http://dx.doi.org/10.14400/JDC.2014.12.2.285

이차원 PDF417 바코드에서 데이터 코드워드의 고속 디코딩 알고리즘*

[1]김영중, [2]조영민, [1]이종연
[1]충북대학교 디지털정보융합학과, [2]충북대학교 컴퓨터교육과

High-Speed Decoding Algorithm of Data Codeword in Two-Dimensional PDF417 Bar Code

Young-Jung Kim[1], Young-Min Cho[2], and Jong-Yun Lee[1]
Dept. of Digital Informatics and Convergence, Chungbuk National University[1]
Dept. of Computer Education, Chungbuk National University[2]

요 약 PDF417은 2차원 바코드로서 사용범위가 넓고 방대한량을 압축하여 저장하는 능력을 가지고 있다. 이러한 특성을 가진 PDF417은 현재 중요한 문서의 위조나 변조를 막기 위한 다양한 용도로 쓰이고 있다. 한편 기존의 PDF417 바코드의 디코딩 기술은 단순히 기본 표준규격서인 AIM에서 제시한 디코딩하므로 비효율적이라 할 수 있다. 따라서 본 논문에서는 PDF417 바코드를 보다 빠르고 효율적인 디코딩 알고리즘을 제시하고자 한다. 결과적으로 제안하는 디코딩 알고리즘은 기존 방법과 비교하여 불필요한 디코딩 과정을 제거함으로서 빠르고 효율적인 디코딩 알고리즘이 될 것이다.

주제어 : PDF417 바코드, 바코드 검증기, 디코딩, 데이터 코드워드

Abstract Two-dimensional PDF417 bar code has a wide range of use and has a storage capacity to compress a large amount of data. With these characteristics, PDF417 has been used in various ways to prevent the forgery and alteration of important information in documents. On the other hand, previous decoding methods in PDF417 barcode are slow and inefficient because they simply employ the standard specifications of AIM (Association for Automatic Identification and Mobility). Therefore, this paper propose an efficient and fast algorithm of decoding PDF417 bar code. As a result, the proposed decoding algorithm will be more faster and efficient than previous methods.

Key Words : PDF417 BarCode, BarCode Verifier, Decoding, Codeword

1. 서론

지금까지 바코드 인식 관련 연구는 지속적으로 연구되어왔으며, PDF417 2차원 바코드 디코딩 기술에 대한 기존연구에는 세 가지가 있다.[1][2][3] 하지만 기존 연구들은 다음과 같은 비효율적 문제점이 존재한다. 첫째, 기

* 본 논문은 미래창조과학부 및 정보통신산업진흥원의 산학협력 특성화 지원사업의 연구결과로 수행되었음(NIPA-2013010918).
Received 3 January 2014, Revised 7 February 2014
Accepted 20 February 2014
Corresponding Author: Jong-Yun Lee(Dept. of Digital Informatics and Convergence, Chungbuk National University)
Email: jongyun@chungbuk.ac.kr

ISSN: 1738-1916

© The Society of Digital Policy & Management. All rights reserved. This is an open-access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.otg/licenses/by-nc/3.0), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

존 방법들은 PDF417의 최소 단위인 모듈의 폭과 높이를 구하는 과정에서 이미지의 전체를 비교함으로써 많은 시간이 필요하다. 둘째, 모든 Start & Stop Pattern과 Row Indicator의 위치를 찾고 정보를 해석 한다. 마지막으로, 표준 규격인 AIM[1][4]에서 제시하는 방법으로 PDF417 바코드가 이미지로 들어왔을 때 바코드의 폭을 측정하여 단위 모듈과 바(bar) 및 스페이스(space)의 폭을 비교하여 해당 기호문자의 바 공간서열을 얻어 코드워드 세트를 변환하고, 이후에 오류 복원 코드워드와 데이터 코드워드를 비교하는 방식으로 불필요한 과정이 많이 포함되어 있다.[1]

따라서 본 논문에서는 위와 같은 비효율적인 방법의 해결을 위해 기존에 제안된 디코딩 기법을 보완한 고속 디코딩 기법을 제안하고자 한다. 즉 본 논문에서는 기존 AIM 기반의 디코딩 알고리즘 보다 더 빠르게 해석이 가능한 PDF417 바코드의 디코딩 알고리즘을 제안한다. 결과적으로 본 논문의 연구결과는 추후 모듈 폭을 구하는 과정을 최소화하였으며, Start & Stop Pattern과 Row Indicator의 불필요한 해석을 제거하여 디코딩 속도를 줄였다는 데 의미를 둔다.

