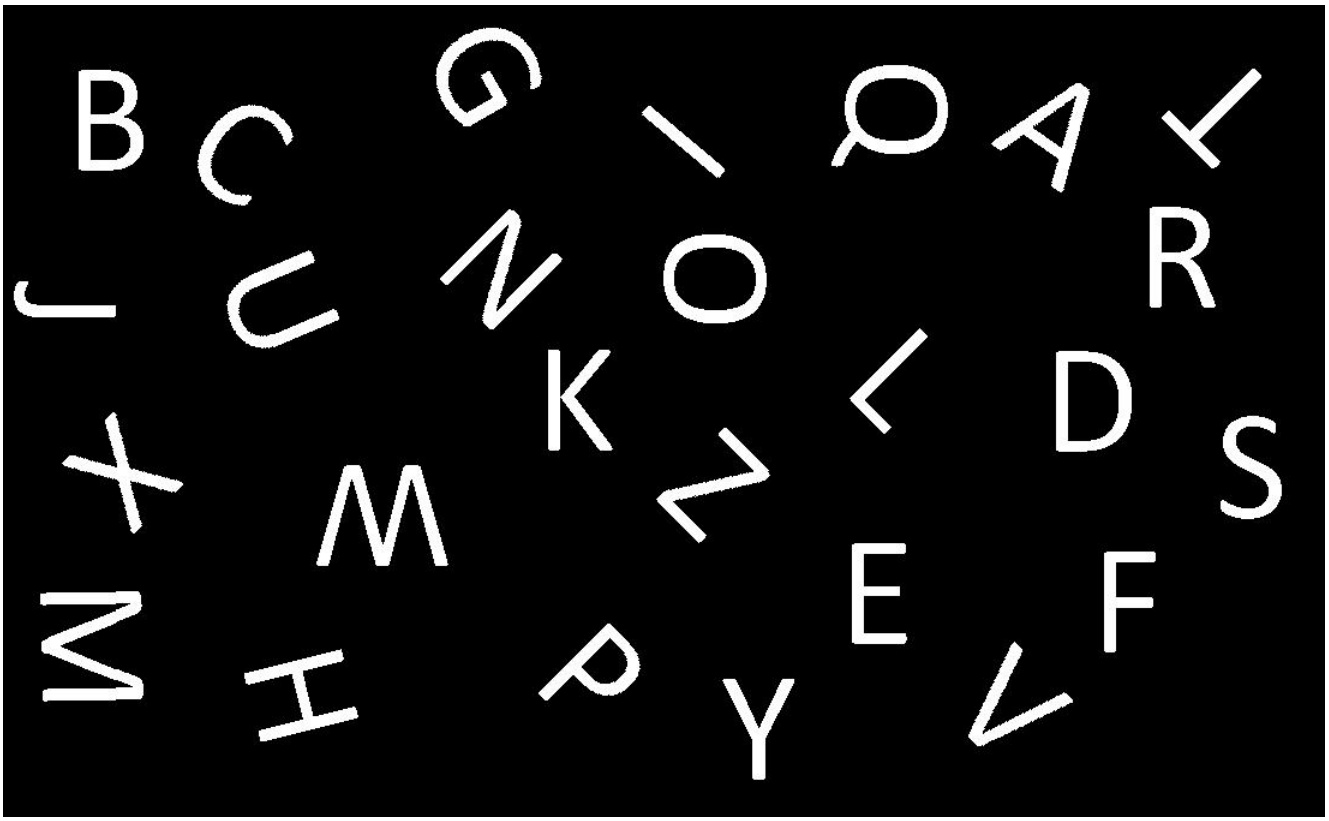


Figure 38. Second Iteration of Image Segmentation of `Image3.jpg` Based On Otsu's Threshold. (result improved)

