Data Matrix 이차원 바코드의 디코딩 알고리즘의 구현

황진희, 한희일
한국외국어대학교 산업공과대학 정보통신공학과

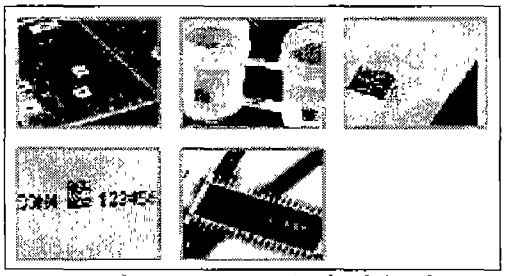
Jinhee Hwang, Hee-il Hahn
Department of Information and Communications Engineering, Hankuk University of Foreign Studies
jhhwang@spice.hufs.ac.kr

요 약

2 차원 바코드는 2 차원 점자식 코드로서 낮은 공간점유, 높은 정보, 다양한 정보처리 기능이 가능한 차세대 라벨링 기법이다. 즉, 2 차원(2D) 심볼로지는 양축(X 방향, Y 방향)으로 데이터를 배열시켜 평면화 시킨것으로서 기존의 일차원(1D) 바코드 심볼로지가 가지는 문제점인 데이터 표현의 제한성, 즉 선적용 패키지와 같은 로트 번호, 구매 주문 번호, 수취자, 수량 기타 정보 등의 다양한 내용을 바코드로 표현하여 대상물에 부착하거나 동반시킴으로써 1980 년대 중반에 등장하게 되었고, 현재 많은 부분에서 사용하고 있다. 본 논문에서는 현재 많이 쓰이는 2 차원 바코드 중 하나인 Data Matrix 의 구성과 디코딩 알고리즘을 제안한다. Data Matrix 는 데이터를 효율적으로 나타내기 위하여 각 정보의 표현에 따라 다른 인코딩 방식을 사용하고 있다. 디코딩 알고리즘은 그에 따라서 구현되었다.

1. 서 론

Data Matrix는 1989년 미국 International Data Matrix사의 사장인 Dennis Priddy에 의해 개발된 매트릭스형 코드이다. Data Code라고도 불리는 이 심볼로지는 심볼당 표현할 수 있는 데이터의 양을 강조하고 있다. 심볼 크기는 한 변이 0.001 인치에서 14 인치까지 가능하며, 심볼당 최대 2334 개의 Alphanumeric 문자나 도트 매트릭스 프린터로 500 개의 수치를 표현할 때는 1 인치 정사각형에 가능하고 500개의 모든 ASCII 문자를 표현할 때는 1.4 인치 정사각형에 가능하다.[1][3] <그림 1>은 Data Matrix의 한 응용 예로 물품라벨로 이용하는 것을 볼 수 있다.



<그림 1> Data Matrix의 사용 예

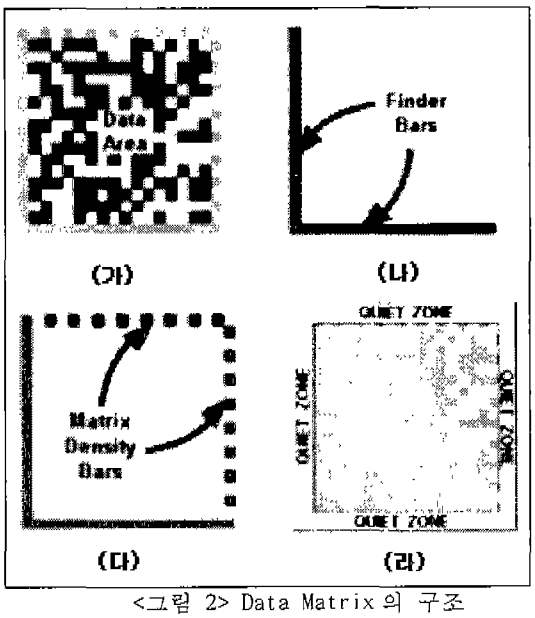
Data Matrix는 오류 검출 및 수정(error checking and correction) 알고리즘에 따라서 ECC 00-140 과 ECC 200 두 종류의 심볼이 있다. ECC 00 - 140은 Convolutional 오류 검출 및 수정 알고리즘을 채택하고, ECC 200은 Reed-Solomon 알고리즘을 채택하고 있다. 심볼의 인쇄와 판독이 한 회사에서 패쇄적으로 이루어지고 그 이용을 전적으로 책임지는 제한적인 부분을 제외한 대개의 경우 ECC 200을 사용하도록 추천하고 있다. 본 논문에서는 이에 근거하여 ECC 200을 기준으로 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 Data Matrix의 심볼 구조와 분포되어있는 행렬이 가지고 있는 정보, 그리고 코드워드의 구성에 대하여 살펴보고, 3장에서는 본 논문에서 구현한 디코딩 알고리즘에 대하여 기술하며, 4장에서는 결론을 제시하였다.