2. 관련 연구

정정구 외(2002)는 바코드 이미지로부터 바코드 영역을 추출한 후에 바-스페이스 패턴을 추출하고, 이를 코드워드로 변환한 다음, AIM 표준 규격에 따른 원래의 데이터로 디코딩 하는 알고리즘 제안하였다.[1]

Jayasuriya(2007)은 [Fig. 1]과 같이 bit shifting decoding algorithm를 사용한 디코딩 방법을 제안하였다.[2] 이 제안 방법은 이미지 기술로 디코딩하기 때문에 이미지가 정확할 경우 100%정확성을 보여준다.

```
Begin
1:   imagedata:=
2:       LOAD_DECODED_IMAGE_GATA(file);
3:   height := imagedata[0];
4:   width := imagedata[1];
5:   decodeImage :=
6:       CREATE_BLANK_IMAGE(height, width);
7:   for i∈(1, ..., height) do
8:       for j∈(1, ..., width) do
9:           location := i * height + j;
10:          value := imagedata [location];
11:          value := value << 5;
12:      endfor
13:  endfor
14:  SETPIXEL_VALUE(decodeImage, i, j );
End
```

[Fig. 1] Algorithm of decoding the bit shift

바코드가 이미지로 들어왔을 때 이미지에서 바코드 데이터의 높이와 넓이를 측정하고 측정된 값을 수치로 배열화한다. 배열화된 값에서 5만큼 이동한 다음 이진화를 처리하여 0과 1로 구분하고 AIM 표준규격에서 따라 클러스터 표에 값을 이용해 디코딩 값을 구한다.

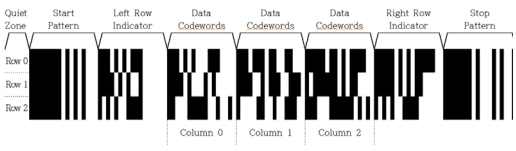
Song (2013)은 PDF417 바코드가 이미지로 들어왔을 때 바와 스페이스의 폭을 측정하여 바코드의 최소단위인 모듈을 얻어 바와 스페이스의 수치배열에 따른 값으로 변환하고, 이후에 오류 복원 코드워드와 데이터 코드워드를 비교하는 알고리즘을 제시하였다.[3]

이러한 기존 연구들을 살펴보면 이미지 기술로 디코딩[5][6]하거나 AIM 표준 규격에 따라 디코딩 기술[7][8][9]로 인해 불필요한 부분까지 디코딩하고 있다.

3. 새로운 PDF417 디코딩 알고리즘

이장에서는 PDF417을 디코딩하기 위한 과정을 설명하고 코드워드에 저장되어 있는 정보를 해석하기 위한 알고리즘에 대해 기술한다.

3.1 PDF417 구조



[Fig. 2] Structure of PDF417 barcode

PDF417 바코드 체계는 [Fig. 2]와 같이 Start Pattern

1) AIM(Association for Automatic Identification and Mobility) : 자동식별 및 이동성 기술 솔루션 제공 업체를 대표하는 국제 무역 협회. 바코드 및 RFID 등에 관한 표준 제시

과 Stop Pattern, PDF417의 행을 의미하는 Row, 열을 의미하는 Column (Start & Stop Pattern과 Indicator는 포함되지 않음), Start & Stop Pattern 안쪽에 붙어있는 Row Indicator로 구성되어 있다.[10]

PDF417 데이터 코드워드의 첫 번째 코드워드(Row 0, Column 0)에는 PDF417에 몇 개의 데이터 코드워드가 있는지 개수를 나타내어주는 부분이다.

데이터 코드워드는 Text Compaction Mode, Byte Compaction Mode, Numeric Compaction Mode의 3가지 표현 양식을 가지며, 각 모드 간에는 Mode Latch와 Mode Shift 지시자를 사용하여 Mode를 바꿀 수 있어 심볼안에 여러 양식을 혼합하여 쓸 수 있다. [11]

코드워드는 심볼의 바와 스페이스의 수치배열에 따라 0~928 사이의 값을 가지는 929개의 코드워드 값을 정의하지만, 데이터 코드워드는 0~899까지는 데이터를 나타내고, 나머지 900~928는 제어를 위한 값이다.