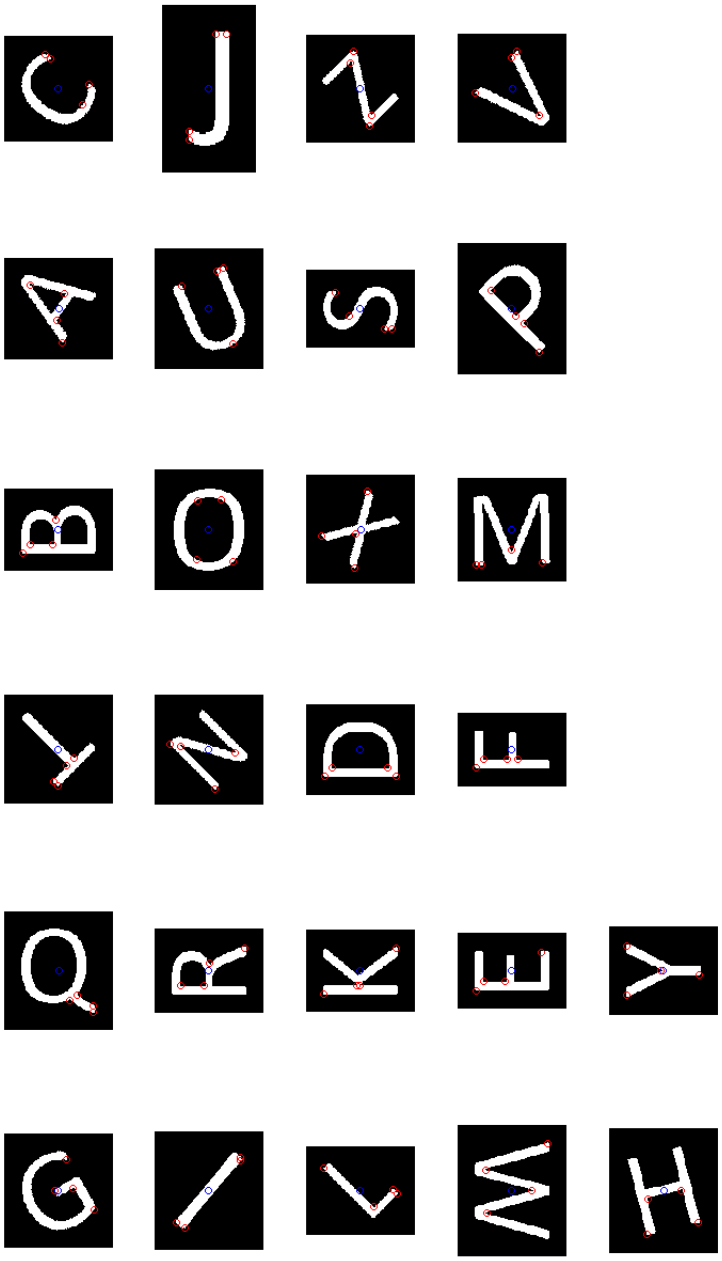


Figure 39. `Image3.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

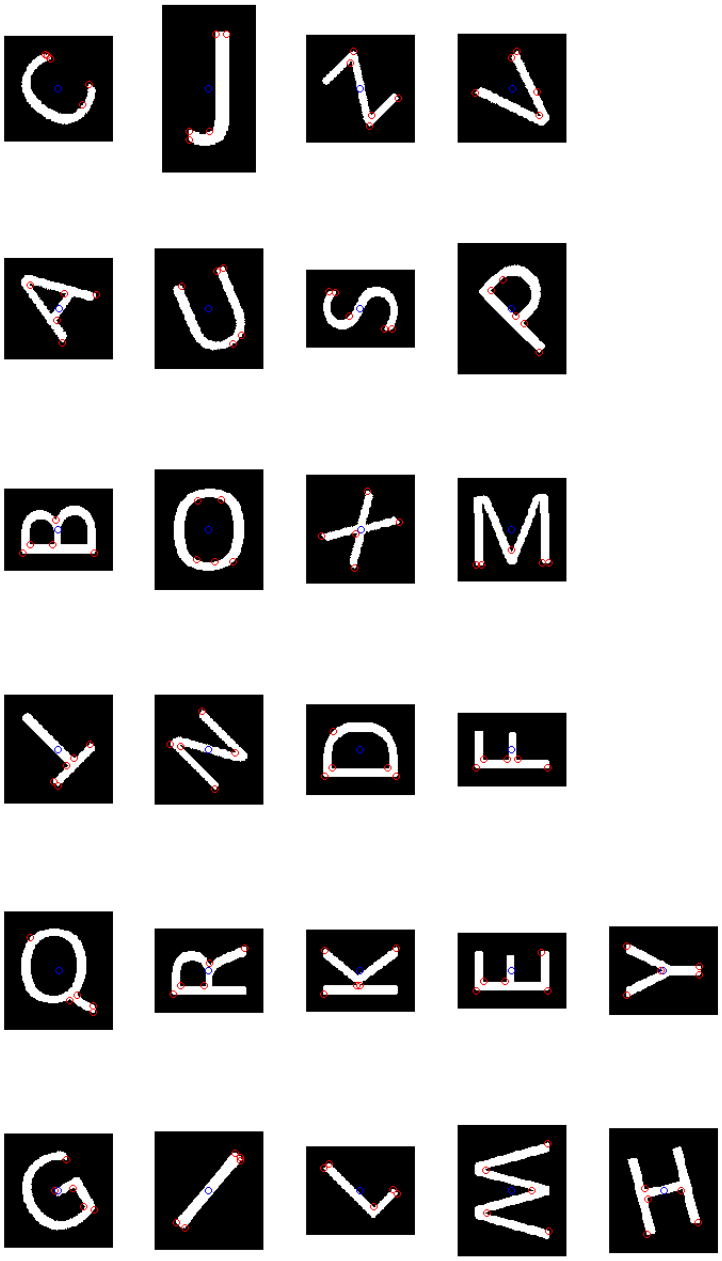


Figure 40. `Image3.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

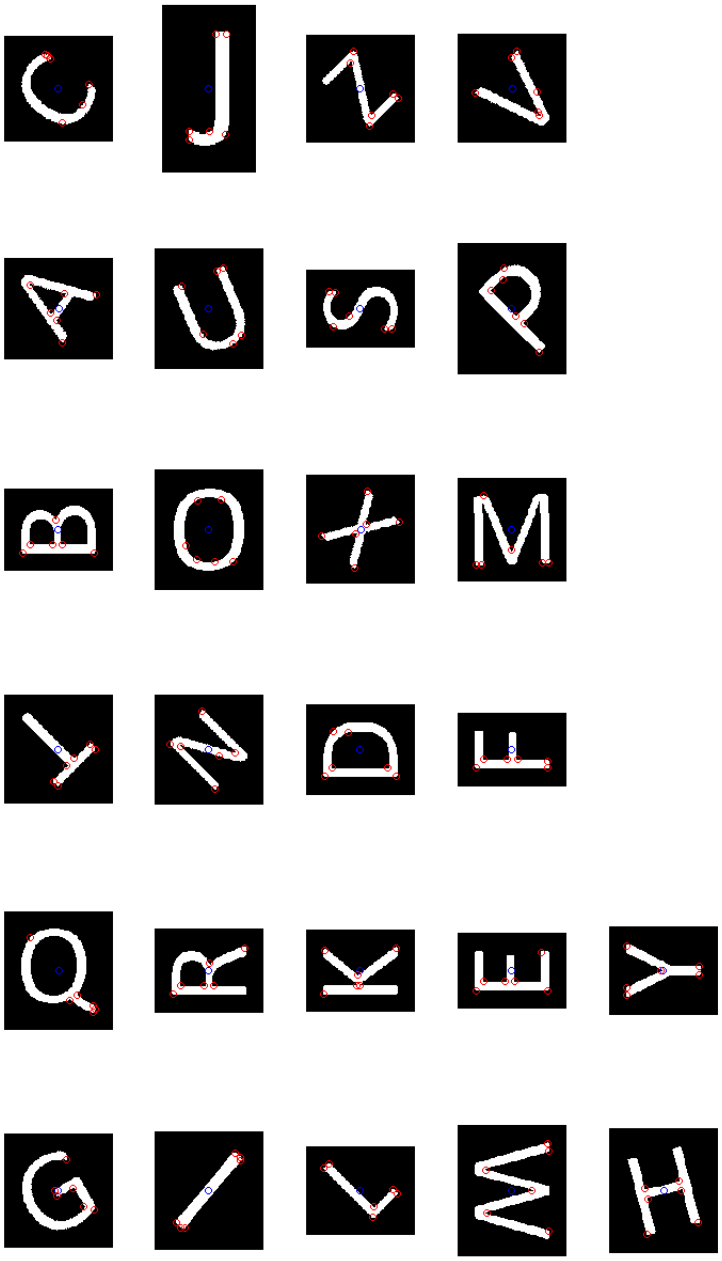


Figure 41. `Image3.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

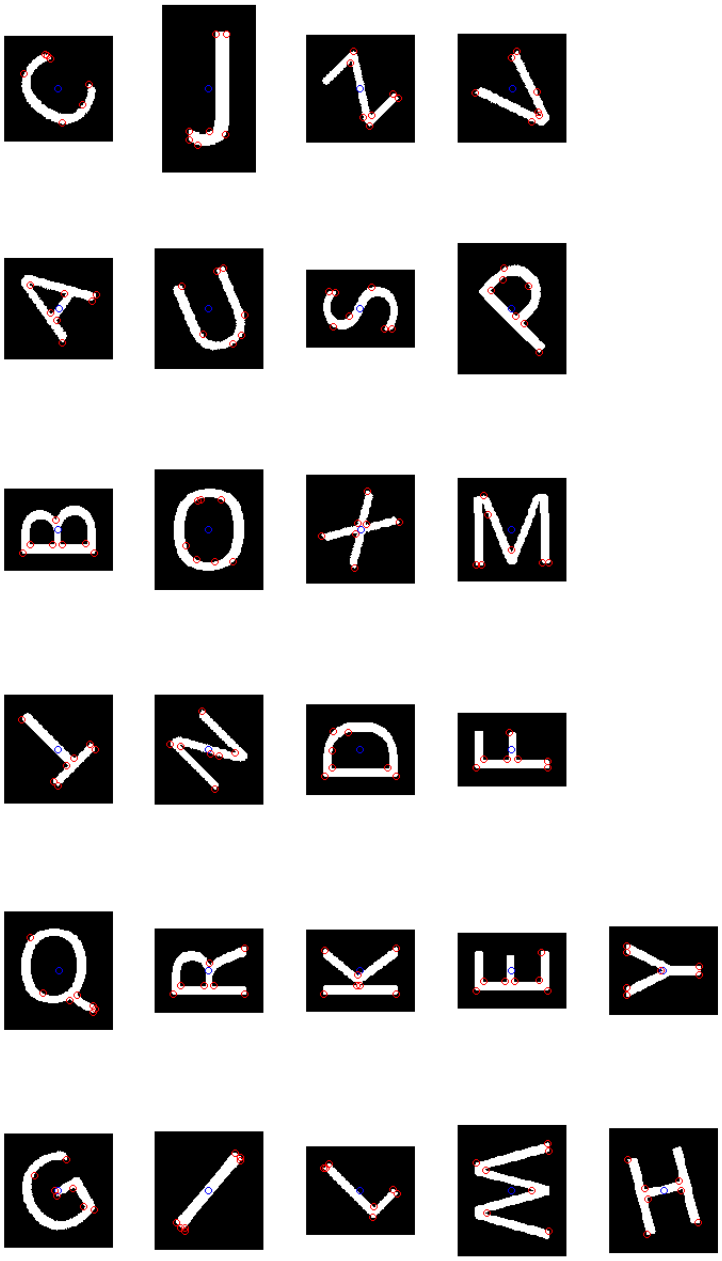


Figure 42. `Image3.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

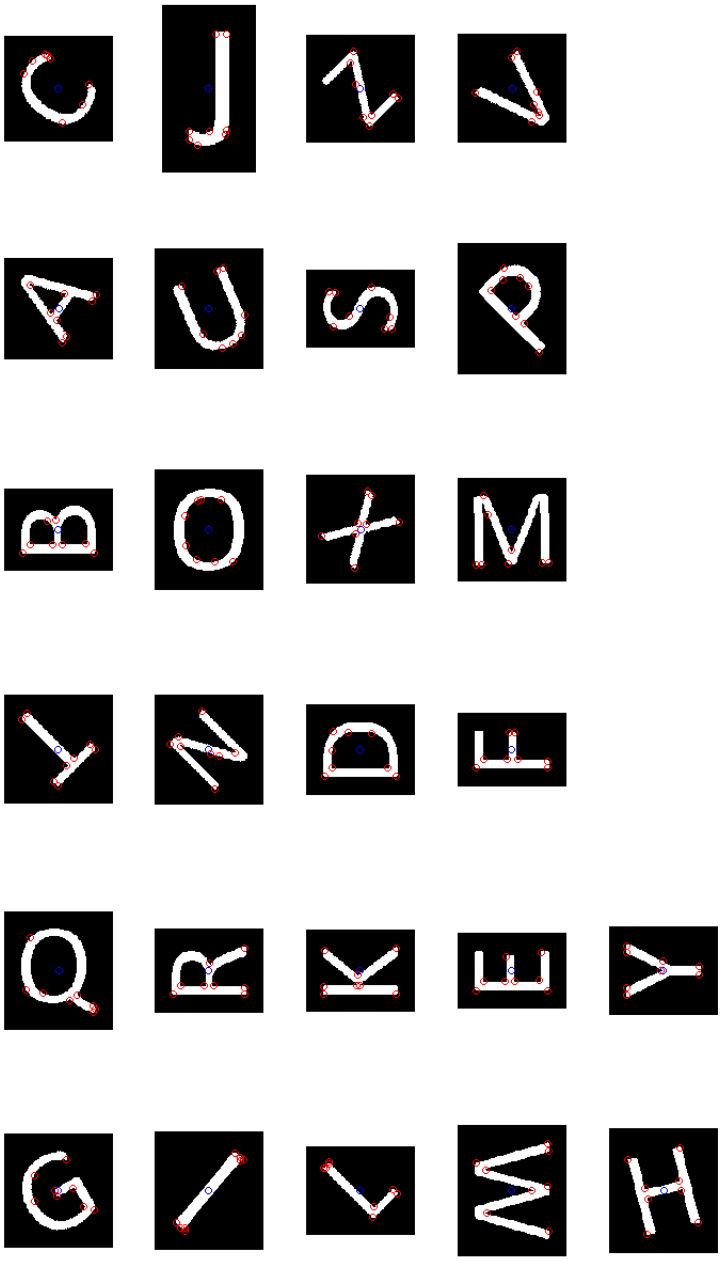


Figure 43. `Image3.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 8$

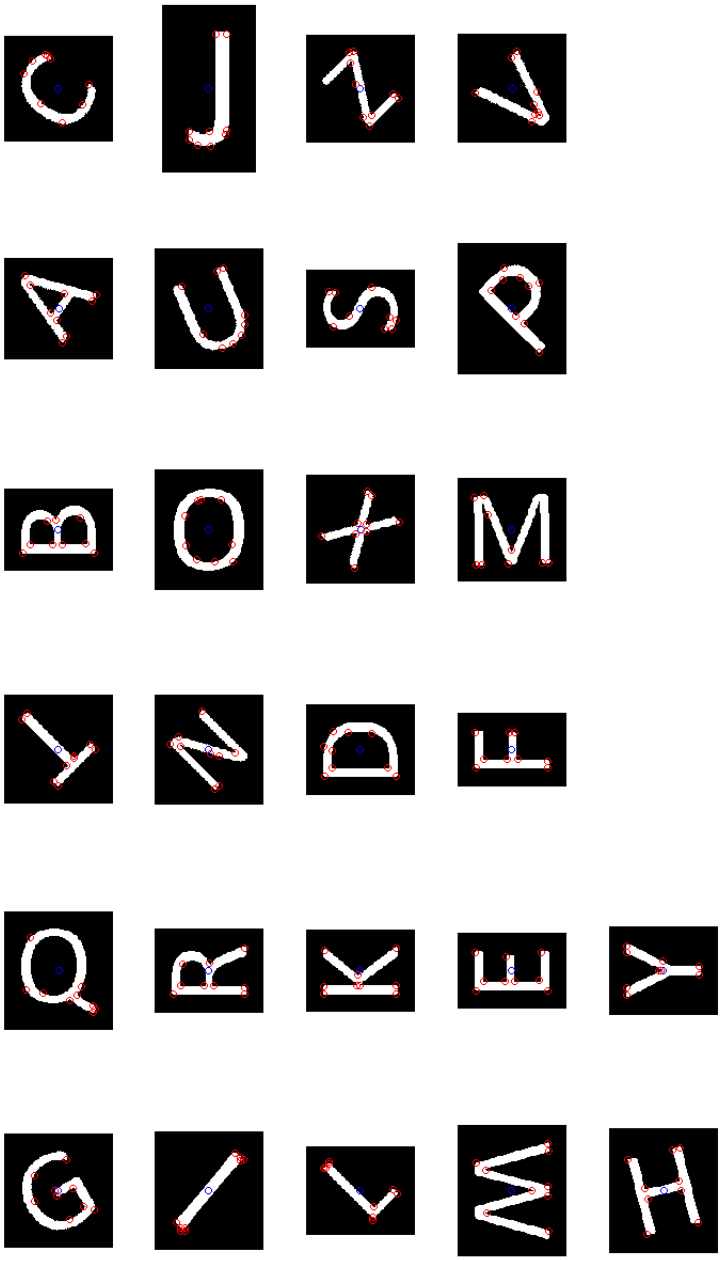


Figure 44. `Image3.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 9$

6.8 Image4.jpg

Figure 45. The Original Image4.jpg

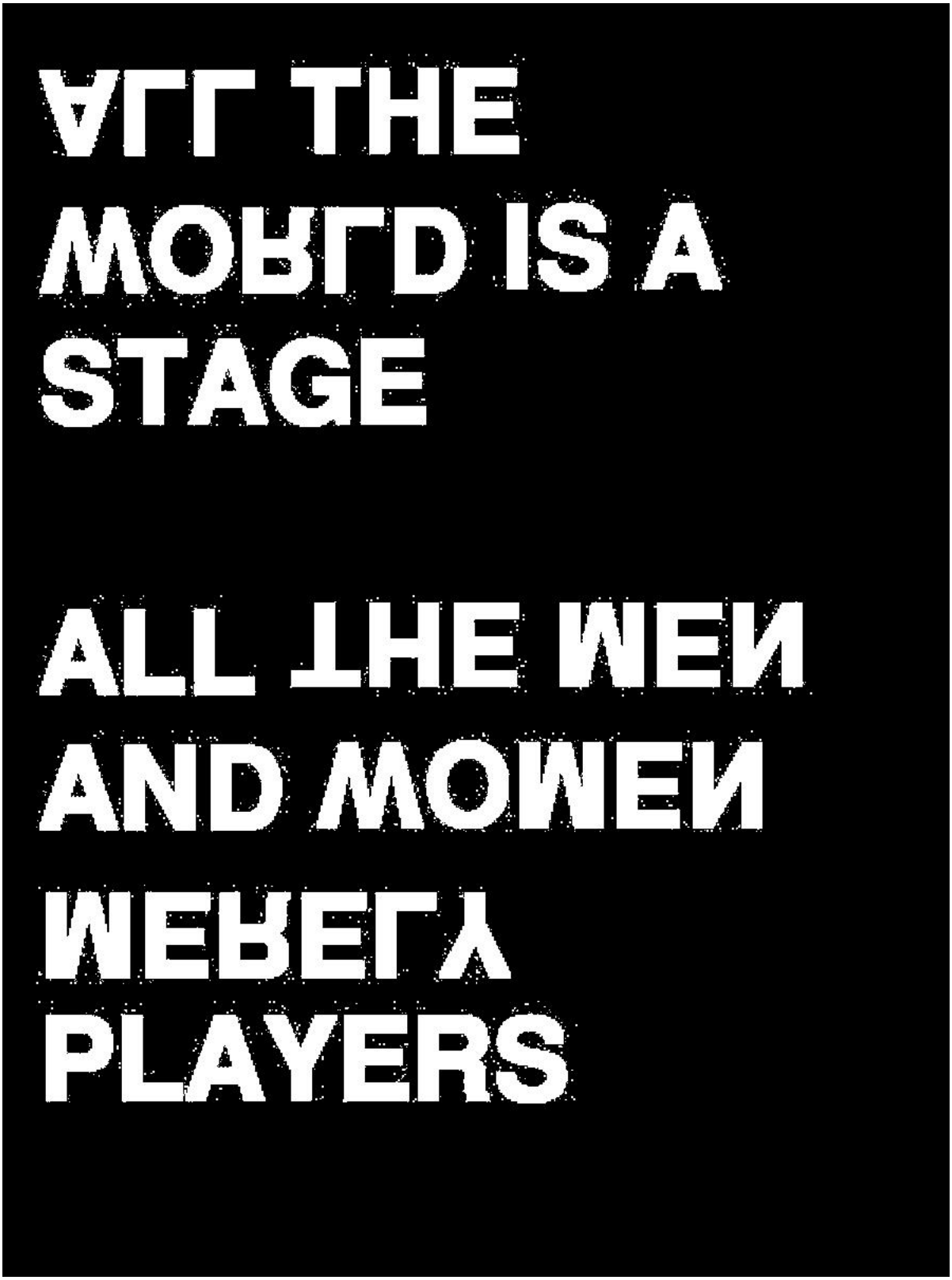


Figure 46. First Iteration of Image Segmentation of `Image0.jpg` Based On Otsu's Threshold.

