Tracking Text in Mixed Mode Documents

J. Patrick Bixler

TR 88-19

TRACKING TEXT IN MIXED MODE DOCUMENTS

J. Patrick Bixler

Department of Computer Science
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

June 10, 1988

Abstract. This paper describes a method for extracting arbitrarily oriented text in documents containing both text and graphics. The technique presented is inspired by the tracking algorithms frequently found in raster to vector conversion systems. By indentifying text components in the document, reducing the resolution of the image by the size of the characters, and then tracking the centers of the character components, all text strings can be removed and subsequently reoriented to the horizontal. They can then be presented for automated character recognition. A by-product of the method is that characters are automatically grouped together to form words and/or phrases. We give a detailed description of the algorithm, discuss its strengths and weaknesses, and present some sample results obtained from a typical city street map.

1. INTRODUCTION

Optical character recognition techniques have now progressed to the point where most documents that consist primarily of large blocks of text can be easily processed by automatic means. Likewise, documents that consist primarily of line art (technical drawings, for example) can also be processed by any of a number of recently proposed raster to vector conversion techniques. Most documents, however, are not as homogeneous as these respective systems would like, rather containing significant amounts of both text and graphics. Clearly, any robust document processing system must be able to deal effectively with both the text and graphic content of a given document. To that end, this paper details a technique for segmenting an arbitrary document into its text and graphics components.

A number of techniques for processing mixed mode documents have been proposed in the recent literature [2,4,5]. Some have attempted to solve significant pattern recognition problems while others have restricted their attention to a specific type of document. The work presented here uses several of the features of the algorithm proposed in [5]. The basic approach of finding text components and reducing the resolution of the image is similar, whereas the method for actually grouping the components to form words and phrases is fundamentally different. Rather than use the Hough transform to group characters as in [5], we attempt to track adjacent or nearly adjacent character components to form as long a string as possible. This approach is inspired by the tracking algorithms frequently found in raster to vector conversion and/or polygonal approximation systems, particularly those of [9,12].

The following is a brief overview of the algorithm as it is currently implemented. The process begins with a high resolution (300 dpi), binary scan of the document. The first step consists of finding all connected components within the image and computing some simple attributes such as height, width, density, etc. Then, based on a pre–determined parameter that gives the size of the text to be detected, components are segmented into those that are likely to be text and those that are not. The non–text components are subtracted from the image and stored, perhaps to be processed by a line following system. The remaining image is then sampled with the center of each text component being mapped to the nearest sampled point. Once this reduced resolution image is formed, adjacent pixels are tracked to form the longest straight line possible. In the ideal situation, the centers of text components that were adjacent characters of a word in the original document would be mapped to adjacent pixels in the sampled image. Of course, the ideal is rarely the case and so the algorithm must also be able to track nearly adjacent pixels as well. Once a text string has been tracked the orientation of the line joining the first and last character is noted and the entire string is rotated to horizontal in order to facilitate automatic character recognition.

The remainder of the paper discusses these steps in detail. Section 2 discusses some of the problems involved with identifying character components from a limited set of attributes, and Section 3 gives the parameters used to create the reduced resolution image from the original document. Section 4 is the main section of the paper and details the tracking algorithm. Section 5 shows some results of the algorithm as applied to a typical city street map, and section 6 provides some conclusions and suggestions for subsequent work.

2. FINDING CHARACTER COMPONENTS

Any good connected component finding algorithm could be used in this context. We have chosen to implement a variation of the stack based, scan line algorithm as presented in [6]. The assumption is that the entire image is in memory and so the cost of accessing any given pixel is constant, that is, the image does not have to be completely processed in scan line order. Ultimately the finding of connected components could be done in hardware and so the efficiency of this part of the algorithm is not a major concern. We have also chosen to find 4–connected components rather than 8–connected components thereby increasing the number of components eventually found. The trade–off here is in risking that a single character might be broken into two pieces in return for improving the chances that characters will be cleanly segmented from each other and from the surrounding components.

Once the connected components have been found, the coordinates of its circumscribing rectangle are recorded. In [5], the authors use the pixel height, width, area, and aspect ratio of this rectangle, as well as the density of the image within the rectangle to decide if the corresponding component is likely to be a character. We have found, however, that area, aspect ratio, and image density can be misleading so as to cause both types of errors, that is, mislabeling text components as graphics and vice–versa. An upper–case I in a font such as Monaco illustrates the problem. If the string containing this character is oriented horizontally (vertically) as in Figure 1, the density of the character is 1.0, the maximum it can be. Also, if the resolution and/or the quality of the scanned image is such that the character is exactly one pixel wide (high), then the area and the aspect ratio are simply equal to the height (width) of the character. In Figure 2, where the character is oriented at 45 degrees, the density is the minimum possible value for any connected component, namely, $1/\sqrt{area}$. Thus, for legitimate text components density can take on any possible value and, therefore, has no discriminating power, and in some cases area and aspect ratio become simply height or width. The point is that one gains no additional useful information about a component by looking at density, aspect ratio, and area as opposed to looking only at height and width.