본 논문에서는 기존 연구와 다르게 카메라로부터 입력받은 PDF417 바코드 이미지에서 바코드 영역을 추출한 후에 이미지에 따른 최소 모듈과 Row, Column의 정보를 찾을 수 있도록 하며, Start 패턴의 오른쪽에 있는 Indicator부분을 해석하지 않고 데이터 코드워드만 추출하여 디코딩 하는 알고리즘을 제안한다.

3.2 제안하는 PDF417 디코딩 알고리즘

PDF417의 이미지만 추출한다고 가정하고, 다음은 PDF417 바코드의 데이터를 해석하는 단계를 제시한다.

단계1: 전처리 및 영상인식

먼저 'Start & Stop Pattern 및 Row Indicator 부분 제거'를 진행하기 전에 바코드의 이미지를 이진화하고 왜곡된 부분을 전처리 단계를 적용하였다고 가정하고 영상인식 처리과정은 다음 단계를 적용한다.

Algorithm, of image processing in PDF417
Step1: Receive input image;
Step2: Convert color image into gray image;
Step3: Convert the gray image into binary image;
Step4: Apply filter to the binary image;
Step5: Localize barcode image;
Step6: Rectify the barcode image;

Step1은 카메라로부터 입력 이미지를 수신하고 Step2에서 컬러 이미지를 그레이 이미지로 변환한다. Step3에서 그레이 이미지를 이진 이미지로 변환하고, Step4에서 이진 이미지에 필터를 적용한다. 이후 Step5는 바코드 이미지 추출한다. 하지만 추출한 바코드 이미지가 카메라 기반이므로 왜곡(회전 및 곡선)되어 들어온다. 따라서 Step6에서는 [Fig. 3]과 같이 왜곡된 바코드 이미지의 왜곡을 바로잡는다.

[Fig. 3] Rectification of Distorted Barcode image:
(a)Distortion of PDF417 Image (b)Rectification of PDF417 image

단계2: Start & Stop Pattern 및 Row Indicator 부분 제거

두 번째 단계는 입력된 바코드 이미지에서 Start & Stop Pattern 및 Row Indicator 부분을 제거하는 것으로 세부적인 처리과정은 [Fig. 4]와 같다.

```
int getCodeword(int x) //x=Codeword value
Begin
1:    if (The first symbol is a start pattern?) then
2:       Move to the next Symbol;
3:       if (The next symbol is a left indicator?) then
4:          Move to the next Symbol;
5:       endif;
6:    endif;
7:    int num[] = new int[8];
8:    num[0] = x / 10000000;
9:    num[1] = (x / 1000000) % 10;
10:   num[2] = (x / 100000) % 10;
11:   num[3] = (x / 10000) % 10;
12:   num[4] = (x / 1000) % 10;
13:   num[5] = (x / 100) % 10;
14:   num[6] = (x / 10) % 10;
15:   num[7] = x % 10;
16:   int clusternumber = (num[0] - num[2]
17:      + num[4] - num[6] + 9) % 9;
18:   int i = clusternumber / 3;
19:   for j∈(0, 1, ..., symboltable.length) do
20:     if (x == symboltable[i][j]) then
21:       return j;
22:     endif;
23:   end for;
24:   return -1;
End
```

[Fig. 4] Algorithm of removing the start and stop patterns, and row indicator from bar code

Start 패턴 심볼값을 찾으면(Step2) 다음 심볼값은 Indicator 값이며 이 부분은 저장하지 않고 다음 심볼값으로 넘어간다.(Steps3-4) 즉 입력 심볼에서 Start & Stop Patterns 2개와 Indicator 2개 패턴을 식별한 후, 데이터 코드워드만을 저장하여 해석한다.(Steps1-6) 그리고 steps7-11은 입력 심볼값 x에 대한 클러스터 값을 결정한다.(steps7-18) Steps19-23은 입력 심볼값 x에 대한 symboltable[i][j]의 값을 결정하여 반환한다.(steps19-23)

[Fig. 4]의 알고리즘과 같이 Start & Stop Pattern 및 Row Indicator 이외의 부분인 심볼값 x의 데이터 코드워드를 추출한다. 여기서 Indicator 부분을 생략하지만, 전처리 단계에서 Indicator 부분의 정보를 미리 알 수 있기 때문에 Indicator 부분 해석을 생략한다.

기존의 데이터 코드워드 디코딩 방식은 심볼값을 0~928 사이 값으로 변환시켜 그 값에 따라 데이터를 해석하지만, 본 논문에서는 미리 클러스터 테이블을 정의하여 계산된 심볼값을 별도로 저장하는 메모리 저장시간을 줄이는 장점이 있다.