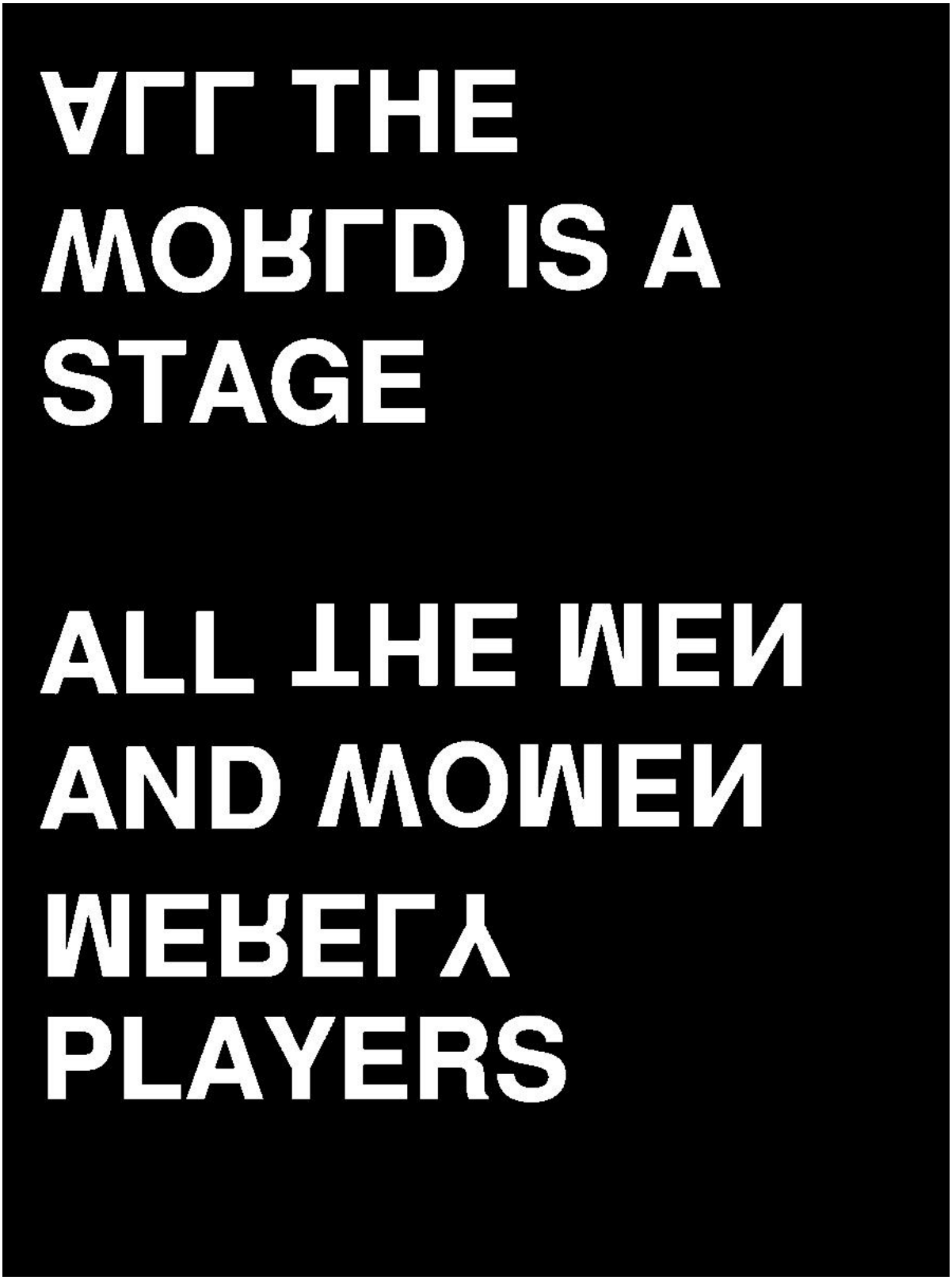


Figure 47. Second Iteration of Image Segmentation of `Image0.jpg` Based On Otsu's Threshold. (result improved)