2. Data Matrix 바코드의 구조

2.1 Data matrix 의 심볼 구조

<그림 2>는 Data Matrix 의 각 심볼의 구조를 나타내고 있다. 이 심볼 구조는 정규적으로 배열된 정사각형 모듈들의 집합을 포함하고 있다. ECC 200 심볼인 경 우 데이터 영역들은 정렬 패턴(alignment pattern)에 의해 분리되어 있다. 데이터 영역은 각각의 인식 패턴으로 둘러싸여 있으며 이 인식패턴은 사방이 빈 여백(quiet zone)으로 둘러 싸여 있다. <그림 2>의 (나)(다)의 인식

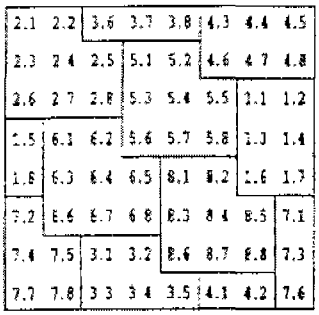


<그림 2> Data Matrix의 구조
데이터 영역 : (가) Data Area
인식패턴 : (나) Finder Bars
(다) Density Bars (라) Quiet Zone

패턴은 데이터 영역을 둘러싸고 있는 1 모듈 굵기의 외곽선이다. (나)의 L 자 모양의 인식 패턴은 리더기가 바코드의 위치와 방향을 인식하도록 해준다. (다)의 인접한 2 개의 변은 굵은 검은 선들은 기본적으로 검은 모듈과 흰 모듈이 교대로 나타나는 점선이며, 심볼의 셀 구조(cell structure)를 결정하는데 이용되고, 또한 심볼의 크기와 뒤틀림을 결정하는데도 도움을 주고 있다. (라)의 빈 여백은 인식 패턴을 둘러싸고 있는 스페이스 영역으로 최소 1 모듈 이상의 폭을 가져야 한다. [1][2][4]

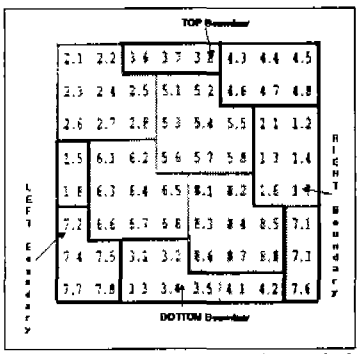
2.2 행렬이 가지고 있는 데이터의 정보

<그림 3>은 <그림 2>에서 언급한, 인식 패턴과 여백을 제외한 데이터 영역 부분으로 심볼의 크기가 10*10이고, 데이터 행렬이 8*8인 행렬이다.