단계3: 데이터 코드워드 디코딩

위에서 설명한 표현양식 Mode는 [Fig. 5]와 같이 데이터 코드워드의 두 번째 값에 따라 900이하의 값은 Text Mode로, 901과 924는 Byte Mode로, 902는 Numeric Mode로 각각 디코딩된다.

```
void checkIndicator()
Begin
1:    for i∈(1,2, ... , (Datacodeword.length−1)) do
2:      switch (DataCodeword[i] == 900) do
3:      case 901: // byte mode
4:      case 924: // byte mode
5:          call byteMode(next.DataCodeword[i]);
6:          break;
7:       case 902: // nummeric mode
8:          call numericMode(next.DataCodeword[i]);
9:          break;
10:      case 900: // text mode
11:      default:
12::          call textMode(Datacodeword[i]);
13:      endswitch
14:   endfor
End
```

[Fig. 5] Algorithm of checking the indicator from data codeword

[Fig. 5]에서 저장된 데이터 코드워드에서 첫 번째 읽은 모드값이 901이나 924이면 바이트 모드(byte mode)의 코드워드 처리모듈을 호출하고,(steps3-6), 읽은 모드값이 902이면 숫자모드(numeric mode)의 코드워드 처리모듈을 호출하고,(steps7-9), 모드값이 900이나 이외의 다른 모드 값이면 택스트 모드(text mode)로 해석하여 처리한다. (steps10-12)

3.2.1 텍스트 모드

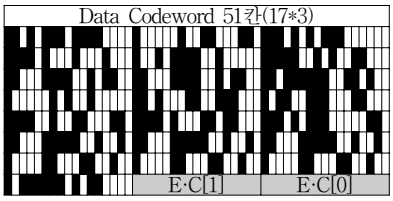
텍스트 모드는 PDF417의 기본 양식으로 900 지시자를 사용한다. Text mode는 Alpha, Lower, Mixed, Punctuation의 4개의 하부 양식으로 나누어지는데 기본 하부 양식모드로 Alpha(영문 대문자 양식)가 쓰이며, 코드워드 하나에 2개의 ASCII 문자를 표현한다. 코드워드 값을 구하고 하부 양식의 문자들은 0에서 29 사이의 값으로 할당받는다. 텍스트 모드의 디코딩 알고리즘은 [Fig. 6]과 같다.

```
int TextMode()
Begin
1:    if (DataCodeword ≧ 900) then
2:       ClusterValue = 900;
3:       return  - 1;
4:    endif
5:    TextCodeword[2*ClusterValue]
                        := DataCodeword/30;
6:    TextCodeword[2*ClusterValue +1]
                        :=DataCodeword%30;
7:    sClusterValue++;
8:    if (TextCodeword[0] ≦ 26) then
9:       call alpha();
10:   else then
11:      int y[] = new int[text.length  - 1];
12:      for j∈(0, 1, ..., y.length) do
13:         y[j] = text[j + 1];
14:      endfor
15:      if (TextCodeword[0] == 27) then
16:         call lower(y); //Move to lower area.
17:      else if (TextCodeword[0] == 28) then
18:         call mixed(y);//Move to mixed area.
19:      else if (TextCodeword[0] == 29) then
20:         call punctuation(y);//punctuation area
21:      endif
22:   endif
23:   return 0;
End
```

[Fig. 6] Algorithm of decoding in text mode

900이상의 값은 제어 값이므로 cluster value를 초기화 시키고 빈 공간으로 간주한다.(steps1~4) 텍스트 모드는 1개의 코드워드로 2개의 문자 값을 가지는데 30으로 나누었을 때 몫과 나머지 값으로 가지게 된다(steps5~6). 그리고 기본적으로 Alpha Mode로 해석(steps8~9)되며 30이하의 값 중에서 27, 28, 29는 하부모드로 변경하는 제어 값으로 사용하고 하부 양식에서 코드워드를 해석한다(steps10~15).

다음은 텍스트 모드의 코드워드에 대한 디코딩의 예이다. 코드워드가 아래 표와 같이 '21113144 15122321 31111136 ... 11611124'의 값이 입력될 때, 각 클러스터 시퀀스마다 클러스터 값(cluster value)가 계산된다.