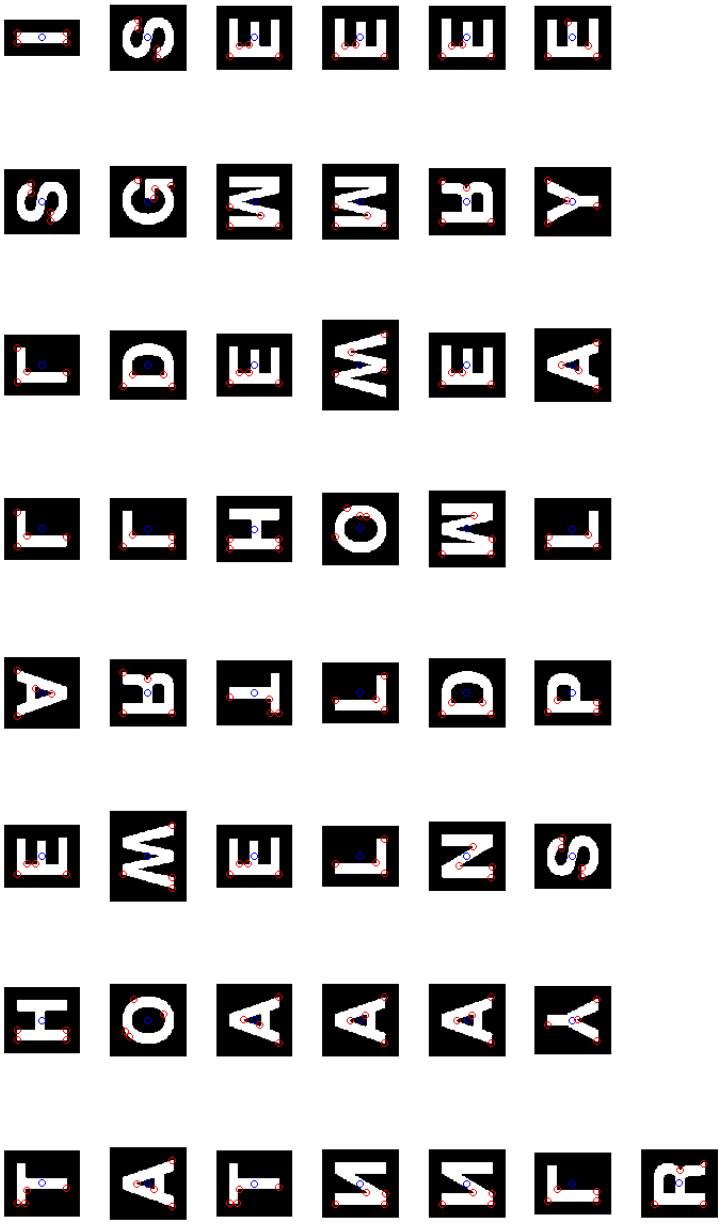


Figure 48. `Image4.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

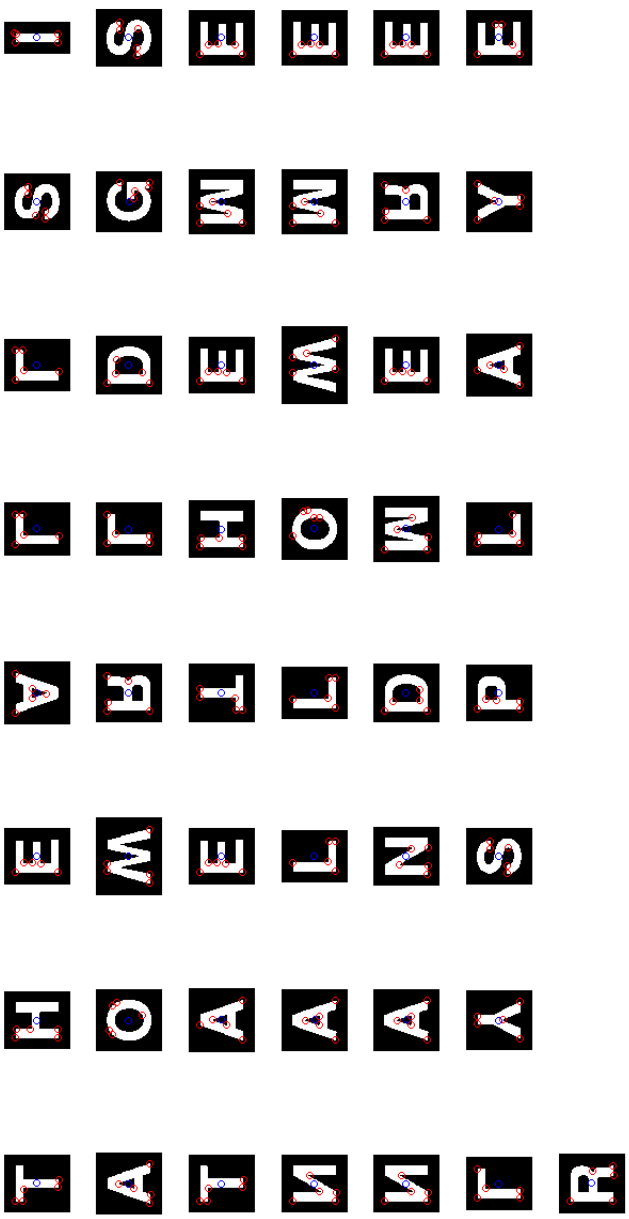


Figure 49. `Image4.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

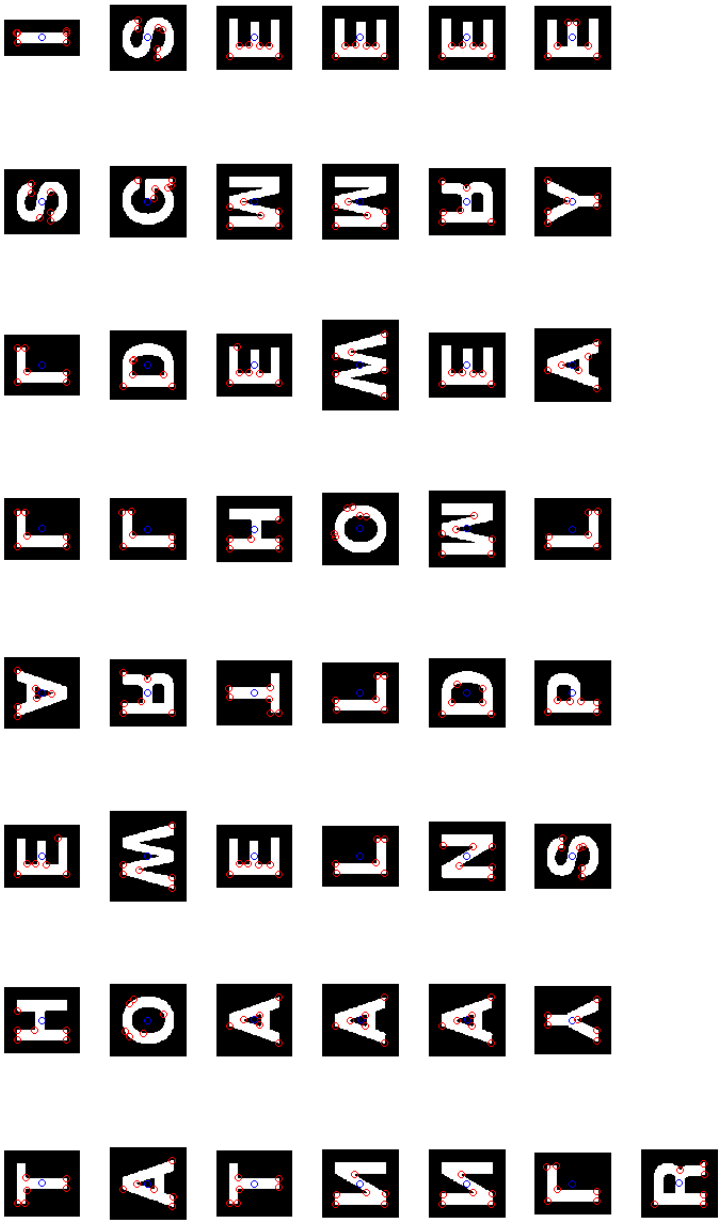


Figure 50. `Image4.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

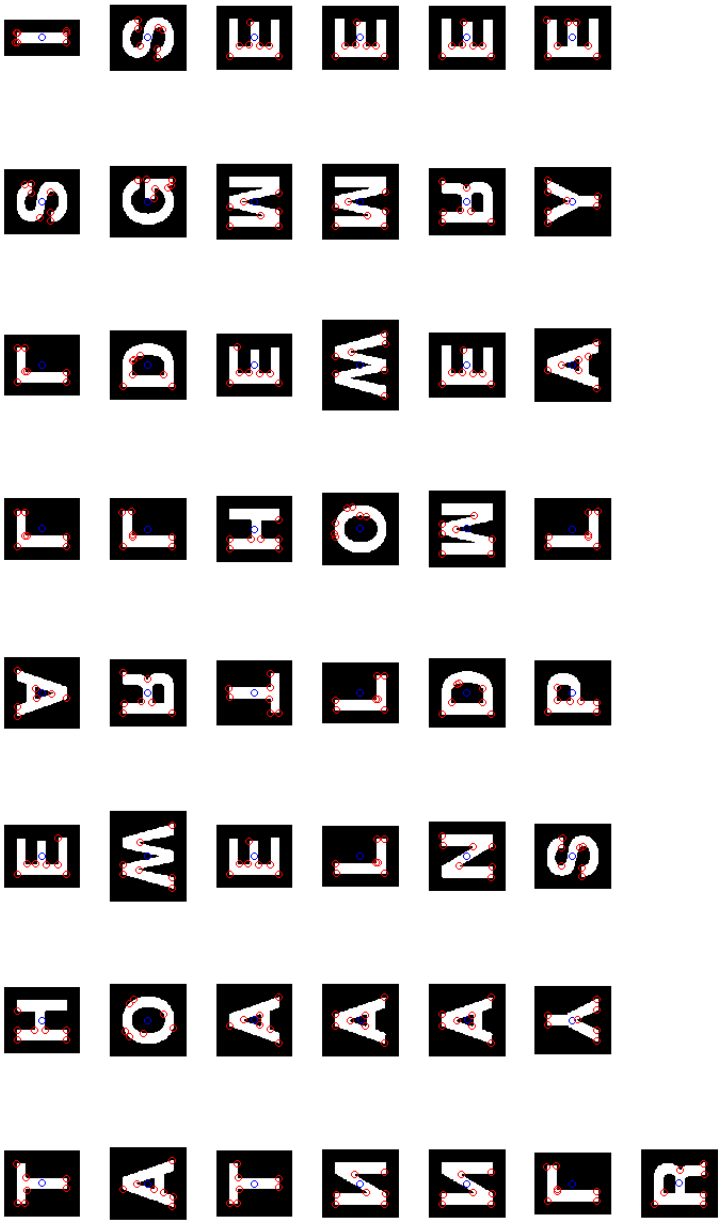


Figure 51. `Image4.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

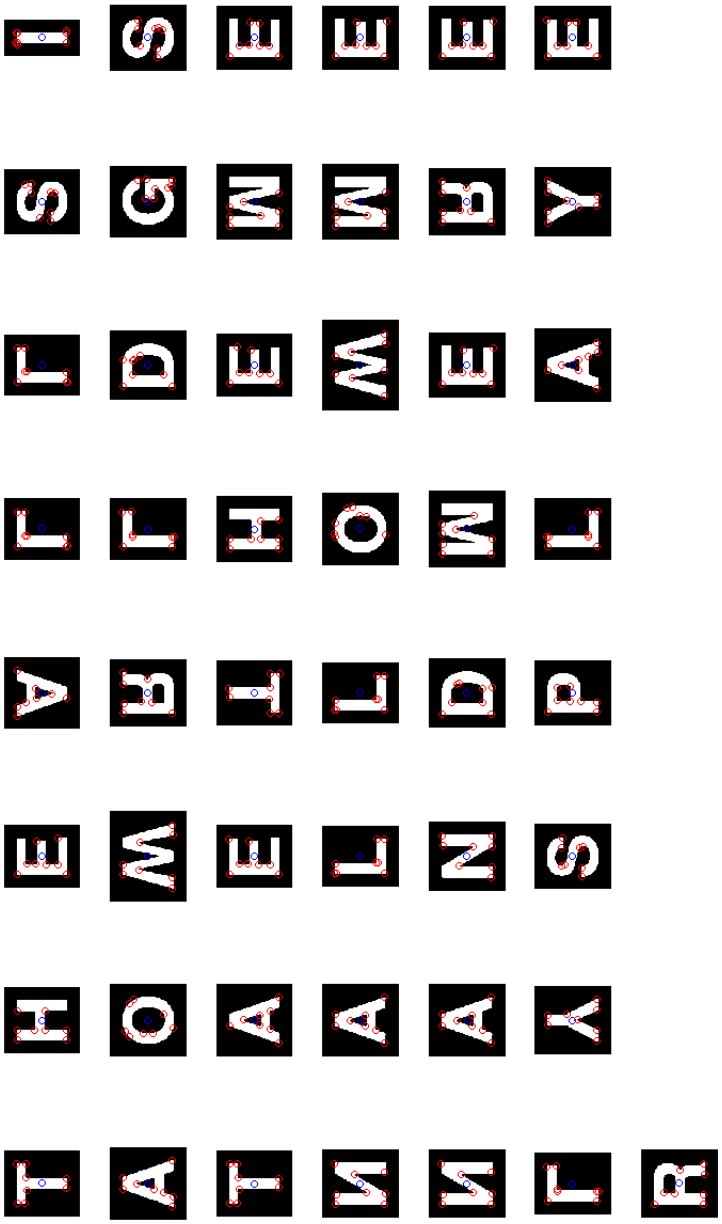


Figure 52. `Image4.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 8$

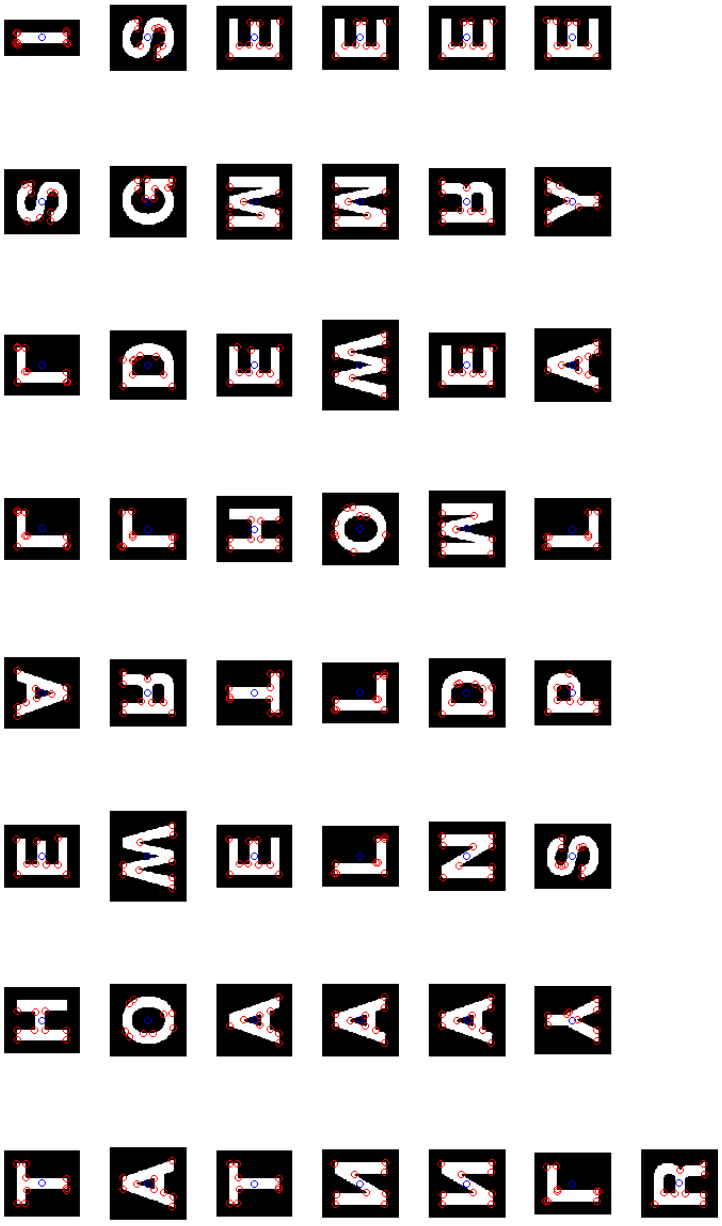


Figure 53. `Image4.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 9$

6.9 `Image5.jpg`: With Pre-Processing

Figure 54. The Original `Image5.jpg`

Figure 55. The processed `Image5.jpg`. If we do not perform image enhancement (increase contrast), we will miss several characters due to their color to close to white

Figure 55-A. The processed `Image5.jpg`. Image5.jpg over enhanced (will be problematic in segmentation).

Figure 56. First Iteration of Image Segmentation of `Image1.jpg` Based On Otsu's Threshold.

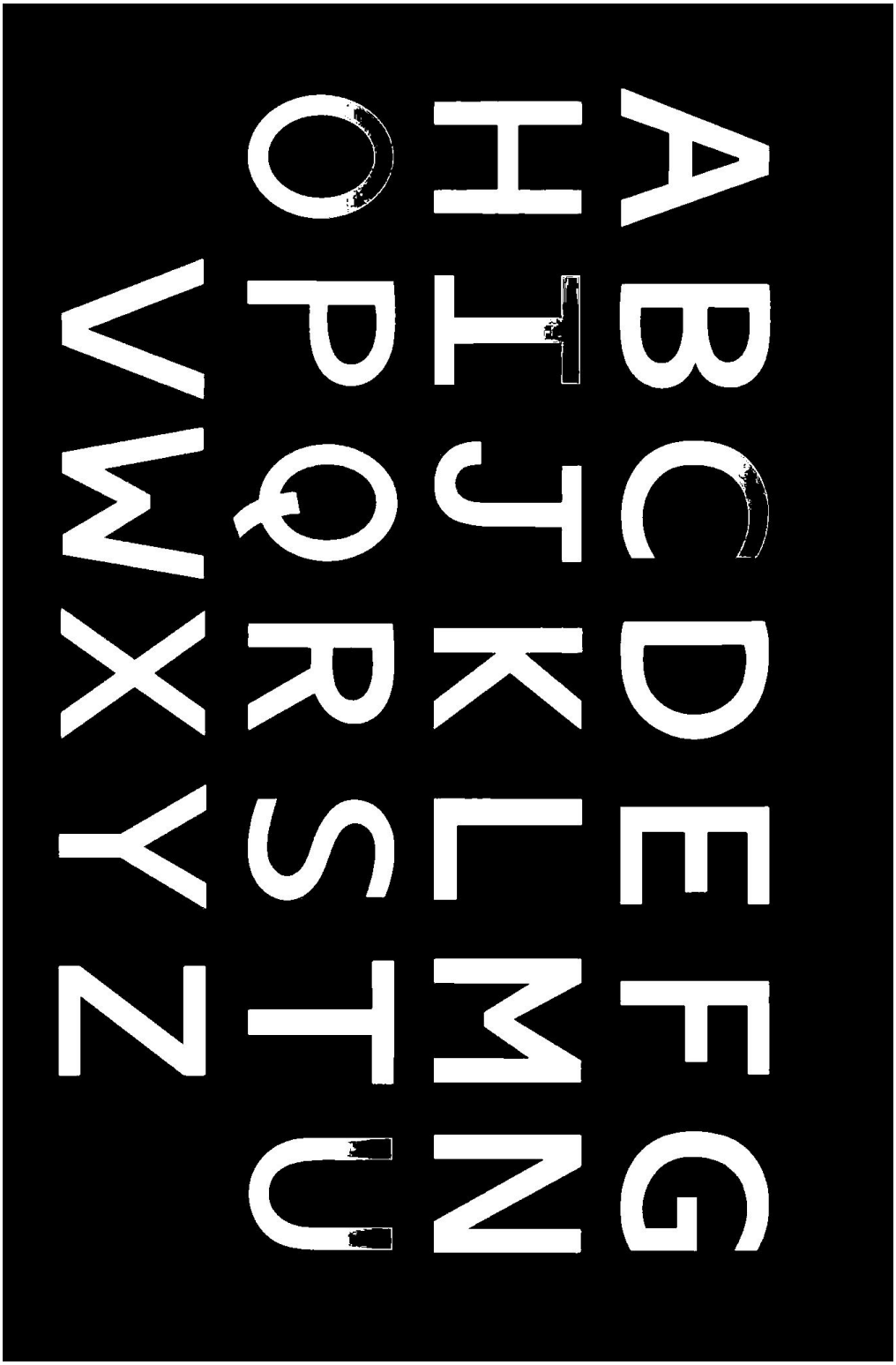


Figure 57. Second Iteration of Image Segmentation of **enhanced** `Image5.jpg` Based On Otsu's Threshold. We can see that several characters are missing parts because their colors are too close to white.

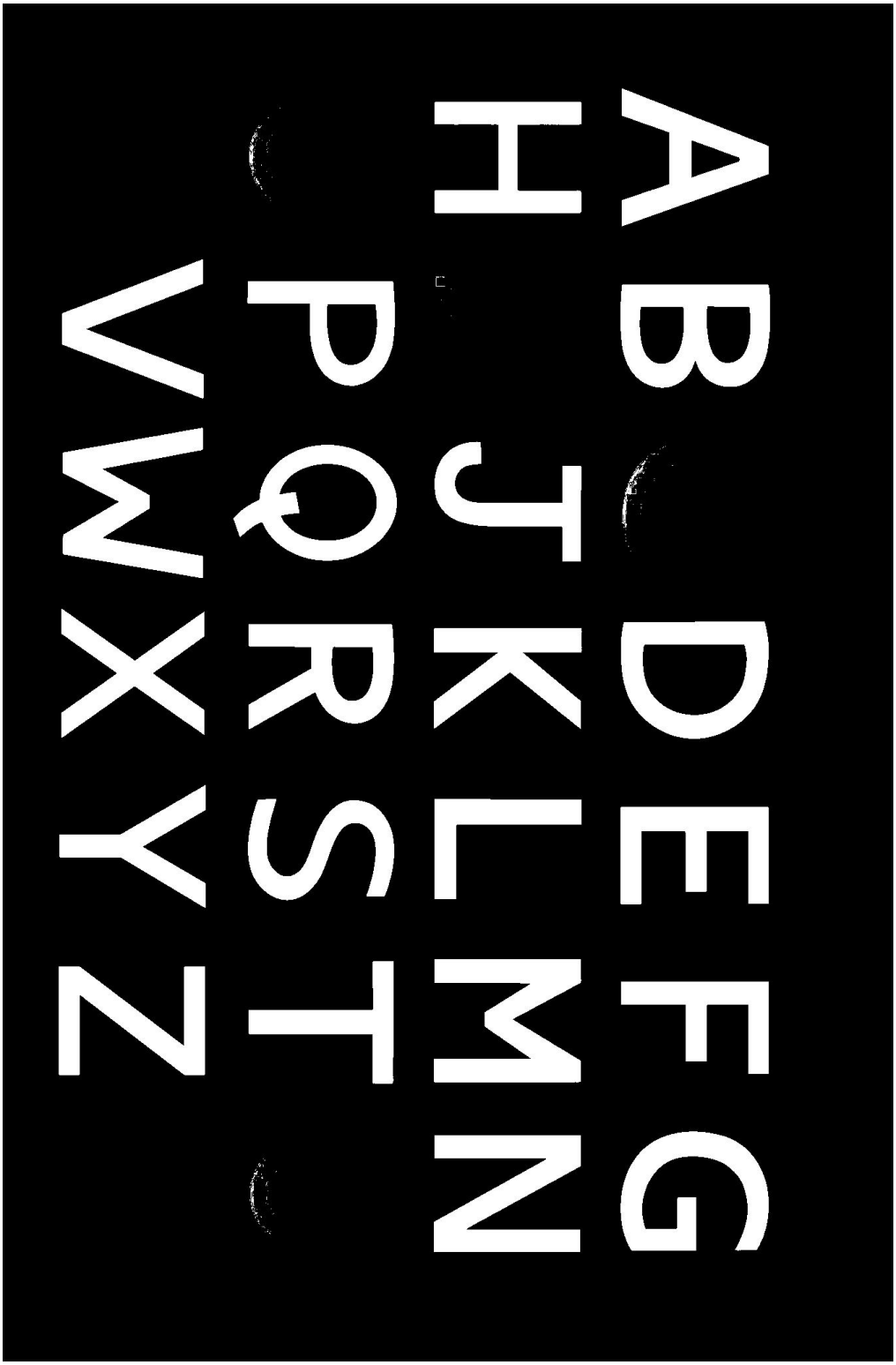


Figure 57 - A. Second Iteration of Image Segmentation of **original** `Image5.jpg` Based On Otsu's Threshold. We can see that several characters are missing are missing entirely.