Fairly accurate segmentation of text and graphics components can be made by making proper use of only the height and width attributes. Let *res* represent the resolution at which the original document was scanned, let *ptsize* be the point size of the text to be detected, and let h_c and w_c

be the height and width of the given component. One obvious approach would then be, allowing for characters oriented at an angle, to mark a component as text if both the height and width are below the appropriate pixel threshold, that is if

$$h_c <= res/72 * ptsize * 1.4$$

and

$$w_c <= res/72 * ptsize * 1.4$$

The factor of 1.4 on the right hand side of the inequalities is to account for characters that are oriented at an angle, and the 72 is the standard number of printer's points per inch. The problem of the upper–case I again points out that imposing a lower bound on the height and/or width is inappropriate. The potential difficulty with this is that it mismarks extremely small components as text, a problem which could be significant in a noisy image. Noting that every character has at least one dimension that is within about one half of its respective point size, we opt for marking a component as text if *either* of the following conditions hold:

$$(res/72 * ptsize * 0.5 <= h_c <= res/72 * ptsize * 1.4) AND\ (w_c <= res/72 * ptsize * 1.4)$$

or,

$$(res/72 * ptsize * 0.5 <= w_c <= res/72 * ptsize * 1.4) AND\ (h_c <= res/72 * ptsize * 1.4)$$

To allow for the simultaneous detection of a range of font sizes within the same document we can modify the above inequalities by replacing *ptsize* with *minptsize* on the left sides and with *maxptsize* on the right, where *minptsize* and *maxptsize* have the obvious meanings. Note, however, that the larger this range is the less discriminating the test becomes. Finally, we notice that this test will most likely exclude subscripts, superscripts and small puntuation, a problem which can be solved after a string has been tracked by looking for nearby small components.

3. RESOLUTION REDUCTION

Once the components have been segmented, those identified as graphics can be subtracted from the image leaving only the text. The graphics portion of the image can either be stored in its original bitmap form or can be processed by an appropriate vectorizing system. The resolution of the remaining image is then reduced by essentially sampling the image based on the size of character components. Each text component is mapped to the sample point nearest to the center of that component, see Figure 3. The ideal sampling rate would be such that adjacent characters in the original image get mapped to adjacent pixels in the reduced image. It is, of course, possible

to achieve such a mapping only in the case of a document that contains a single font size and no proportional spacing of characters. Because of the variety in size and spacing there will necessarily be gaps and overlaps in the reduced image.

Referring to Figure 3, we let the sampling rate be equal to $(dpi/72) * 0.5 * minptsize$. With this sampling rate, adjacent characters whose dimensions are approximately *minptsize* will be mapped to pixels that are one apart from each other rather than adjacent. If the gap between the adjacent characters in the original image is greater than $0.5 * minptsize$, then they will be mapped to pixels that are two apart, and so on. With most fonts, however, the width of a character is usually closer to one half the point size and so adjacent characters (that are separated by less than $0.5 * minptsize$) will be mapped to adjacent pixels. It is this fact that motivates the 0.5 in the formula for the sampling rate. Of course, if adjacent characters are less than $0.25 * minptsize$ wide and are close enough, then they may be mapped to the same pixel in the reduced image. In this case, the components are linked together and a pointer to the list is stored at the appropriate pixel.

When *maxptsize* is different from *minptsize* the potential exists for adjacent characters to be mapped to pixels that are at significant distances in the reduced image. If characters from the font of size *maxptsize* are adjacent, assuming again that they are approximately $0.5 * maxptsize$ wide and are no further than $0.5 * maxptsize$ apart, then they will be mapped to pixels that are at a distance of $maxptsize/(0.5 * minptsize)$ from each other. Keeping $maxptsize <= 2 * minptsize$ will insure that such characters are no more than four pixels apart. Clearly, then, the range of fonts to be detected determines how far apart adjacent characters can be in the reduced image and, in turn, influences the complexity of the tracking algorithm.

4. TRACKING ALGORITHM

The next step in the process is to actually track adjacent characters as represented by the reduced image. We look here to a number of algorithms that have been used to produce polygonal approximations to curves in binary images. In particular we note the algorithms of [9] and [12] as being appropriate starting points for our application. There are, however, two important differences between our application and those for which the existing algorithms were developed. First, since we are attempting to track text strings, we are looking at data that *should* be straight rather than using straight segments to approximate data that might really be curved. This could allow us to impose slightly more stringent restrictions on the data as it is being tracked. And secondly, because of the effect of the resolution reduction discussed in the previous section, we are faced with tracking data points that are not adjacent. These two facts lead to the following tracking algorithm.