Cluster Sequence	ClusterValue	Cluster Sequence	Cluster Value	Cluster Sequence	Cluster Value
21113144	22	21622112	849	23521211	900
15122321	841	15111215	806	23521211	900
31111136	0	21132233	88	14232212	900
41131313	126	11223323	271	14232212	900
12114215	249	42122321	298	14232212	900
52111223	214	41132123	131	11611124	900
23421122	806	31113521	39	E·C	
23421122	806	23521211	900	E·C	

위 표에서 각 클러스터 값은 식(1)에 따라 각 코드워드에 대한 값(Value)으로 몫(H)과 나머지(L)가 계산된다.(900 이후의 값은 생략)

$$Value = 30*H + L \qquad (식1)$$

Cluster Value	Value		Cluster Value	Value	
22	22		849 = 28 * 30 + 9	28	9
841 = 28 * 30 + 1	28	1	806 = 26 * 30 + 26	26	26
0 = 0 * 30 + 0	0	0	88 = 2 * 30 + 28	2	28
126 = 4 * 30 + 6	4	6	271 = 9 * 30 + 1	9	1
249 = 8 * 30 + 9	8	9	298 = 9 * 30 + 28	9	28
214 = 7 * 30 + 4	7	4	131 = 4 * 30 + 11	4	11
806 = 26 * 30 + 26	26	26	39 = 1 * 30 + 9	1	9
806 = 26 * 30 + 26	26	26	900	900	

앞에서 계산된 H, L 값들을 이용하여 TC하부양식구조에서 각 Value에 대응하는 Char 값을 찾으면 다음 표와 같다.

Value		Char		Value		Char		Value	Char
22		22		28	9	al,	J	900	
28	1	ml	1	26	26	Space	Space	900	
0	0	0,	0	2	28	C	ml	900	
4	6	4	6	9	1	9	1	900	
8	9	8	9	9	28	9	al	900	
7	4	7,	4	4	11	E	L	900	
26	26	Space	Space	1	9	B	J		
26	26	Space	Space	900					

최종적으로 입력된 코드워드는 위의 표에서 계산된 Char 값을 위에서 아래로 순서대로 읽으면서 '100468974 J C919ELBJ'의 코드 정보를 얻는다.

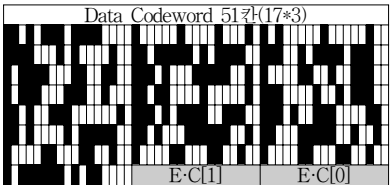
3.2.2 바이트 모드

Byte Compaction Mode는 코드워드 5개에 문자 6개를 표현하기 때문에 다른 모드에 비해 많은 코드워드가 사용되지만, 표현할 수 있는 문자의 폭이 넓다는 특징을 가진다. 바이트 모드는 256개의 국제 문자를 표현하는데 사용되며, 우선 표현하는 문자 개수를 알아야 한다. 바이트 모드로 넘어가는 것은 2종류의 양식인 901, 924가 있으며, 901은 코딩되는 문자 개수가 6의 배수가 아닐 경우이고 924는 코딩되는 문자 개수가 6의 배수 일 경우이다.

```
int ByteMode()
Begin
 1: if (DataCodeword ≧ 901 or 924) then
 2:     ClusterValue = 900;
 3:     return -1;
 5: endif
 6: if (DataCodeword[1] == 901) then
 7:     for i∈(0,1, ..., (DataCodeword.length/6)) do
 8:         for j∈(0,1, ..., 5) do
 9:             Add (DataCodeword[(i * 6) + j] * 900 +
                DataCodeword[(i*6)+j+1]) to
                temp10Codeword
10:         endfor
11:     endfor
12: endif
13: else if (DataCodeword[1] == 924) then
14:     for i∈(0, 1,..., (DataCodeword.length/6)*5) do
15:         add (temp10Codeword % 256) to
                temp256Codeword[i];
16:     endfor
17: endif
18: return 0;
End
```

[Fig. 7] Algorithm of decoding in byte mode

[Fig. 7]에서 바이트 모드는 6개의 데이터 코드워드 값으로 5개의 문자를 표현하므로 900진법(steps5~9)인 코드워드 값을 256진법으로 변경(steps10~12)하여 5개의 값을 계산한다.