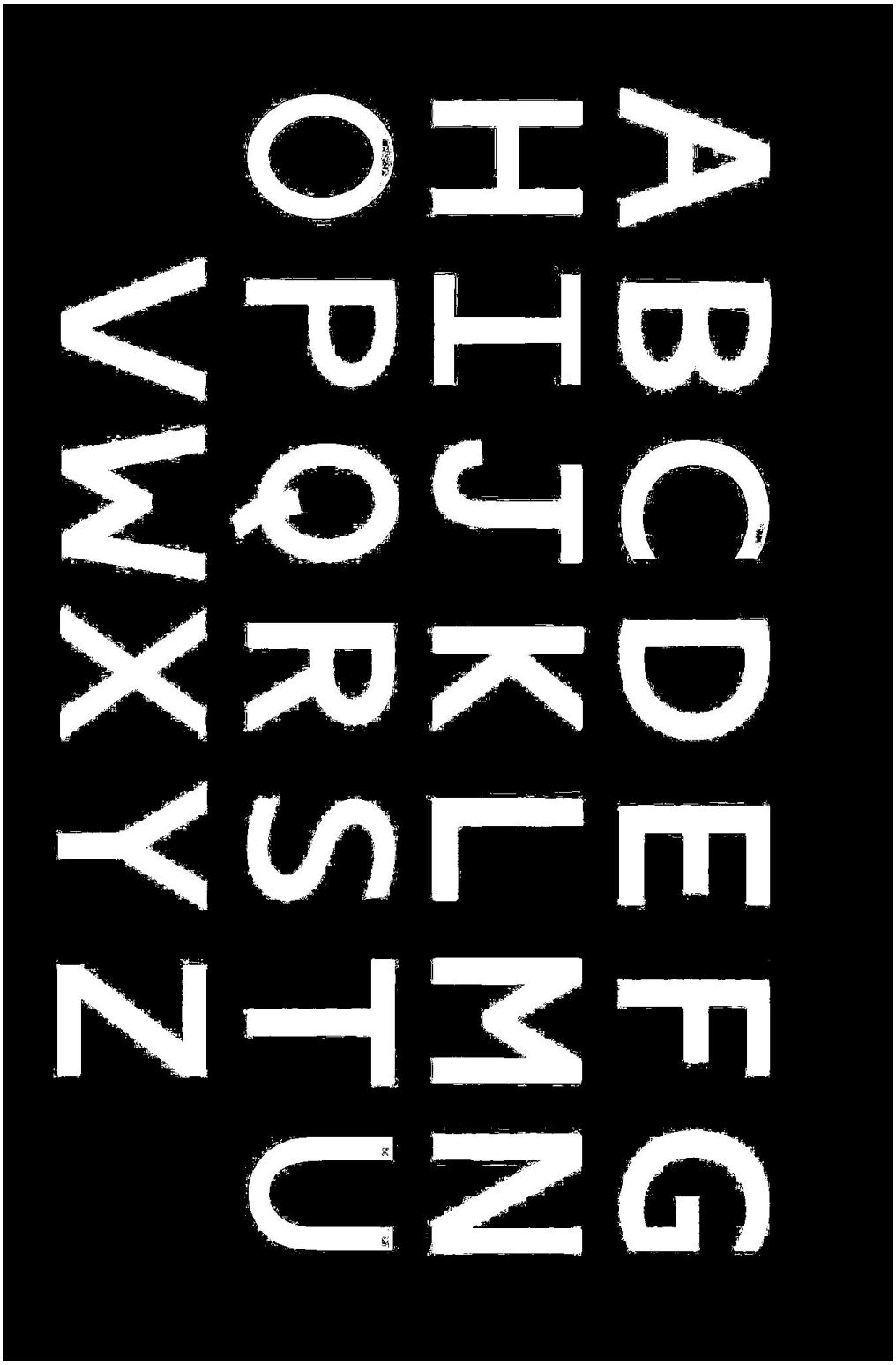


Figure 57 - B. Second Iteration of Image Segmentation of **over-enhanced** `Image5.jpg` Based On Otsu's Threshold. If the image is over-enhanced (contrast too high), we will catch a lot of noise along the edge. This will be problematic in Harris Corner Detection.

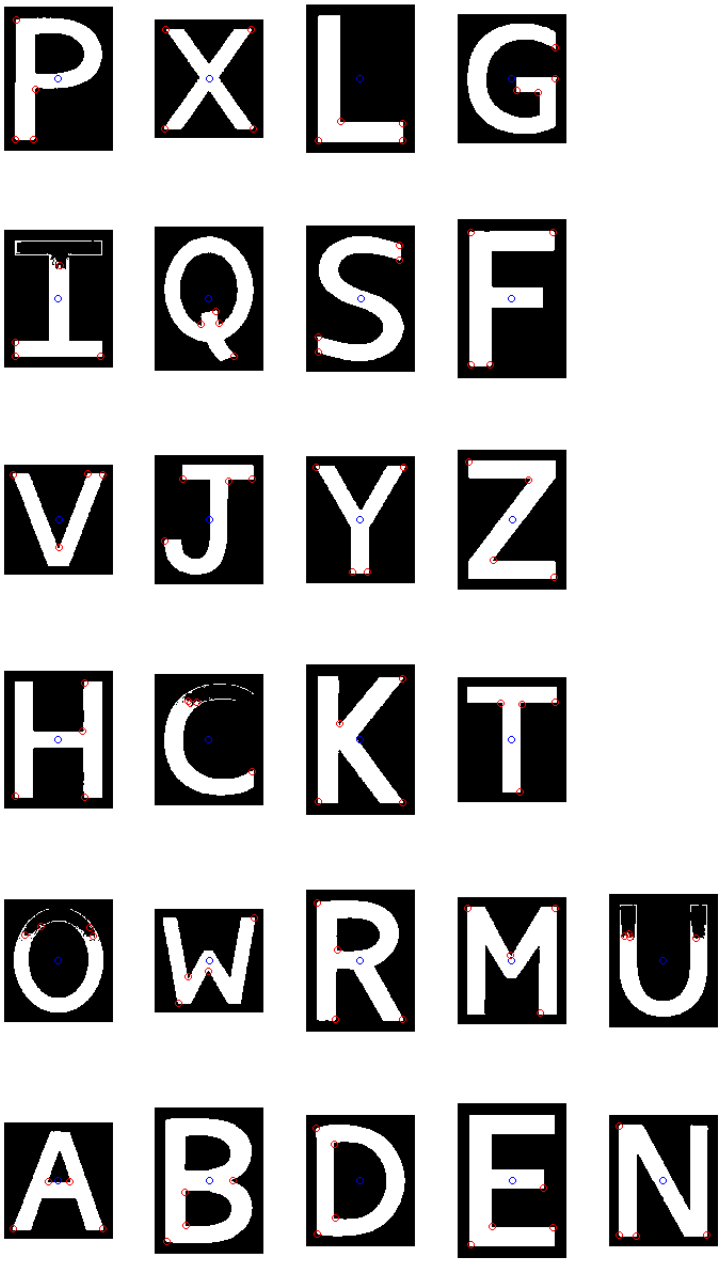


Figure 58. `Image5.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 4$

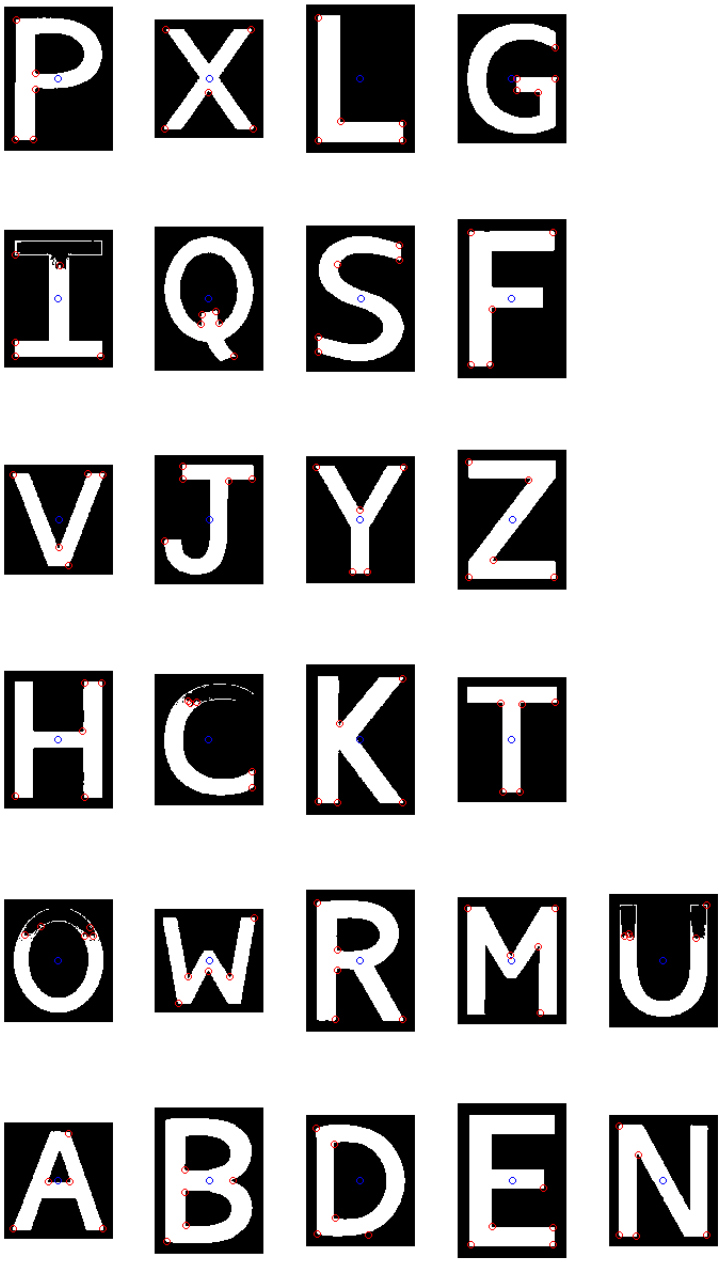


Figure 59. `Image5.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

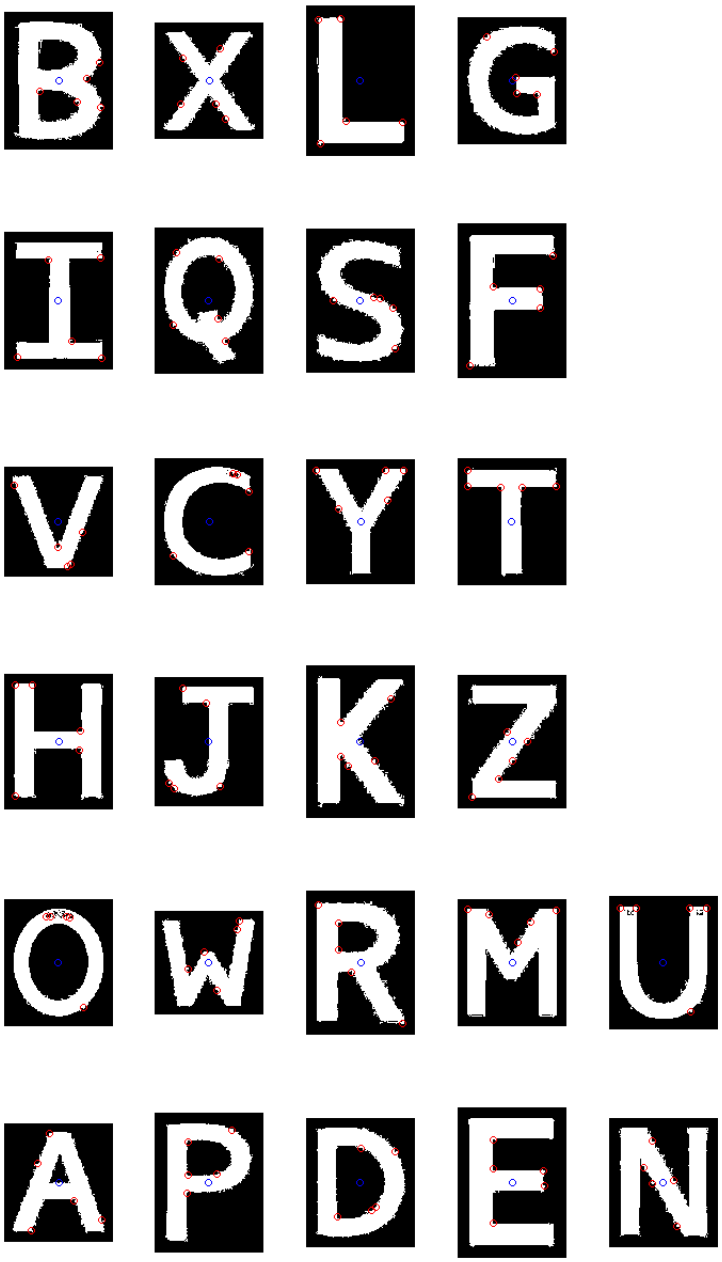


Figure 59-A. Catastrophic Result: `Over Enhanced Image5.jpg` Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 5$