The algorithm begins by scanning the entire (reduced) image from left to right and top to bottom searching for a starting point for the first string. Any pixel that is non–zero (ie. corresponds

to the center of a text component) can be a starting point. After a starting point is found the reduced image is searched in a neighborhood of that point to find the nearest text component. The size of the neighborhood searched is given by $(2 * max(h_c, w_c))/(0.5 * minptsize)$. This ensures that the next component really is adjacent to the starting point and not a character from a different nearby string. Once the second point is found, the horzontal and vertical offsets from the first point are recorded, and the algorithm then searches in the appropriately sized neighboorhood offset by those same amounts from the second point, and so on. When checking for the nearest component, note that distances are measured in the original image rather than in the reduced image. As soon as two points have been tracked, the angle that the line joining the centers of the two corresponding components forms with the horizontal is computed. Tracking then continues by searching in the appropriately offset neighborhood subject to two additional constraints which insure that the components lie on a straight line. The first is a global angle constraint which requires that the angle between the line joining the starting component to the most previously tracked component and the line joining the starting component to the component being considered is below some threshold. The second is a local angle constraint which requires that the angle between the line joining the starting component to the most previously tracked component and the line joining the previously tracked component to the component being considered is below another threshold. The component under consideration must also be about the same size as the most previously tracked component. If all of these constraints are met, the component is added to the string and tracking continues. Finally, when tracking ends for any reason, the algorithm reverses direction and attempts to extend the string by tracking in the opposite direction. This is necessary because we cannot guarantee that tracking began at one of the end components of the string.

We define a number of identifiers before summarizing the individual steps of the algorithm. Let *first*, *current*, and *next* be the initial component in the string, the currently tracked component, and the next component under consideration by the tracker, respectively. Let $rowoff$ and $coloff$ be the average row and column offsets between successive components for the entire string. Let h_{curr}, w_{curr}, h_{next}, and w_{next} be the height and width of the current and next components, respectively, and let α_g and α_l be the global and local angle thresholds.

STEP 1) Scan the reduced image left to right and top to bottom until a text component is found. Set *first* and *current* to this component, set $rowoff$ and $coloff$ to 0.

STEP 2) Using *NC*, the nearest component rule, find the nearest component to *current* in a neighborhood offset from *current* by $(rowoff, coloff)$. If a component is found, set *next* equal to the component found, or to 0 if no component is found.

STEP 3) If *next* is not 0, apply A, the angle rule. If A holds, add *next* to the string, update *current*, $rowoff$, and $coloff$ and go to STEP 2. If *next* is 0 or if A fails, continue to STEP 4.

STEP 4) Swap *first* and *current*, negate $rowoff$ and $coloff$ and continue tracking in the reverse direction.

STEP 5) (Same as STEP 2)

STEP 6) (Same as STEP 3, except go to STEP 5 or EXIT)

NC *(nearest component rule)*: Set *next* equal to the nearest component to *current* within the given neighborhood, where distances are measured in the original image. If no components can be found in this neighborhood, or if either h_{curr} and h_{next} or w_{curr} and w_{next} differ by more than a factor of 2, then set *next* to 0.

A *(angle rule)*: The angle between the line joining *first* to *current* and the line joining *first* to *next* must be less than or equal to α_g, and the angle between the line joining *first* to *current* and the line joining *current* to *next* must be less than or equal to α_l.

5. EXPERIMENTAL RESULTS

Figure 4 shows the test image consisting of a typical city street map which was digitized at 300 dots per inch on a Ricoh IS-400 scanner. Text is oriented at a variety of angles and is printed in fonts ranging from about 8 to 12 pts. The map itself is approximately 8 by 10.5 inches yielding an image that is 3150 pixels high by 2400 pixels wide and, with one bit per pixel, requires a buffer of just under 1MB. Two additional buffers are needed, one to mark the connected components as they are being marked and one to hold the text strings after they have been re–oriented. The entire system was written in C and was developed on a Macintosh II running the A/UX operating system. All times reported were for a CPU configuration that included 5MB of memory so that swapping was keep to a minimum. No optimization of the code was attempted before performing the experiments.

Figures 5 and 6 show the the two segmented portions of the test image. The few characters that remain in Figure 6 are missed primarily because they are touching some larger graphic component or some other character. There are 579 connected components in the image (almost all of the streets are connected and form a single graphic component) of which 477 qualify as text. These components are tracked to form 107 strings of length 2 or more and are presented horizontally in Figure 6.