<그림 3> 데이터 영역의 랜덤한 분포

하나의 모듈은 0 과 1 의 이진수로 표현이 된다. <그림 3>에서 보듯이 데이터 부분은 8 개의 모듈이 한 개의 단위 심볼 문자로 만들어 지고, 각 심볼 문자는 0 ~ 256 사이의 값을 가지게 된다. 이 값을 코드워드(codeword)라고 한다. 이러한 분포는 데이터의 양에 따라서 서로 다른 모양을 가지게 된다. 이러한 행렬의 구성은 데이터의 크기에 따라 쓰이는 Corner Condition 을 다르게 한다. <그림 4>는 <그림 3>을 기준으로 랜덤하게 만들어지는 코너의 모양을 보여주고 있다.



<그림 4> 랜덤한 구성의 행렬을 위한 기본 corner condition

<표 1>은 행렬의 크기에 따른 코드워드의 수와 오류 데이터의 수를 나타낸다. ECC 200 은 <표 1>에서 보듯이 심볼을 정사각형과 직사각형의 두 가지로 나누어 만들게 된다. 정사각형과 직사각형 심볼에서 가장 기초적인 것은 데이터 영역의 수가 하나만 있는 것이고, 가장 복잡한 경우는 정사각형 심볼의 최대 36 개까지 데이터의 영역을 나누어 놓은 144*144 의 행렬이다. Data Matrix 의 경우는 엄청난 수의 문자를 표현하였을 때 패턴의 인식을 쉽게하기 위하여 일정한 문자 이상의 데이터를 하나의 영역으로 나누어 놓았다. [5]

<표 1> ECC 200 이 가지는 데이터의 수

Symbol Size Row	Symbol Size col	Data Region Size	Data Region Number	Mapping Matrix Size	Total Codewords Data	Total Codewords Error	Reed-Solomon Block Data	Reed-Solomon Block Error
정사각형 심볼								
10	10	8*8	1	8*8	3	5	3	5
12	12	10*10	1	10*10	5	7	5	7
14	14	12*12	1	12*12	8	10	8	10
16	16	14*14	1	14*14	12	12	12	12
18	18	16*16	1	16*16	18	14	18	14
20	20	18*18	1	18*18	22	18	22	18
22	22	20*20	1	20*20	30	20	30	20
24	24	22*22	1	22*22	36	24	36	24
26	26	24*24	1	24*24	44	28	44	28
32	32	14*14	4	28*28	62	36	62	36
36	36	16*16	4	32*32	86	42	86	42
40	40	18*18	4	36*36	114	48	114	48
44	44	20*20	4	40*40	114	56	144	56
48	48	22*22	4	44*44	174	68	174	68
52	52	24*24	4	48*48	204	84	102	42
64	64	14*14	16	56*56	280	112	140	56
72	72	16*16	16	64*64	368	144	92	36
80	80	18*18	16	72*72	456	192	114	48
88	88	20*20	16	80*80	576	224	144	56
96	96	22*22	16	88*88	696	272	174	68
104	104	24*24	16	96*96	816	336	136	56
120	120	18*18	36	108*108	1050	408	175	68
132	132	20*20	36	120*120	1304	496	163	62
144	144	22*22	36	132*132	1558	620	156	62
							155	62
직사각형 심볼								
8	18	6*16	1	6*16	5	7	5	7
8	32	6*14	2	6*28	10	11	10	11
12	26	10*24	1	10*24	16	14	16	14
12	36	10*16	2	10*32	22	18	22	18
16	36	14*16	2	14*32	32	24	32	24
16	48	14*22	2	14*44	49	28	49	28

2.3 코드워드의 표현

Data Matrix는 아래의 <표 2>와 같이 15가지의 코드워드로 표현된다.

<표 2> 코드워드를 표현하는 정보

Codeword	Data or Function
1-128	ASCII data(ASCII value + 1)
129	Pad
130-229	2-digit data 00-99 (Numeric Value + 130)
230	Latch to C40 Encodation
231	Latch to Base 256 Encodation
232	FNC1
233	Structured Append
234	Reader Programming
235	Upper Shift (shift to Extended ASCII)
236	05 Macro
237	06 Macro
238	Latch to ANSI X12 Encodation
239	Latch to Text Encodation
240	Latch to EDIFACT Encodation
241	ECI Character

많은 양의 데이터를 최소한의 코드워드로 만들기 위하여 상황에 따라 그 쓰임새가 다르게 나타난다. 이 15가지 코드워드 중 가장 많이 쓰이는 것이 ASCII 데이터, Pad, 수치 데이터, Latch to C40, Latch to Text 그리고 Latch to EDIFACT이다. 각각을 설명하면 다음과 같다.