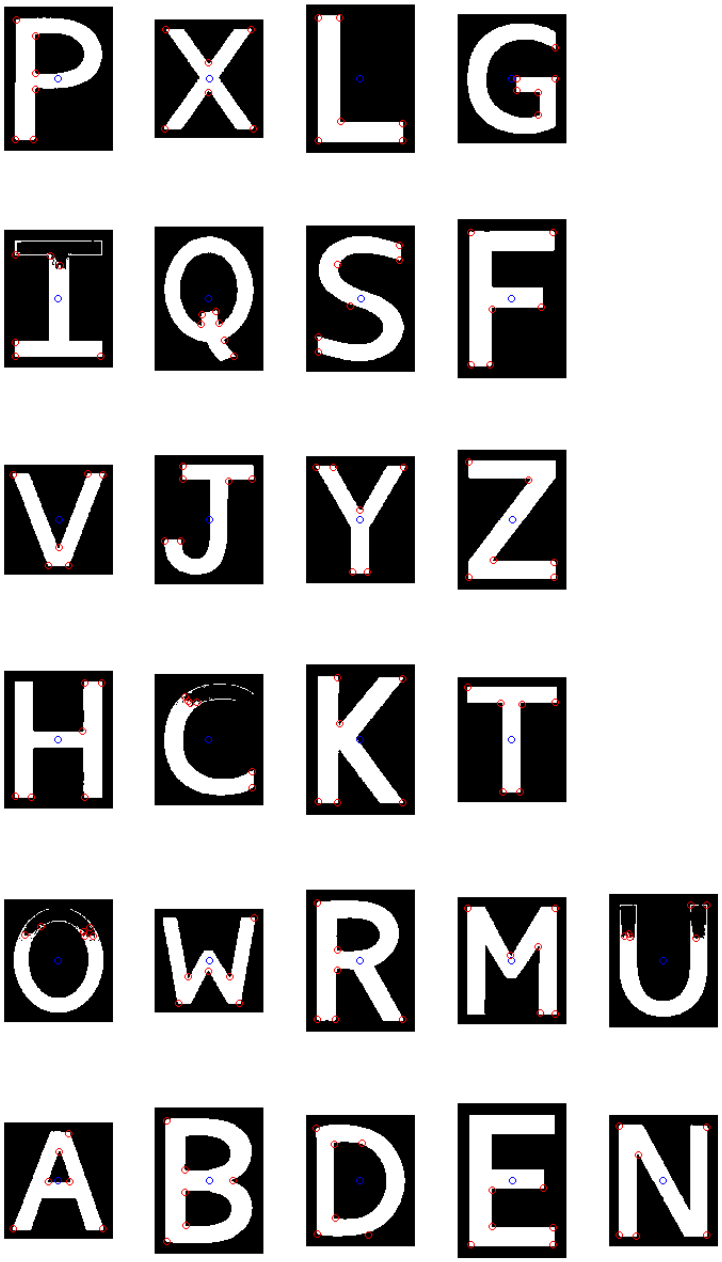


Figure 60. `Image5.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 6$

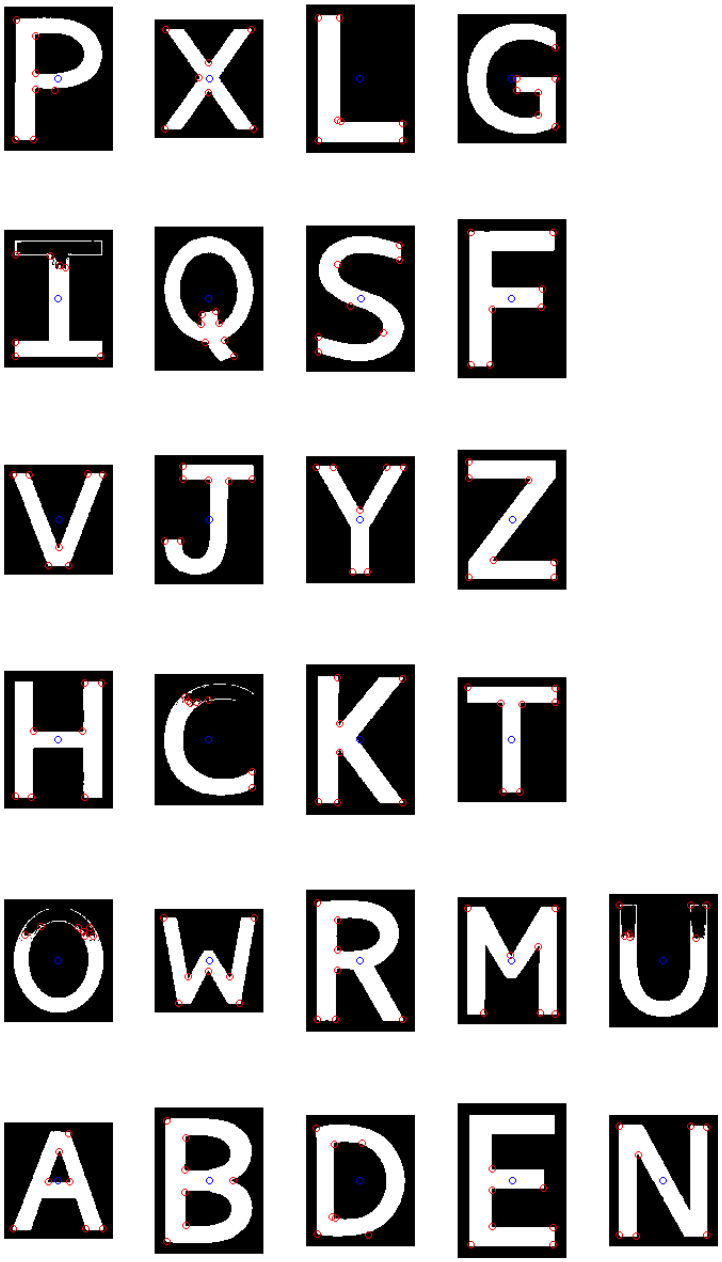


Figure 61. `Image5.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 7$

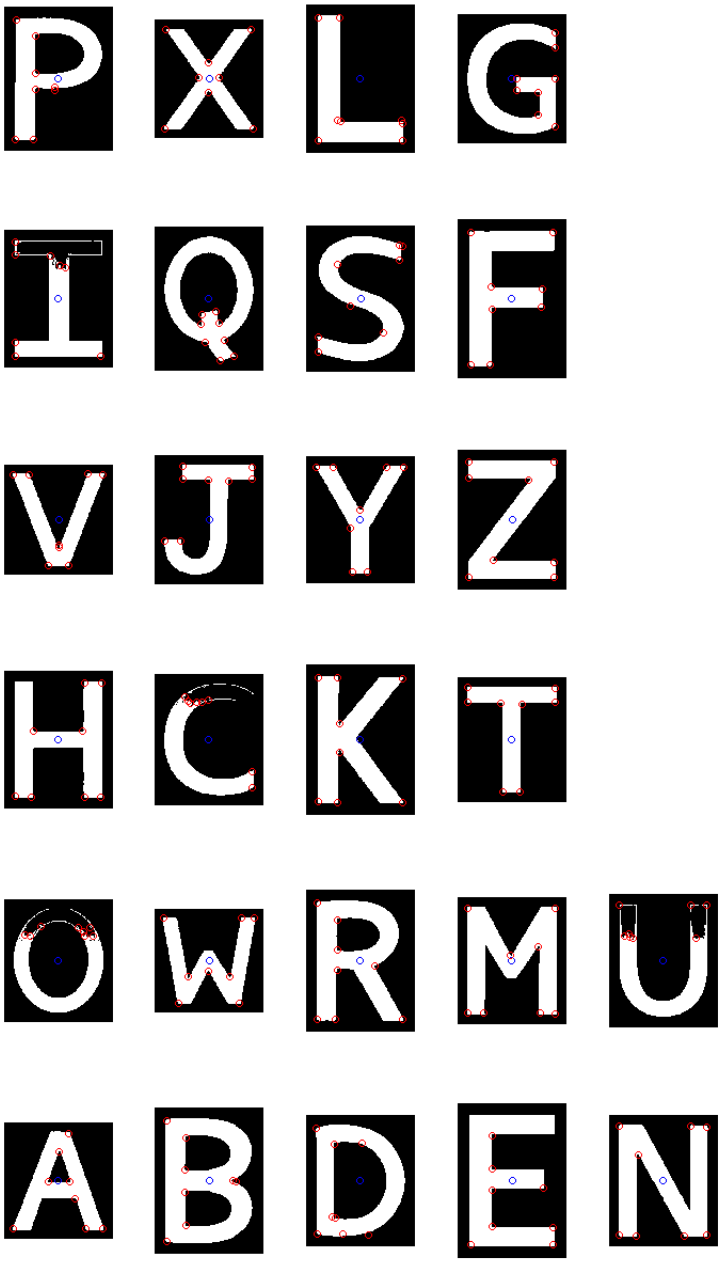


Figure 62. `Image5.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 8$

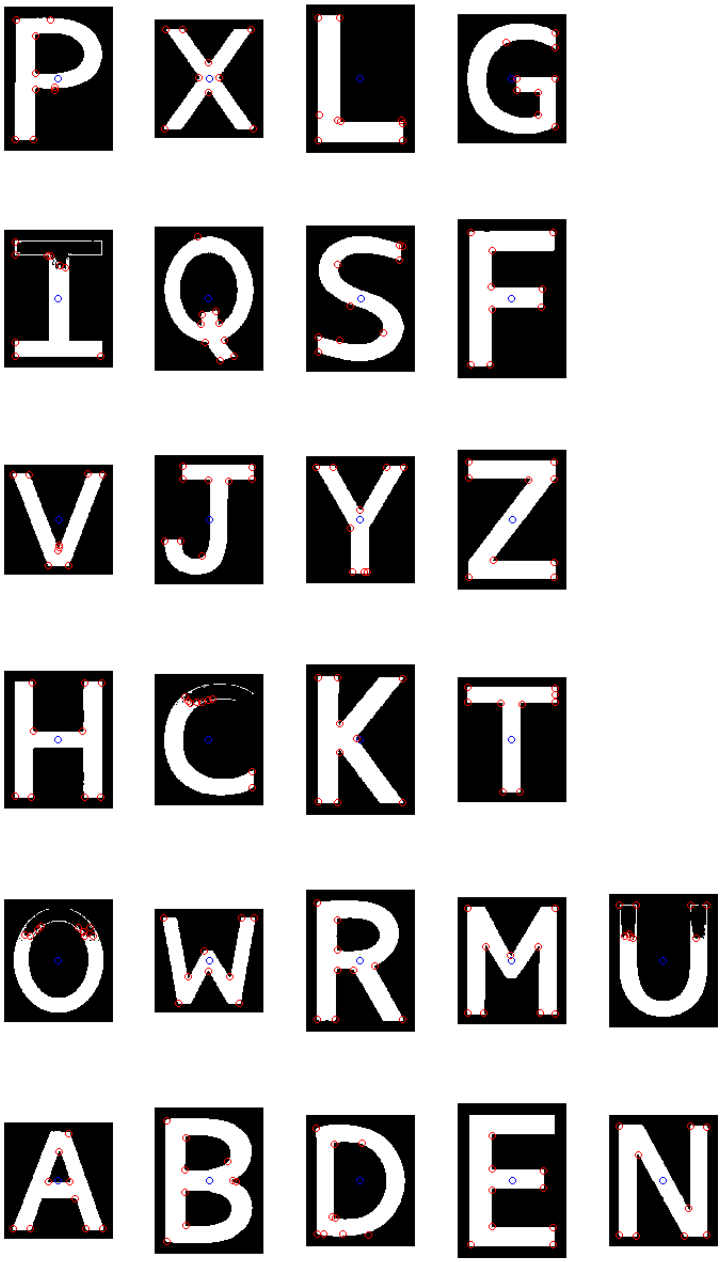


Figure 63. `Image5.jpg`: Segmented Letters With Harris Corners (red circle) Detected and Center Estimated (blue circle). $N = 9$

6.10 `Image6.jpg`: With Pre-Processing

Generally, the image segmentation failed on `Image6.jpg`. Thus letter was not extracted accurately and hence very little can be done.

Figure 64. Original `Image6.jpg`

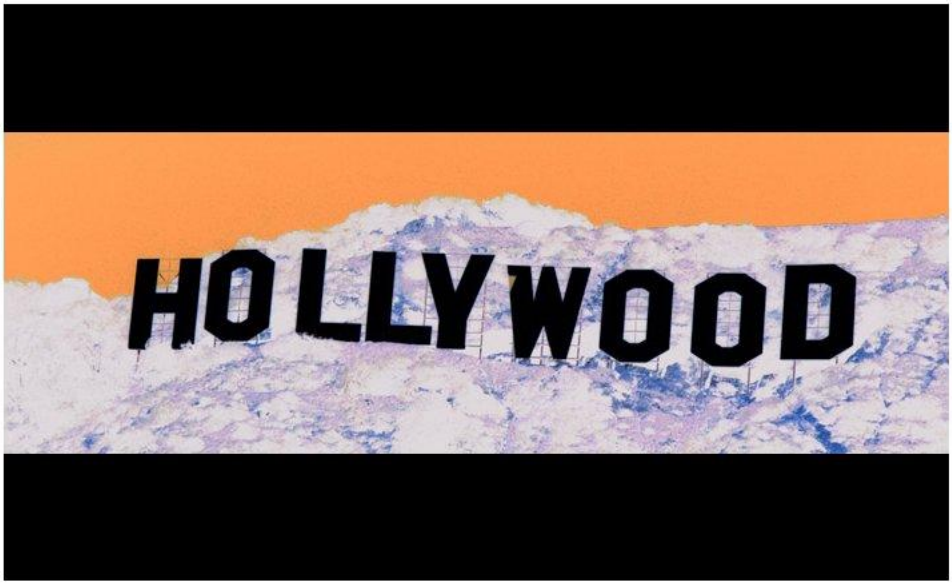


Figure 65. Processed `Image6.jpg`

Figure 66. First Iteration of Image Segmentation of processed `Image6.jpg` Based On Otsu's Threshold.