There are only five errors that are not due to broken characters or characters which are touching other parts of the image, and that could be considered the fault of the tracker. The first is the second string in column two of Figure 6, where an arrow symbol was included as part of the string. This also occurs in the last string in column 4. Two strings, the sixth and the eight in column 4, have been presented upside down. Prior to rotation, each string is reoriented so that the left most character

becomes the first character in the string. As long as text is written so that it is right side up in the document, everything will turn out correct. If not, the potential exists for a string to be presented upside down since the algorithm has no way of knowing the orientation of a single character. In such a case, character recognition would fail the first attempt and could retry after rotating by 180 degrees. The two errors observed here are with strings that are very nearly vertical in the image. Apparently the centers of the first and last components in these strings line up so that the strings appear to be written upside down. The final error is near the bottom at the center of the image where the word 'ROANOKE' comes very near the vertical abbreviation 'ST'. Apparently the 'R' and 'O' are touching and form a large component which is then actually closer to the 'S' than it is to the 'A'. Such mis–tracking should be very unlikely once the correct direction is established and we note that this particular error occured on the first character of a string. Finally, note that some strings appear a bit off from horizontal in Figure 7. This is because the line joining the first and last character is not exactly parallel to the baseline of the string. Even so, the tilt should be well within the capabilities of any good character recognition system.

Although most of the text in this image is printed in upper–case, there are a few lower–case strings ('2nd', '3rd', for example). Even though these strings were tracked successfully, it is conceivable that mixed case strings might have problems since the centers of their indivual characters do not usually form a very nice straight line. It might be advantageous in this case to use not only the centers of the components but also their corners as tracking points. Tracking could continue as long as at least one of these tracking points satisfied all of the tracking constraints.

It is interesting to note the word 'FRANKLIN' near the lower left of the image. Although it has a gently curving path it gets tracked correctly and appears as one string in the output. The same is true of the two occurrences of the word 'SHENANDOAH' near the top of the map. The angle threshold is slack enough to allow the tracker to follow the string as it takes the bend in the road. It would, perhaps, be better if the tracker stopped where the word changed direction and picked up a new string. Then there could be some postprocessing to rejoin such strings so that they would appear straight in the final output for the character recognition step. Note also that the large title 'CENTRAL ROANOKE' is segmented as graphics, since it was printed in approximately an 18pt font and the test run was set to detect text in the 8–12pt range. We could easily iterate the algorithm for different point sizes in order to pick up such a title or any extra fine type that might be in the document.

The processing times for various steps of the algorithm are listed in Table 1. Clearly, the bulk of the time is spent simply in finding connected components This is a fairly simple task and could easily be implemented in hardware. It would make sense to position the segmentation and tracking procedure immediately after the scanning process, and so the time for reading and the image from

disk would be eliminated. This leaves only the time for deciding if the component is text and then tracking the strings. Assuming that the algorithm could be implemented in hardware and that doing so would reduce the time by an order of magnitude, the entire process could be done in less than 2 seconds which is about the time it takes to scan a document.

6. CONCLUSIONS

We have presented a fast, effective, and reasonably robust algorithm for segmenting text strings from graphics in mixed mode documents. Image components are first segmented into text and graphics based primarily on the height and width of their surrounding rectangles, and the algorithm then uses a fast tracking technique to link the characters together into longer text strings. Once the characters have been tracked they can be rotated to horizontal for processing by automated character recognition. By using this tracking approach the computational complexity of other techniques, such as the Hough transform, is avoided.

The only parameters to the algorithm are the resolution of the input image and the range of text sizes to be detected. The current algorithm expects the input image to be of fairly high quality, as no attempt is made to repair broken characters or to detect characters that are partially obscurred by other image elements. Some attempt could be made to infer the existance of these partially obscured characters by analysing the patterns of the strings that are found. Although strings are also expected to form a reasonably straight line, the algorithm can tolerate a somewhat gently curving path. The current algorithm could easily be modified to join together adjacent words to form longer phrases if desired. It could also be made to straighten the path of a string that changes direction as might be the case of a street name that is written to follow the contour of the street. Some additional logic could be developed to locate punctuation, subscripts, superscripts, etc., and to fuse them with the corresponding text.

We have demonstrated the effectiveness of this approach to text segmentation by applying the algorithm to a typical city street map that contains text strings of a variety of sizes and orientations. Other applications could include survey maps where distances and bearings are typically written along the corresponding boundaries, contour maps, and other technical drawings. By using color separation in the scanning process, the technique could also be applied to standard topographic maps.

REFERENCES

1 **Black, W, T P Clement, J F Harris, B Llewellyn, and G Preston** 'A general purpose follower for line-structured data' *Pattern Recognition.* 14 (1981) 33–42.