2.3.1 ASCII 데이터 코드워드

인코딩된 데이터의 개수가 5개가 넘지 않는 경우에 사용된다. 각 글자 하나하나가 나타내는 ASCII 값에 1을 더한 값으로 표현된다.

코드워드 값 = ASCII 값 + 1

2.3.2 Pad

Pad 값은 나타날 수도 있고 없을 수도 있다. 만약, 이 값이 나타난다면 그 전까지의 값이 실제 데이터의 코드워드이고, 없다면 모든 값이 데이터 코드워드가 된다.

2.3.3 수치 데이터

숫자 하나는 4 비트로 표현이 되므로, 8 비트로 표현하기 위하여 00-99의 값을 사용하게 되고, 다음의 식과 같이 계산을 하게 된다.

코드워드 값 = (00-99) + 130

2.3.4 Latch to C40

C40 함수는 코드워드를 40의 배수로 표현한 것으로 알파벳의 대문자를 위주로 나타내며, 3개의 데이터를 2개의 코드워드로 바꾼다. 다음의 식을 이용하게 된다.

$$(1600 \times C_1) + (40 \times C_2) + C_3 + 1$$

C_1은 첫번째 문자, C_2는 두 번째, C_3는 세 번째 문자의 C40 인덱스 값을 가지게 되고, <그림 5>는 C40 코드워드를 만드는 예이다.

Data Characters	AIM
C40 Values	14, 22, 26
Calculate 16-bit value	(1600 * 14) + (40 * 22) + 26 + 1 = 23307
1st Codeword (16-bit value) div 256	23307 div 256 = 91
2nd Codeword : (16-bit value) mod 256	23307 mod 256 = 11
Codewords	91, 11

<그림 5> C40 코드워드를 만드는 예

2.3.5 Latch to Text

이 코드워드는 C40을 만드는 것과 같은 방법을 사용한다. 그러나 C40과는 달리 대문자와 소문자의 위치가 바뀌어 있어 데이터 변환을 할 때, 차이가 나게 된다.

2.3.6 Latch to EDIFACT

Latch to C40이나 Text는 3개의 문자를 2개의 코드워드로 만들었다. 그러나 Latch to EDIFACT는 4개의 문자를 3개의 코드워드로 만들게 된다. <그림 6>은 ASCII 문자를 EDIFACT 문자로 표현하는 방법을 설명한 것이고, <그림 7>은 그 예를 보여주고 있다.

Data Charater	ASCII		EDIFACT
	Decimal Value	8-bit Binary Value	Value
A	65	01000001	000001
09	57	00111001	111001

<그림 6> ASCII 문자의 EDIFACT 문자로의 변환

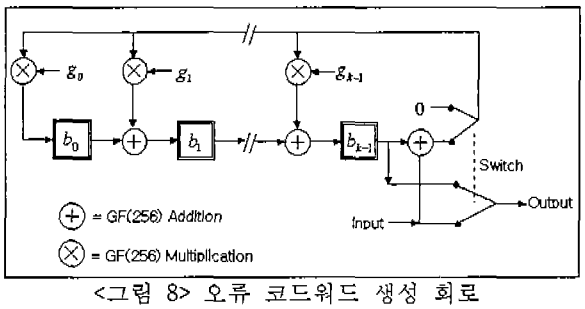
Data characters	D			A			T			A		
Binary values(Annexe J 3)	00	01	00	00	00	01	01	01	00	00	00	01
Divide into 8 8-bit bytes	00	01	00	00	00	01	00	01	00	00	00	01
Codeword values	16			21			1					

< 그림 7> EDIFACT 변환의 예

2.4 오류 검출 및 수정 알고리즘

코드워드를 원래의 데이터로 복원하기 전에 오류가 있는 지를 검사하고 수정하기 위하여 ECC 200은 Reed-Solomon 오류 검출 및 수정 알고리즘을 사용하고 있다. ECC 200은 <표 1>에서 보듯이 데이터 코드워드와 오류 코드워드, 이 두 개의 코드워드로 표현되어 있는 것을 볼 수 있다. <그림 8>

은 에러 코드워드를 만들어내는 회로를 나타내고 있다.