Data Code words	Cluster Sequence	Cluster Value	Data Code words	Cluster Sequence	Cluster Value
D[0]	21113144	22	D[12]	21123611	101
D[1]	15141311	901	D[13]	31114232	46
D[2]	11151512	134	D[14]	31123421	99
D[3]	43111412	461	D[15]	21141152	111
D[4]	21511142	860	D[16]	21136211	109
D[5]	32114132	248	D[17]	23521211	900
D[6]	11432222	599	D[18]	14232212	900
D[7]	31135211	107	D[19]	14232212	900
D[8]	14221232	648	D[20]	14232212	900
D[9]	11333213	573	D[21]	11611124	900
D[10]	21241511	330	D[22]	E·C	
D[11]	34121132	628	D[23]	E·C	

따라서 바이트 모드는 다음과 같이 클러스터 값(Cluster Value) 5개를 가지고 6개의 문자정보를 얻을 수 있다. 또한 다음의 5개 클러스터 값에 대해 아래의 반복적인 계산과정을 통해 최종적인 코드 정보를 얻을 수 있다.

$$(((134\times900+461)\times900+860)\times900+248)\times900+599 = 88,254,165,823,799$$

$88,254,165,823,799 \;/\; 256^5 = 80 + 293,235,601,719$

$293,235,601,719 \;/\; 256^4 = 68 + 1,177,825,591$

$1,177,825,591 \;/\; 256^3 = 70 + 3,420,471$

$3,420,471 \;/\; 256^2 = 52 + 12,599$

$12,599 \;/\; 256^1 = 49 + 55$

$55 \;/\; 256^0 = 55 + 0$

나머지 5개의 데이터 코드워드 값 자체가 256진법 값과 같으므로 ASCII 테이블에서 찾으면 다음과 같다.

Value	Char	Value	Char	Value	Char
80	P	71	G	99	c
68	D	111	o	111	o
70	F	111	o	109	m
52	4	103	g	900	
49	1	108	l	900	
55	7	101	e	900	
64	@	46	.	900	

위의 표와 같이 순서대로 문자를 읽으면 입력코드에 대한 최종적인 'PDF417@Google.com'의 코드정보을 읽을 수 있다

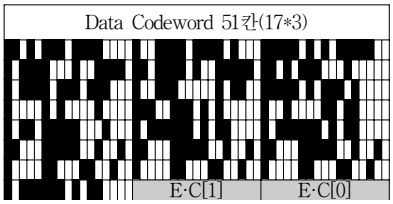
3.2.3 숫자 모드

숫자모드(numeric mode)는 902 지시자 코드워드가 사용되며, 1개의 코드워드에 2.95개의 문자를 표현할 수 있어 코드워드 개수를 적게 사용한다.

```
int Numeric()
Begin
 1: if (DataCodeword ≧ 902) then
 2:     ClusterValue = 900;
 3:     return -1;
 4: endif
 5: for i∈(0,1,...,DataCodeword.length) do
 6:     Add (DataCodeword[i]*900 +
          DataCodeword[i+1]) to DataCodeword[i+1];
 7: endfor
 8: for i∈(0,1,...,(DataCodeword.length−1)) do
 9:     Add (DataCodeword[dataCodeword.length].At(i+1))
          to Text[i];
10: endfor
11: return 0;
End
```

[Fig. 8] Algorithm of decoding in numeric mode

[Fig. 8]은 숫자 모드 디코딩 알고리즘으로서 숫자 모드의 특징인 데이터 코드워드 한 개에 최대 2.95개의 숫자를 저장하는 방식으로 900진법의 값을 10진법으로 누적시켜 변경(steps5~7)시켜주어 맨 앞자리 숫자를 제거해줌(steps7~10)으로서 정보를 해석할 수 있다.



Cluster Sequence	Cluster Value	Cluster Sequence	Cluster Value	Cluster Sequence	Cluster Value
21113144	22	51113114	22	23521211	900
12414113	901	14111414	598	23521211	900
51111152	2	32311115	655	14232212	900
11431241	799	12142124	310	14232212	900
21411251	765	11215313	327	14232212	900
21113315	33	11611124	900	11611124	900
11413133	548	11611124	900	E·C	
12351122	611	23521211	900	E·C	

이러한 값들은 900진법 값이므로 다시 10진법으로 변환된다.