2 **Bunke, H** 'Automatic Interpretation of lines and text in circuit diagrams' in *Pattern Recognition Theory and Applications*, J Kittler, K S Fu, and L F Pau, Editors, D. Reidel Boston, (1982) pp 297-310.

3 **Clement, T P** 'The extraction of line-structured data from engineering drawings' *Pattern Recognition* 14 (1981) pp 43–52 .

4 **Ejiri, M, S Kakumoto, T Miyatake, S Shimada, and H Matsushima** 'Automatic Recognition of design drawings and maps' *Proc. 7th Intl. Conf. on Pattern Recognition* (1984) pp 1296–1305.

5 **Fletcher, L A and R Kasturi** 'Segmentation of binary images into text strings and graphics' *Proc. SPIE Conf. Applications of Artificial Intelligence* 786 (1987) pp 533-540.

6 **Pavlidis, T** *Algorithms for Graphics and Image Processing* Computer Science Press, Rockville, MD. (1982).

7 **Ramachandran, K** 'A coding method for vector representation of engineering drawings' *Proc. IEEE* 68 (1980) pp 813–817.

8 **Ramer, U E** 'An iterative procedure for the polygonal approximation of plane curves' *Comput. Graphics Image Process.* 1 (1972) pp 244–256.

9 **Sklansky, J and V M Gonzalez** 'Fast polygonal approximation of digitized curves' *Pattern Recognition* 12 (1980) pp 327–331.

10 **Tomek, I** 'Piecewise linear approximations' *IEEE Trans. Computers* C-23 (1974) pp 445–448.

11 **Wall, K and P Danielsson** 'A fast sequential method for polygonal approximation of digitized curves' *Comput. Vision Gr. Image Process.* 28 (1984) pp 220-227.

12 **Watson, L T and K Arvind, R W Ehrich, R M Haralick** 'Extraction of lines and regions from grey tone line drawing images' *Pattern Recognition* 17 (1984) pp 493–507.

FIGURE CAPTIONS

Fig. 1. Character I with density equal to 1.0.

Fig. 2. Character I at 45 degrees with density equal to $1/\sqrt{area}$.

Fig. 3. Character components mapped to nearest point in sampled image.

Fig. 4. Original street map.

Fig. 5. Text only portion of Figure 1.

Fig. 6. Graphics only portion of Figure 1.

Fig. 7. Text strings tracked and oriented to horizontal.