Figure 66. Second Iteration of Image Segmentation of processed `Image6.jpg` Based On Otsu's Threshold.

Figure 67. Third Iteration of Image Segmentation of processed `Image6.jpg` Based On Otsu's Threshold.

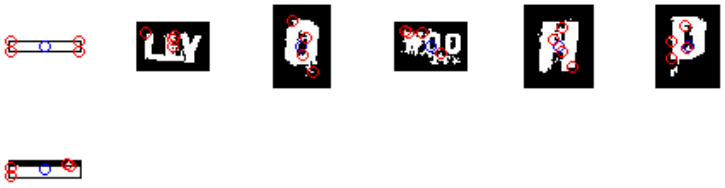


Figure 68. Segmentation results were catastrophic. So was the accuracy of Harris Corner Detector.

7 Appendix: Matlab Code

7.1 main.m

```
clear all; close all; clc;

N = input('Please pick a number N (N harris corners as interest points): ');
test_img_name = input('Please Slected the image you want to test (TestImg.jpg)
    : ');
%% Load And Process Training Images

disp('                  Starting Training Process...')
img_training = imread('Training.jpg');
img_size = size(img_training);
I_image = ones(img_size(1),img_size(2));

% figure
% imshow(img_training);  % Display the orginal image
img_gray = 0.2125*img_training(:,:,1) + 0.7154*img_training(:,:,2) + 0.0721*
    img_training(:,:,3);
img_gray = rgb2gray(img_training);


%% First Iteration of Training Image Segmentation: Try to extract the
    chracters
disp('Performing first iteration of image segmentation based on Otsu threshold
    ...')

I_image = image_segmentation( I_image, img_gray );

% figure
% imshow(I_image*255) %%% The result have some noise in it. Hence perform
    second iteration
imwrite(uint8(I_image*255),'training_first_iteration.jpg','jpeg')
% truesize

%% Second Iteration of Training Image Segmentation: Try to extract the
    chracters
disp('Performing second iteration of image segmentation based on Otsu
    threshold...')

I_image = image_segmentation( I_image, img_gray );

% figure
% imshow(I_image*255)
% truesize
imwrite(uint8(I_image*255),'training_second_iteration.jpg','jpeg')
%After second iteration, the result looks better.

%% Extract and process the segmented image

segmented_image = I_image;
```

```
42
43 disp('Extracting letters separately based on 8 neighberihood... ')
44 % Character Segmentation and Component Labelling Using Connected Component
45 chars_lables = connected_component(segmented_image);
46 % Extract Individual Letters
47 [save_img, corner_locs] = extract_letter(chars_lables, N);
48 disp('Formulating training feature vectors...')
49 % Project the Harris Corners and Form Feature Vector
50 [angles, arc_corners] = project_corners( save_img, corner_locs, N );
51
52 % test_arc_corners = arc_corners;
53 % test_img = save_img;
54
55
56
57 %% Training Data Obtained, Training Process End
58 %% Testing Process start
59
60 %% Load And Process Training Images
61 disp('                  Start Classifying Process...')
62
63 img_testing = imread(test_img_name);
64 img_size = size(img_testing);
65 I_image = ones(img_size(1),img_size(2));
66
67 % figure
68 % imshow(img_testing);  % Display the orginal image
69 img_gray = 0.2125*img_testing(:,:,1) + 0.7154*img_testing(:,:,2) + 0.0721*
       img_testing(:,:,3);
70 img_gray = rgb2gray(img_testing);
71
72 %% First Iteration of Training Image Segmentation: Try to extract the
       chracters
73 disp('Performing first iteration of image segmentation based on Otsu threshold
       ...')
74
75 I_image = image_segmentation( I_image, img_gray );
76
77 save_name = ['first_iteartion_test_', test_img_name];
78 % figure
79 % imshow(I_image*255) %%% The result have some noise in it. Hence perform
       second iteration
80 imwrite(uint8(I_image*255),save_name,'jpeg')
81 % truesize
82
83 %% Second Iteration of Training Image Segmentation: Try to extract the
       chracters
84 disp('Performing second iteration of image segmentation based on Otsu
       threshold...')
85
86 I_image = image_segmentation( I_image, img_gray );
87 save_name = ['second_iteartion_test_', test_img_name];
88 % figure
89 % imshow(I_image*255)
```

```
90  % truesize
91  imwrite(uint8(I_image*255), save_name, 'jpeg')
92  %After second iteration, the result looks better.
93
94  %% For image6 only process
95  % %% Third Iteration of Training Image Segmentation: Try to extract the
        chracters
96  % disp('Performing third iteration of image segmentation based on Otsu
        threshold...')
97  %
98  % I_image = image_segmentation( I_image, img_gray );
99  % save_name = ['third_iteartion_test_', test_img_name];
100 % % figure
101 % % imshow(I_image*255)
102 % % truesize
103 % imwrite(uint8(I_image*255), save_name, 'jpeg')
104 % %After second iteration, the result looks better.
105
106 %% Extract and process the segmented image
107
108 segmented_image = I_image;
109
110 disp('Extracting letters separately based on 8 neighberihood... ')
111 % Character Segmentation and Component Labelling Using Connected Component
112 chars_lables = connected_component(segmented_image);
113 % Extract Individual Letters
114 [test_img, corner_locs] = extract_letter(chars_lables, N);
115 disp('Formulating testing feature vectors...')
116 % Project the Harris Corners and Form Feature Vector
117 [test_angles, test_arc_corners] = project_corners( test_img, corner_locs, N );
118
119 disp('Matching the result based on feature vectors...')
120 % Classification Based on Euclidean Distance
121 classify_matrix = euc_matching( test_arc_corners, test_img, arc_corners,
        save_img, N);
```

7.2 Otsu.m

```
function [ Tk ] = Otsu(I_img, img )
%% Otsu's algorithm to detect threshold value for image segmentation

% Disregard those points at which the indicator value is '0'
% (As our approach is iterative based thus this is important)
img_size = size(img);
imgmin = img;
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
        if (img(i,j) == 0)
            imgmin(i,j) = 255;
            I_img(i,j) = 0;
        else
        end
    end
end

% Find the range of pixel values
img_min = min(min(imgmin));
img_max = max(max(img));

% Find the mu_T value
mut = sum(sum(img))/sum(sum(I_img));

% Find the threshold value
cnt = 1;
for k = img_min:1:img_max
    p(cnt) = sum(sum(img == k))/sum(sum(I_img));
    cnt = cnt + 1;
end

cnt = 1;
for k = img_min:1:img_max
    p_sum(cnt) = sum(p(1:cnt));

    m0(cnt) = 0;
    m1(cnt) = 0;

    cnt_i = 1;
    for i = img_min:1:k
        m0(cnt) = m0(cnt) + i*p(cnt_i);
        cnt_i = cnt_i + 1;
    end

    for i = k+1:1:img_max-1
        m1(cnt) = m1(cnt) + i*p(cnt_i);
        cnt_i = cnt_i + 1;
    end

    mu0(cnt) = m0(cnt)/p_sum(cnt);
```

```
51      mu1(cnt) = (m1(cnt))/(1-p_sum(cnt));
52      sigma_b(cnt) = (p_sum(cnt)*(1-p_sum(cnt)))*((mu0(cnt) - mu1(cnt))^2);
53
54      cnt = cnt + 1;
55  end
56
57  size_sigma = size(sigma_b);
58
59  % Return the threshold value
60  for k = 1:1:size_sigma(2)
61      if (sigma_b(k) == max(sigma_b))
62          Tk = img_min + k;
63      else
64      end
65  end
66
67
68  end
```

7.3 image_segmentation.m

```
1   function [ I_image ] = image_segmentation( I_image, img_gray )
2   %% Image Segmentation: This is Image Segmentation Based on Otsu's Algorithm
3   %  This step can not extract individual characters
4
5   T = Otsu(I_image, img_gray);
6   img_size = size(I_image);
7
8   for i = 1:1:img_size(1)
9       for j = 1:1:img_size(2)
10          if img_gray(i,j) < T
11              I_image(i,j) = 1;  % If lower then threshold, asisgn '1'
12          else
13              I_image(i,j) = 0;  % Otherwise, assign '0'
14          end
15      end
16  end
17
18  end
```

7.4 connected_component.m

```
function [ chars_lable ] = connected_component( segmented_image )

%% Extract the letters separately base don segmentation result from previous
    step
segment_counter = 1;
img_size = size(segmented_image);


for i = 2:1:img_size(1)-1
    for j = 2:1:img_size(2)-1
        % Whenever we detect a pixel location that is in our interest
        % region, segmentation started

        if segmented_image(i,j) == 1
            tmp_img = zeros(img_size(1),img_size(2));
            cnt = 1;
            cnt_follow = 0;
            x = i;
            y = j;
            % Perform segmentation using 8 components connected neighbours
            for m = -1:1:1
                for n = -1:1:1
                    if(segmented_image(x+m,y+n) == 1)
                        tmp_img(x+m,y+n) = 1;
                        segmented_image(x+m,y+n) = 0;
                        loc_vec(cnt, 1) = x + m;
                        loc_vec(cnt, 2) = y + n;
                        cnt = cnt + 1 ;
                    else
                    end
                end
            end
            % Keep pushing the connected component to the set: C
            while (cnt_follow < (cnt-1))
                cnt_follow = cnt_follow + 1;
                x = loc_vec(cnt_follow, 1);
                y = loc_vec(cnt_follow, 2);
                for m = -1:1:1
                    for n = -1:1:1
                        if(segmented_image(x+m,y+n) == 1)
                            tmp_img(x+m,y+n) = 1;
                            segmented_image(x+m,y+n) = 0;
                            loc_vec(cnt, 1) = x + m;
                            loc_vec(cnt, 2) = y + n;
                            cnt = cnt + 1 ;
                        else
                        end
                    end
                end
            end
```

```
50
51      %% Extract each of those letters
52              if (sum(sum(tmp_img)) > 750) % Pixels included in a single region
                    need to be larger than a certain number
53                  chars_lable(:,:,segment_counter) = tmp_img;
54                  segment_counter = segment_counter + 1;
55              else
56              end
57
58          else
59          end
60      end
61  end
62
63  end
```

7.5 extract_letter.m

```
function [ save_img, corner_locs ] = extract_letter( chars_lables,N )

%% Extract the small pixel blob bar around each distinct letter
% After This step, each letter would be in a individual pixel-block


size_chars_lables = size(chars_lables);

for k = 1:1:size_chars_lables(3)
    % Find the block around a certain tagged area (individual letter)
    char_letter(:,:) = chars_lables(:,:,k);
    max_x = 1;
    min_x = size_chars_lables(1);
    max_y = 1;
    min_y = size_chars_lables(2);

    for i = 1:1:size_chars_lables(1)
        for j = 1:1:size_chars_lables(2)
            if (char_letter(i,j) ~= 0)
                if (i > max_x)
                    max_x = i;
                else
                end

                if (j > max_y)
                    max_y = j;
                else
                end

                if (i < min_x)
                    min_x = i;
                else
                end

                if (j < min_y)
                    min_y = j;
                else
                end
            end
        end
    end

    tmpimg= chars_lables(min_x:max_x,min_y:max_y,k);
    save_img{k} = zeros(max_x-min_x+41,max_y-min_y+41);
    save_img{k}(21:max_x-min_x+21,21:max_y-min_y+21) = tmpimg;

    clear min_x max_x min_y max_y;
    % After each of the individual letter was extracted, go to the Harris
    % Corner Detection
    corner_locs{k} = harris_corner(save_img{k},N);
```

```
51
52 end
53
54
55 end
```

7.6 harris_corner.m

```
function [corner_loc1 ] = harris_corner( pic1,N )
%% Harris Corner Detector
% This is loosely based on my HW4. Retunr 'N' sharpest corners
pic1 = pic1.*250;
scale  = 1;
pic1_gray = double(pic1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up some coefficient, Rssd = ratio for SSD, Rncc = ratio for NCC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k = 0.04;

I1 = (pic1);
size_I1 = size(I1);
scale_flag = 0;
if ((size_I1(1)*size_I1(2)) < (160*160))
    pic1 = imresize(pic1,2);
    I1 = pic1;
    scale_flag = 1;
else
end
size_I1 = size(I1);
I1x = zeros(size_I1(1),size_I1(2));
I1 = double(I1);
haar_size = round(round((4*scale+1))/2)*2;

if (size_I1(1)*size_I1(2))

    %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Smooth the image a bit before processing to make sure those noise would
    % not be detected as corners (to improve computational efficiency)
    %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%     smooth_filter = fspecial('gaussian', 5*scale, scale);
%     I1 = imfilter(I1,smooth_filter);

    %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Applying 'Haar' Filter
    %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Hx(1:haar_size,1:haar_size/2) = -1;
    Hx(1:haar_size,haar_size/2+1:haar_size) = 1;
    Hy(1:haar_size/2,1:haar_size) = 1;
```

```
    Hy(haar_size/2+1:haar_size,1:haar_size) = -1;

    I1x = imfilter(I1,Hx);
    I1y = imfilter(I1,Hy);

    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % This part no longer useful. The part is for the initial implementation of Sobel filter
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % This part is for sobel operator
    %

%     for i = 2:1:size_I1(1)-1;
%         for j = 2:1:size_I1(2)-1;
%             I1x(i,j) = I1(i-1,j+1) + 2*I1(i, j+1) + I1(i+1, j+1) -I1(i-1,j-1) ...
%                 - 2*I1(i,j-1) - I1(i+1,j-1);
%             I1y(i,j) = I1(i-1,j-1) + 2*I1(i-1,j) + I1(i-1,j+1)  - I1(i+1,j-1) ...
%                 -2*I1(i+1, j) - I1(i+1, j+1);
%         end
%     end

    %
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Plot the gradient as intermediate result to make sure Haar filter was correctly implemented
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     figure
%     subplot(2,1,1)
%     image(I1x)
%     colormap(gray(256))
%     subplot(2,1,2)
%     image(I1y)
%     colormap(gray(256))
%     truesize