2 * 900 + 799 = 2,599
2,599 * 900 + 765 = 2,339,865
2,339,865 * 900 + 33 = 2,105,878,533
2,105,878,533 * 900 + 548 = 1,895,290,680,248
1,895,290,680,248 * 900 + 611 = 1,705,761,612,223,811
1,705,761,612,223,811 * 900 + 22 = 1,535,185,451,001,429,922
1,535,185,451,001,429,922 * 900 + 598 =
1,381,666,905,901,286,930,398
1,381,666,905,901,286,930,398 * 900 + 655 =
1,243,500,215,311,158,237,358,855
1,243,500,215,311,158,237,358,855 * 900 + 310 =
1,119,150,193,780,042,413,622,969,810
1,119,150,193,780,042,413,622,969,810 * 900 + 327 =
1,007,235,174,402,038,172,260,672,829,327

10진법 값 1,007,235,174,402,038,172,260,672,829,327에서 맨 앞자리 1을 지움으로서 읽고자하는 숫자 '00723 51744 02038 17226 06728 29327' 코드정보를 읽을 수 있다.

본 논문에서는 오류 복원 코드워드 디코딩을 1단계에서 데이터 코드워드 오염이 검출될 경우 데이터 코드워드부분을 해석하지 않고 오류 복원 코드워드를 해석함으로서 불필요한 해석부분을 줄인다. 오류 복원 코드워드 해석은 기존 방식과 같이 Reed-Solomon[12][13]에 따른 데이터 값을 복원하고 해석하면 정보를 얻을 수 있다.

4. 기존연구와의 비교검토

기존 연구의 공통점은 AIM 표준 규격에 따라 데이터를 디코딩한다. AIM 표준 규격에 따른 디코딩 절차는 Start & Stop 패턴과 Indicator와 같은 디코딩 하지 않아도 될 부분까지 디코딩을 하게 된다. 기존 연구 중 정정구 외(2002)의 경우 바-스페이스 패턴을 추출 후에 Start & Stop 패턴과 Indicator 부분까지 데이터로 변환한 후 AIM 표준 규격에 따라 디코딩한다. Jayasuriya et al.(2007) 연구는 정정구(2002)의 연구와 유사하지만, 바-스페이스 패턴 추출 후에 오류 복원 코드워드와 데이터 코드워드를 비교하므로 코드 정보는 정확하지만 불필요한 과정까지 포함하며 두 가지 디코딩을 하므로 디코딩 속도가 다소 느리다는 단점이 있다. Song et al.(2013)의 Bit Shifting decoding algorithm은 이미지 기술로 디코딩하기 때문에 이미지가 정확할 경우 100% 정확한 정보를 보여주지만 이미지가 손실되어 있는 경우 디코딩된 바코드 이미지의 정보가 손실발생에도 불구하고 잘못된 정보를 인식한다는 문제점이 있다.

따라서 기존 연구와 비교한다면 기존 디코딩 알고리즘은 위와 같이 Start & Stop 패턴과 Indicator 부분의 모든 정보를 디코딩하지만 제안하는 알고리즘은 1단계 과정에서 Indicator부분의 해석을 좀 더 간소화하고 데이터 코드워드 디코딩 알고리즘을 더 개선함으로서 시간을 충분히 단축할 수 있는 장점이 있다.

5. 결론

본 논문에서는 PDF417 바코드의 디코딩 과정을 기존의 방법들보다 빠르게 디코딩 할 수 있는 기술을 제안했다. 기존에 제시된 알고리즘들은 바-스페이스 패턴을 추출하고 Start & Stop 패턴과 Indicator, Codeword등 을 모두 데이터로 인식하고 그 데이터를 디코딩을 하게 된다. 실제 이미지가 들어와서 불필요한 부분까지 데이터로 사용하여 디코딩을 하는 과정이 포함된 반면에 제안하는 알고리즘은 불필요한 과정인 Start & Stop 패턴과 Indicator 제거하고 실질적으로 바코드 내부에서 데이터가 담긴 데이터 코드워드만 추출하여 디코딩하므로 디코딩 시간을 단축을 할 수 있다. 또한, 오류복원 코드워드와

데이터 코드워드를 비교해서 결과를 보여주는 알고리즘은 정확도가 개선되었지만, 오류복원 코드워드를 디코딩하고, 코드워드를 디코딩하고 나온 값들을 비교하는 과정이 디코딩 시간을 더 오래 걸리게 한다.

본 논문에서는 기존의 불필요한 작업을 하는 알고리즘에서 디코딩이 필요없는 부분을 확실히 제거함으로써 빠른 결과를 얻을 수 있는 알고리즘을 제안하며, 더 정확한 결과를 위한 알고리즘을 제시했다는 데 의의가 있다고 하겠다.