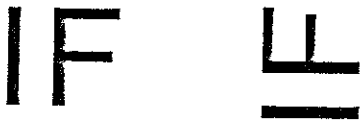


Fig 1

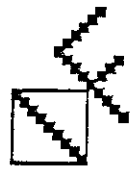


Fig 2

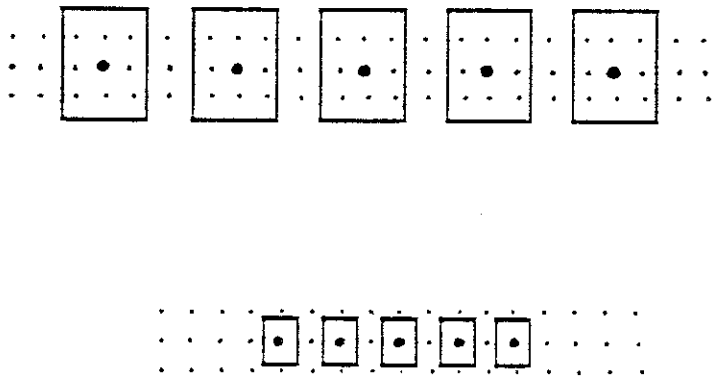
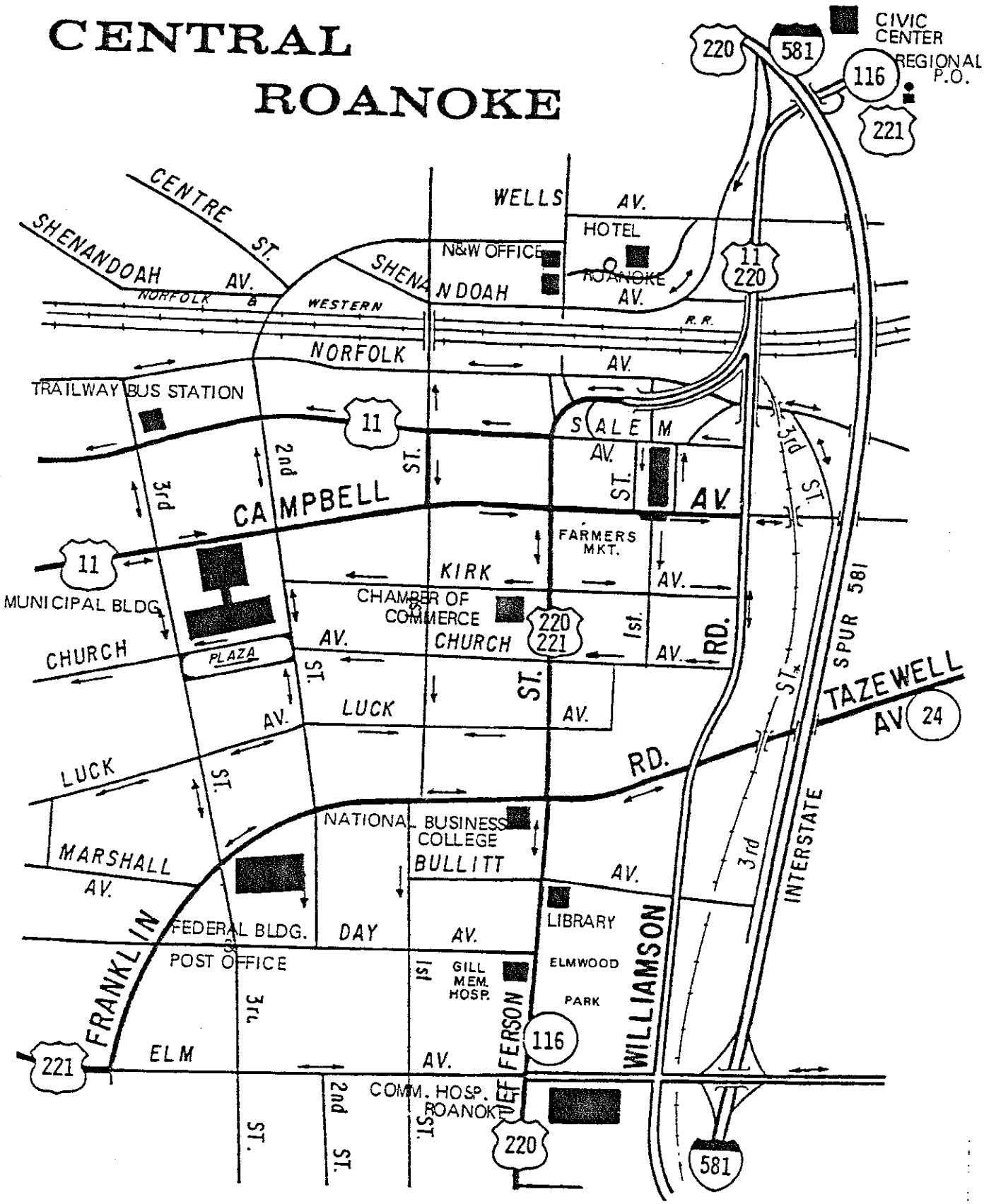
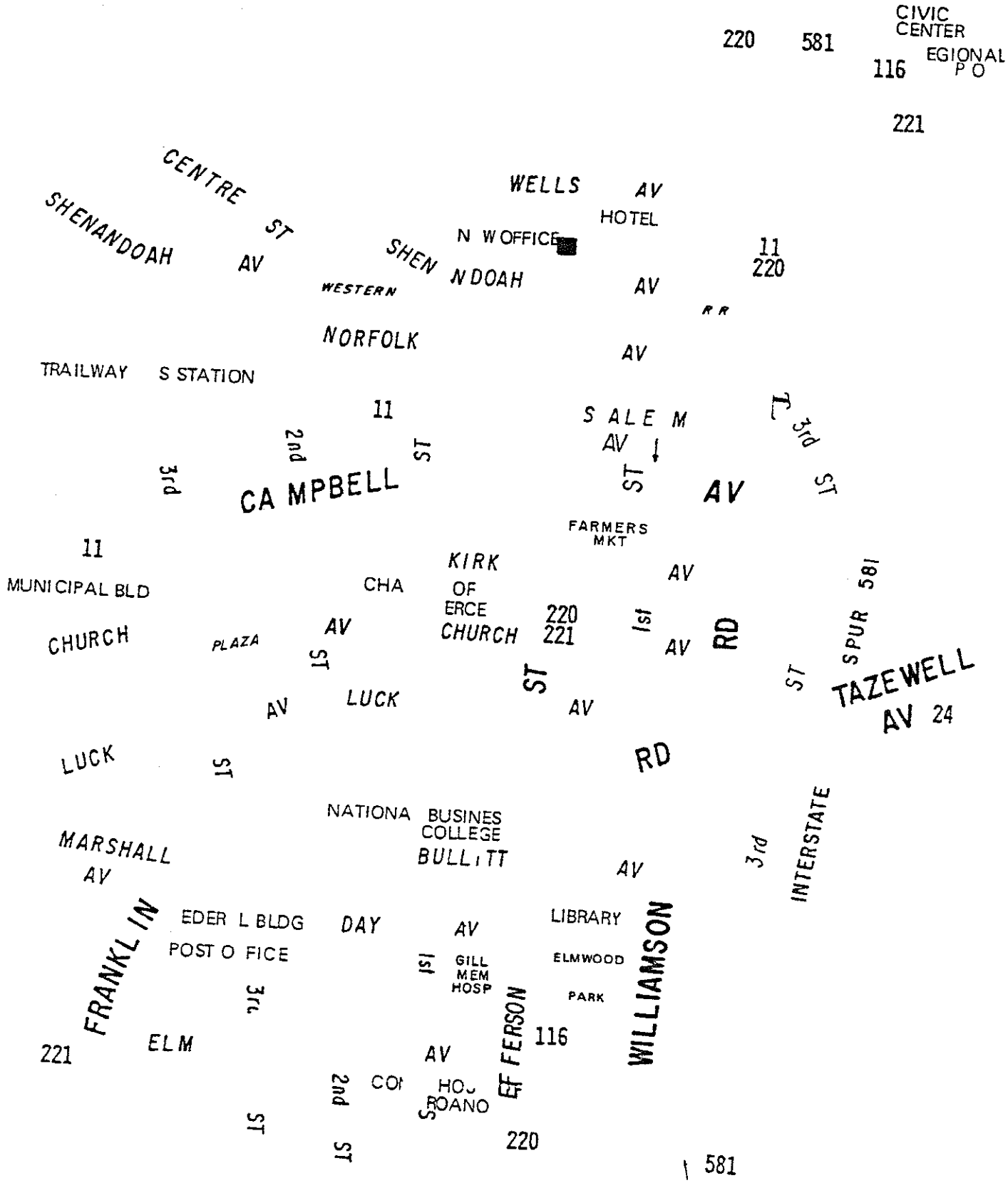
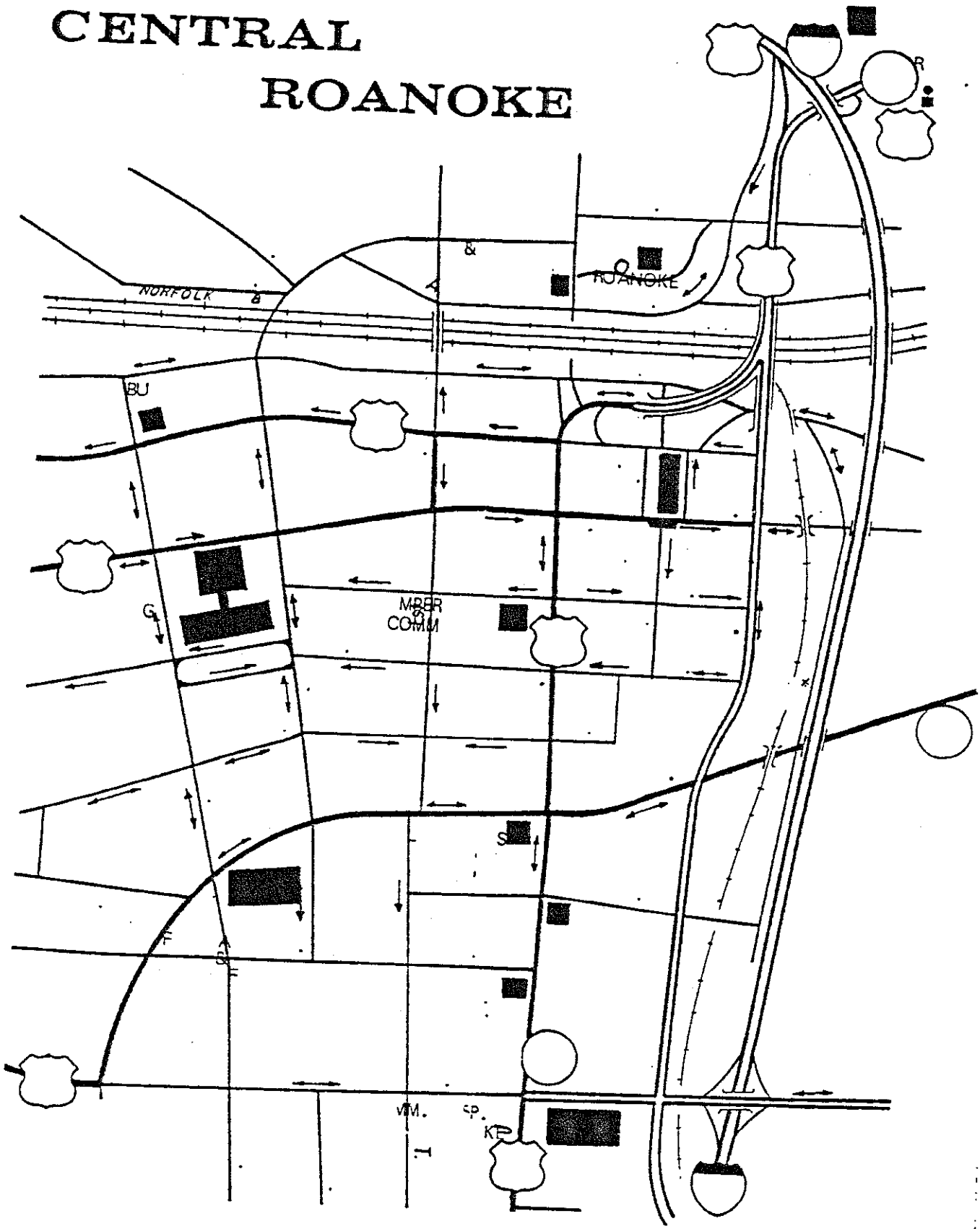