```

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the C matrix for image 1
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =21:1:size_I1(1)-20
    for j = 21:1:size_I1(2)-20
        C_Matrix_I1 = [0,0;0,0];
        for m = -(5*scale-1)/2:1:(5*scale-1)/2
            for n = -(5*scale-1)/2:1:(5*scale-1)/2
                if (i+m>0) && (i+m < size_I1(1)) && (j+n>0) && (j+n < size_I1(2))
                    C_Matrix_I1(1,1) = C_Matrix_I1(1,1) + I1x(i+m,j+n)*I1x(i+m,j+n);
                    C_Matrix_I1(2,2) = C_Matrix_I1(2,2) + I1y(i+m,j+n)*I1y(i+m,j+n);
                    C_Matrix_I1(1,2) = C_Matrix_I1(1,2) + I1x(i+m,j+n)*I1y(i+m,j+n);
                    C_Matrix_I1(2,1) = C_Matrix_I1(2,1) + I1x(i+m,j+n)*I1y(i+m,j+n);
                else
                end
            end
        end
        C_I1{i,j} = C_Matrix_I1;
    end
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check the rank of C matrix. If rank not euqal to 2 then dump the points
% If rank(C) = 2 then save the points for further processing
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =21:1:size_I1(1)-20
    for j = 21:1:size_I1(2)-20
        C_Matrix_I1 = C_I1{i,j};
        if (rank(C_Matrix_I1) == 2)
            I_corner_I1(i,j) = 1;
        else
            I_corner_I1(i,j) = 0;
        end
    end
end


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % Estimate the corner strength at the remaining cadidate locations for image 1
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Corner_strength_H_I1 = zeros(size_I1(1),size_I1(2));
    for i =21:1:size_I1(1)-20
        for j = 21:1:size_I1(2)-20
            if (I_corner_I1(i,j) == 1)
                %[U,S,V] = svd(C_I1{i,j});   %%%No need to use SVD
                %Corner_strength_I1(i,j) = S(1,1)*S(2,2) - k*(S(1,1)+S(2,2))^2; %%%No need to use SVD
                Corner_strength_H_I1(i,j) = det(C_I1{i,j}) - k*(trace(C_I1{i,j}))^2; %%%This is better way to calculate corner strength
            end
        end
    end



    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Set up a dynamic threshold, thus if a candidate has a corner strength lower than
    % the threshold, it would be filtered out (to improve computational efficiency)
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    threshold = (max(max(Corner_strength_H_I1)))/5;

    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Counting the corners detected in both image and plot those corners
    % This is intermediate results and will not appear on homework report
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     figure
%     imshow(I1)
%     hold on;
    Actual_Corner_I1 = zeros(size_I1(1),size_I1(2));


    cnt_cor1 = 0;

    for n = 1:1:N
        [i,j] = find(Corner_strength_H_I1 == max(max(Corner_strength_H_I1)));
        Corner_strength_H_I1(i,j) = 0;
        cnt_cor1 = cnt_cor1 + 1;
        corner_loc1(cnt_cor1,1:2) = [i(1);j(1)];
```

```
%          plot(j(1),i(1),'ro');
          Corner_strength_H_I1((i(1)-5):1:(i(1)+5), (j(1)-5):1:(j(1)+5)) = 0;

    end

    if (scale_flag == 1)
        corner_loc1 = round(corner_loc1.*0.5);
    else
    end

    % for i = 11:1:size_I1(1)-10
    %     for j = 11:1:size_I1(2)-10
    %         if (Corner_strength_H_I1(i,j) > threshold) && ...
    %                 (Corner_strength_H_I1(i,j) == max(max(
    Corner_strength_H_I1(i-10:1:i+10,j-10:1:j+10))))
    %             Actual_Corner_I1(i,j) = 1;
    %             plot(j,i,'rx');
    %             cnt_cor1 = cnt_cor1 + 1;
    %             corner_loc1(cnt_cor1,1:2) = [i;j];
    %         else
    %         end
    %     end
    % end




end
```

7.7 project_corners.m

```
function [ angle, arc_corners ] = project_corners( save_img, corner_locs, N )
%% Corner projection, poprject the Harris Corners onto the unit circle
%  After all Harris Corners are projected, form the feature vector.
img_size = size(save_img);
column  = 5;
row = 6;
figure

for k = 1:1:img_size(2)
    subplot(column,row,k)
    imshow(save_img{k})
    hold on;
    c_loc = round(size(save_img{k}).*0.5);
    c_x = c_loc(1);
    c_y = c_loc(2);
    plot(c_y,c_x,'bo');
    for n = 1:1:N
        plot(corner_locs{k}(n,2), corner_locs{k}(n,1), 'ro');
    end
end
```

```
21
22 for k =1:1:img_size(2)
23     c_loc = round(size(save_img{k}).*0.5);
24     c_y = c_loc(1);
25     c_x = c_loc(2);
26     for n = 1:1:N
27         y = corner_locs{k}(n,1);
28         x = corner_locs{k}(n,2);
29         if ((x - c_x) == 0) && (y - c_y ==0 )
30             theta = 0
31         else
32             % Check the quadrat: if 1st and 4th qudrant
33             if ((x - c_x) >= 0)
34                 theta = atan((c_y - y)/(x - c_x));
35             else
36             end
37             % check the quadrant: if 2nd and 3rd quadrant
38             if ((x - c_x ) < 0)
39                 theta = atan((c_y - y)/(x - c_x)) + pi;
40             else
41             end
42
43         end
44
45         % Fix the angles in 4th quadrant from negative to 3/2 pi - 2 pi
46         if theta < 0
47             theta = 2*pi + theta;
48         else
49         end
50         angle{k}(n) = theta;
51     end
52     % Sort the anles
53     sort_vec = angle{k}(1:N);
54     sort_vec = sort(sort_vec);
55     angle{k}(1:N) = sort_vec;
56     angle{k};
57     % form the feature vectors based on arc length between consecutive
58     % projections points
59     arc_corners{k}(1) = 0;
60     for n = 2:1:N
61         arc_corners{k}(n) = angle{k}(n) - angle{k}(n-1);
62     end
63     arc_corners{k}(1) = 2*pi - sum(arc_corners{k});
64     arc_corners{k};
65     sum(arc_corners{k}); % check if sum of arc length equal to 2*pi
66 end
67
68
69
70
71 end
```

7.8 euc_matching.m

```
function [ classify_matrix ] = euc_matching( test_arc_corners, test_img,
    arc_corners, save_img, N)
%% Classification based on Euclidean distance
test_size = size(test_arc_corners);
training_size = size(arc_corners);

% Need to check the circular minimunm
for k = 1:1:test_size(2)
    test_vec = [test_arc_corners{k},test_arc_corners{k}];

    for t = 1:1:training_size(2)
        training_vec = arc_corners{t};
        for n = 1:1:N
            euc_vec(n) = sumsqr(training_vec(1:N) - test_vec(n:n+N-1));
        end
        classify_matrix(k,t) = min(euc_vec);
    end
end
%Now classify based on the minimum distance
for k = 1:1:test_size(2)
    match(k) = find(classify_matrix(k,:) == min(classify_matrix(k,:)));
    figure
    subplot(2,1,1)
    imshow(test_img{k})
    subplot(2,1,2)
    imshow(save_img{match(k)})
end


end
```

7.9 Trivial: Image Enhancement, Draw Circles, etc...

```
1
2  clear all; close all; clc;
3
4  N = input('Please pick a number N (N harris corners as interest points): ');
5  test_img_name = input('Please Slected the image you want to test (TestImg.jpg)
       : ');
6  %% Load And Process Training Images
7
8  disp('                   Starting Training Process...')
9  img_training = imread('Training.jpg');
10 img_size = size(img_training);
11 I_image = ones(img_size(1),img_size(2));
12
13 % figure
14 % imshow(img_training);  % Display the orginal image
15 img_gray = 0.2125*img_training(:,:,1) + 0.7154*img_training(:,:,2) + 0.0721*
       img_training(:,:,3);
16 img_gray = rgb2gray(img_training);
17
18
19 %% First Iteration of Training Image Segmentation: Try to extract the
       chracters
20 disp('Performing first iteration of image segmentation based on Otsu threshold
       ...')
21
22 I_image = image_segmentation( I_image, img_gray );
23
24 % figure
25 % imshow(I_image*255) %%% The result have some noise in it. Hence perform
       second iteration
26 imwrite(uint8(I_image*255),'training_first_iteration.jpg','jpeg')
27 % truesize
28
29 %% Second Iteration of Training Image Segmentation: Try to extract the
       chracters
30 disp('Performing second iteration of image segmentation based on Otsu
       threshold...')
31
32 I_image = image_segmentation( I_image, img_gray );
33
34 % figure
35 % imshow(I_image*255)
36 % truesize
37 imwrite(uint8(I_image*255),'training_second_iteration.jpg','jpeg')
38 %After second iteration, the result looks better.
39
40 %% Extract and process the segmented image
41
42 segmented_image = I_image;
43
```

```
disp('Extracting letters separately based on 8 neighberihood... ')
chars_lables = connected_component(segmented_image);

[save_img, corner_locs] = extract_letter(chars_lables, N);
disp('Formulating training feature vectors...')
[angles, arc_corners] = project_corners( save_img, corner_locs, N );
figure
for k = 1:1:9
 subplot(3,3,k)
 draw( k,N,save_img,angles,corner_locs )
end

 figure
for k = 10:1:18
 subplot(3,3,k-9)
 draw( k,N,save_img,angles,corner_locs )
end

figure
for k = 19:1:26
 subplot(3,3,k-18)
 draw( k,N,save_img,angles,corner_locs )
 end
```

```
function [ ] = draw( k,N,save_img,angles,corner_locs )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
img = save_img{k};

size_img = size(img);
new_img = zeros(2*size_img(1),2*size_img(2));
new_img(round(size_img(1)/2):1:(round(size_img(1)/2)+size_img(1) -1),...
    round(size_img(2)/2):1:round(size_img(2)/2)+size_img(2)-1) = img;

imshow(new_img)
hold on
angles{k};
    c_loc = round(size(save_img{k}).*0.5);
    c_x = c_loc(1) + round(size_img(1)/2);
    c_y = c_loc(2) + round(size_img(2)/2);
    plot(c_y,c_x,'bo');

xc = round(size_img(1));
yc = round(size_img(2));
radii = (xc^2 + yc^2)^(1/2);
viscircles([yc,xc],radii/1.6);

    for n = 1:1:N
        plot(corner_locs{k}(n,2) + round(size_img(2)/2), ...
            corner_locs{k}(n,1) + round(size_img(1)/2), 'ro');
        y_arc = round(radii*cos(angles{k}(n))/1.6);
        x_arc = round(radii*sin(angles{k}(n))/1.6);
        plot([c_y,(y_arc+c_y)],[c_x,-x_arc+c_x],'b')
```

```
        plot(y_arc+c_y, -x_arc+c_x, 'r*')
    end

end
```

```
close all
clear all
clc;

img = imread('Image1.jpg');
whos img

img_size = size(img);

for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
        if (img(i,j,1) == 0) && (img(i,j,2) == 0) && (img(i,j,3) == 0)
            img(i,j,:) = 255;
        else
        end
    end
end
img = ((img./255).^5).*255;
figure
imshow(img)

imwrite(uint8(img), 'conImage1.jpg','jpeg')

img = imread('Image6.jpg');
whos img
figure
imshow(img)
img_size = size(img);

img(40:img_size(1)-40,30:img_size(2)-30,1) = 255 - img(40:img_size(1)-40,30:
    img_size(2)-30,1);
img(40:img_size(1)-40,30:img_size(2)-30,2) = 255 - img(40:img_size(1)-40,30:
    img_size(2)-30,2);
img(40:img_size(1)-40,30:img_size(2)-30,3) = 255 - img(40:img_size(1)-40,30:
    img_size(2)-30,3);


figure
imshow(img)

imwrite(uint8(img), 'conImage6.jpg','jpeg')

img = imread('Image5.jpg');
figure
subplot(1,2,1)
imshow(img)
img_size = size(img);
img1 = double(img);
```

```
46 % img1 = ((img1./255).^3).*255;
47 img1 = ((img1./255).^20).*255;
48 img1 = uint8(img1);
49
50 subplot(1,2,2)
51 imshow(img1)
52 whos img
53 whos img1
54 imwrite(uint8(img1), 'conImage5_1.jpg','jpeg')
```