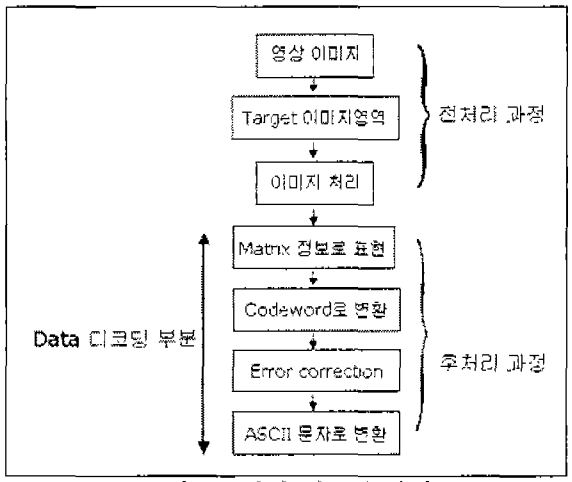


<그림 8> 오류 코드워드 생성 회로

오류 코드워드의 모든 계산은 GF(256)이라는 것을 사용하게 된다. 오류 코드워드의 생성은 스위치의 위치에 의해서 두 가지로 나눌 수가 있다. 스위치가 아래로 향하는 경우는 데이터 입력을 받아들이고, 회로에서 생성된 것과 함께 발생이 되고, 스위치가 위로 향하는 경우 데이터 입력을 0으로 유지하게 된다. Data Matrix에서 사용되는 Reed-Solomon 오류 제어 알고리즘은 일정한 오류 수정 레벨을 가지고 있고, 코드워드를 일정한 블록으로 나누어 오류를 검사하고, 수정하게 된다. [5]

3. Data Matrix 디코딩 알고리즘

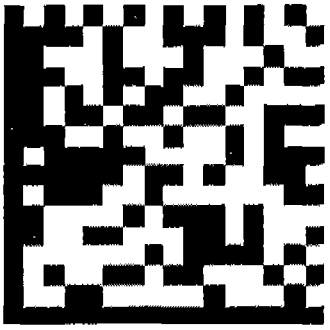
본 논문에서는 바코드 이미지에서 인식 패턴을 제외한 행렬 데이터를 얻었다는 가정을 하고 있다. 따라서, 얻어진 행렬 데이터를 원래의 데이터로 재해석하는 알고리즘을 설명하기로 한다. <그림 9>는 Data Matrix 의 전체 시스템의 블럭도이다.