Fig 3


CENTRAL
ROANOKE
CIVIC CENTER
REGIONAL P.O.
220
581
116
221
WELLS AV.
HOTEL ROANOKE
N&W OFFICE
CENTRE ST.
SHENANDOAH AV.
SHENANDOAH
NORFOLK & WESTERN R.R.
NORFOLK AV.
TRAILWAY BUS STATION
11
220
SALEM AV.
2nd ST.
3rd ST.
CAMPBELL AV.
FARMERS MKT.
KIRK AV.
MUNICIPAL BLDG.
CHAMBER OF COMMERCE
CHURCH
PLAZA
CHURCH AV.
1st RD.
SPUR 581
TAZEWELL AV
24
LUCK AV.
LUCK RD.
NATIONAL BUSINESS COLLEGE
MARSHALL AV.
BULLITT AV.
FEDERAL BLDG.
DAY AV.
LIBRARY
ELMWOOD PARK
POST OFFICE
GILL MEM. HOSP.
FRANKLIN
JEFFERSON
WILLIAMSON
INTERSTATE
ELM AV.
COMM. HOSP. OF ROANOKE
2nd ST.
3rd ST.


CIVIC CENTER
220
581
116
EGIONAL
P O
221
CENTRE ST
SHENANDOAH AV
WELLS AV
HOTEL
N W OFFICE
SHEN NDOAH AV
11
220
WESTERN
R R
NORFOLK
AV
TRAILWAY
S STATION
11
S ALE M
AV
2nd ST
3rd
CA MPBELL
ST
AV
3rd ST
FARMERS MKT
11
KIRK
AV
MUNI CIPAL BLD
CHA
OF
ERCE
220
CHURCH
221
1st
AV
RD
581
CHURCH
PLAZA
AV
ST
ST
SPUR
ST
TAZEWELL
AV 24
AV
LUCK
AV
LUCK
ST
RD
NATIONA
BUSINES
COLLEGE
MARSHALL
BULLITT
AV
AV
3rd
INTERSTATE
FRANKLIN
EDER L BLDG
DAY
AV
LIBRARY
POST O FICE
1st
GILL
MEM
HOSP
ELMWOOD
PARK
3rd
221
ELM
AV
116
JEFFERSON
WILLIAMSON
2nd
COI
HO
SROANO
ST
ST
220
581

CENTRAL ROANOKE



CIVIC
CENTER
220
581
EGIONAL
116
P O
221
CENTRE
WELLS
AV
HOTEL
SHENANDOAH
ST
N W OFFICE
11
SHEN NDOAH
220
AV
AV
WESTERN
R R
NORFOLK
AV
TRAILWAY
S STATION
S ALE M
3rd

11
AV
2nd
ST
ST
ST
AV
3rd
CA MPBELL
FARMERS
MKT
581
11
KIRK
AV
CHA
OF
MUNICIPAL BLD
1st
S PUR
ERCE
220
RD
AV
CHURCH
221
AV
TAZEWELL
CHURCH

PLAZA
ST
ST
ST
24
AV
LUCK
AV
AV
RD
LUCK
ST
INTERSTATE
NATIONA
BUSINES
3rd
COLLE E
MARSHALL
BULLITT
AV
AV
WILLIAMSON
LIBRARY
FRANKLIN
AV
EDER L BLDG
DAY
ELMWOOD

POST O FICE
1st
GILL
MEM
HOSP
FERSON
PARK
116
ELM
AV
221
2nd
COI
HO
ANO
ST
220
ST
581

Table 1. Performance results for map

task	sec
read 1MB image	24.3
find 579 components	165.6
segment components	0.6
track 107 strings	5.2
rotate 107 strings	65.5