앞으로의 연구방향은 PDF4176 디코딩 알고리즘 구현과 더불어 알고리즘의 구현, 카메라 기반의 바코드 이미지를 인식하여 디코딩하는 부분과의 연동, 바코드 인쇄기와의 연동을 통한 in-line 바코드 검증기의 구현이 필요할 것이다. 끝으로 이외도 다른 바코드 검증기와의 비교 실험을 통한 성능 우수성의 검토가 필요하리라 사료된다.

ACKNOWLEDGMENTS

This research was supported by the MSIP(Ministry of Science, ICT and Future planning), Korea, under the Specialized Co-operation between industry and acadmic support program (NIPA-2013010918) supervised by the NIPA(National IT Industry Promotion Agency)

REFERENCES

[1] Hee Il Hahn and Joung Koo Joung, "Implementation of Algorithm to Decode Two-Dimensional Barcode PDF-417", Signal Processing, 2002 6th International Conference on, Vol. 2, pp. 1791-1794, 2002

[2] Tharindu Jayasuriya An Image Encoding/Decoding Algorithm for PDF417 Image, University of Moratuwa, 2007

[3] Qian Song and Li Liu, "Decoding of PDF417 Barcode in Identity Authentication based on LabVIEW," iaesjournal, Vol. 11, No. 6, June 2013, pp. 3005 ~ 3011

[4] AIM USA, "Uniform Symbology Specification PDF417", http://www.aimusa.org, 2009

[5] Young Kyu Choi, "A 2-D Barcode Detection Algorithm based on Local Binary Patterns", Journal of the Semiconductor & Display Equipment Technology, Vol. 8, No. 2. June 2009.

[6] Zhijun D. "The Reading and Recognition Technology of PDF417 2D Barcode" Master Thesis, Zhongnan University ; 2009.

[7] Silver Bay Software LLC, PDF417 Encoder Programmer's Manual, 2012 .

[8] Zxing, "Multi-format 1D/2D barcode image processing library with clients for Android Java," https://code.google.com/p/zxing/. 2013

[9] Fengmei L. "PDF417 Barcode and Its Decoding", Computer Development and Application, Vol 15, No. 6, pp.17~19, 2002;

[10] Ho Geun Oh, Latest Barcode Application Technology, Seong Ahn Dang, 1997

[11] Asial Corporation, JpGraph Manual : Chapter 25. PDF417 (2D-Barcode), http://jpgraph.net, 2010

[12] Reed, I. S. and Solomon, G., "Polynomial Codes Over Certain Finite Fields," SIAM Journal of Applied Math., vol. 8, pp. 300-304, 1960

[13] Bernard Sklar, Digital Communications: Fundamentals and Applications, Second Edition: Reed-solomon Codes, Prentice-Hall, 2001

[14] GS1, GS1 General Speciffications Version 13, pp.350-372, January 2013.

[15] Kriste Krstovs, "Driver's License Scanner Pocket PC Application", Technical Report ECE., P54, 2002

김 영 중(Kim, Young Jung)

- 2012년 6월 : 충주대학교 항공·기계설계학과(공학사)
- 2012년 9월 ~ 현재 : 충북대학교 대학원 디지털정보융합학과 석사과정
- 관심분야 : 데이터베이스 시스템, 과학 데이터베이스, 바코드 검증기
- E-Mail : rex@chungbuk.ac.kr

조 영 민(Cho, Young Min)

- 2008년 3월 ~ 현재 : 충북대학교 컴퓨터교육과 학사과정
- 관심분야 : 데이터베이스시스템, 바코드 검증기
- E-Mail : mouse0944@naver.com

이 종 연(Lee, Jong Yun)

- 1985년 2월 : 충북대학교 전자계산공학과(공학사)
- 1987년 2월 : 충북대학교 대학원 전자계산기공학과(공학석사)
- 1999년 2월 : 충북대학교 대학원 전자계산학과(이학박사)
- 1990.2 ~ 1996.5 : 현대전자산업(주) SW연구소 및 현대정보기술(주) CIM사업부 근무(책임연구원)
- 1999.3 ~ 2003.3 : 강원대학교 삼척캠퍼스 정보통신공학과 조교수
- 2003.3 ~ 현재 : 충북대학교 컴퓨터교육과 및 디지털정보융합학과 교수로 재직 중
- 관심분야 : 질의 처리 및 최적화, 시공간 데이터베이스, 과학 데이터베이스, e-learnimg 및 평가모델, 유통물류, GIS
- E-Mail : jongyun@chungbuk.ac.kr