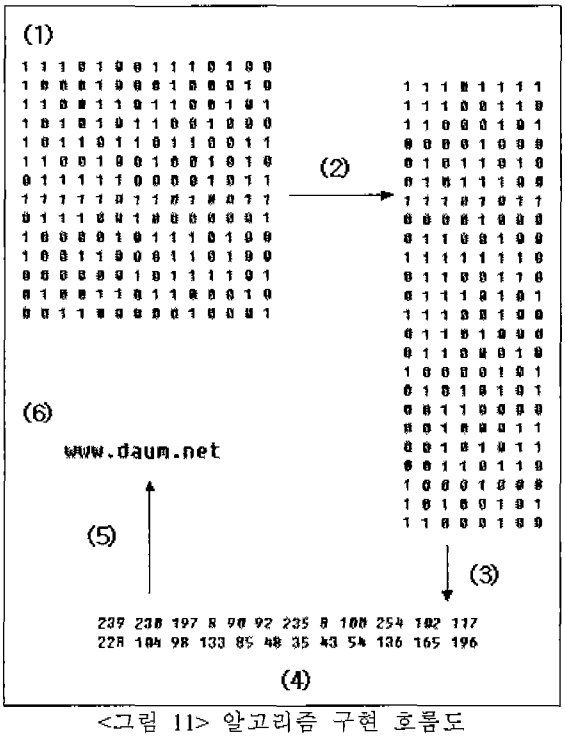
< 그림 9> 전체 시스템 블럭도

Data Matrix 의 디코딩 알고리즘은 <그림 9>의 시스템 블럭도에 설명되어 있는 것과 같다.

1. <그림 10>의 이미지는 16*16 의 이미지 데이터로 각 모듈의 흰 부분은 0, 검은 부분은 1 로 나타내어 14*14 의 행렬 정보로 나타낸 것이 <그림 11>의 (1)이다.
2. (2)는 <그림 11> (1)의 행렬 정보를, 코너 구성 요소들의 배열을 이용하여 8 비트의 모듈 단위로 만드는 과정이다.
3. 8 비트의 단위를 10 진수로 만들어 코드워드로 만들고, 오류 검사와 수정 과정을 거치게 되면 (4)와 같은 데이터 코드 12 개와 에러 코드 12 개로 해석할 수 있다.
4. 코드워드의 해석은 처음 앞의 수가 무엇인가에 따라 15 개의 방법으로 나뉘어 진다고 앞 절에서 설명하였다. <그림 11>의 (4)에 나타난 데이터 코드의 처음은 239 로서 Latch to Text 로 이루어진 것을 알 수 있다.
5. 코드워드를 해석할 때, 데이터 코드워드 부분에서 254 라는 수를 볼 수 있다. 이 수는 Text 모드에서 ASCII 모드로 리턴하는 명령 코드워드이다.



<그림 10> Data Matrix 이미지



<그림 11> 알고리즘 구현 흐름도

6. 2.3.4 절의 <그림 5> 예제를 거꾸로 해석하게 된다.
230 *256 + 197 = 59077

(1) 59077 / 1600 = 36 나머지 : 1477
(2) 1477 / 40 = 36 나머지 : 37
(3) 37 – 1 = 36
Latch to Text 의 인덱스 값에서 36 은 'w'를 나타낸다.
254 의 리턴 명령이 있을 때까지 반복된다.

7. 254 리턴 다음의 나타난 102 는 2.3.1 절에서 설명한것과같이 1 을 뺀 수의 대응되는 문자를 찾으면 된다.
그 값은 102 - 1= 101 로 e 이다.
8. 7 까지의 코드워드 해석이 끝나게 되면 원래 인코딩한 데이터인 'www.daum.net'을 얻을 수 있다.

4. 결론

본 논문에서는 Data Matrix 바코드의 구성과 디코딩 알고리즘에 대하여 알아 보았다. Data Matrix 는 많은 데이터를 함축적이고, 효과적으로 나타내기 위하여 여러 가지 형태의 코드워드 데이터로 바꾸어서 사용한다. 또한, Reed-Solomon 오류제어 알고리즘을 이용하여 코드워드의 분실 및 오류에 대하여 정확한 값을 얻을 수가 있었다.

향후 연구 과제로서 <그림 9>의 시스템 블록도에서 아직 구현이 안된 이미지를 얻어서 행렬변환을 하는 전처리 과정이 이루어져야 하며, 한글 디코딩의 구현도 이루어져야 할 것이다.

참고 문헌

[1] http://www.webcode.co.kr/changwon/education /BARCODE/code-2.htm
[2] 오호근, "최신 바코드 기술 및 응용", 성안당
[3] Information Encoding with Two-Dimensional Barcodes Theo Pavlidis, * Jerome Swartz, and Ynjiun P.Wang Symbol Technolgies
[4] http://www.rvsi.com/acuitycimatrix/index.htm
[5] AIM-USA "International Symbology Specification Data Matrix", Data Matrix Spec