모바일 안드로이드 환경에서의 이차원 바코드 인식

조혜근*, 김인중**
한동대학교 정보통신대학원
e-mail : *usmael@gmail.com, **ijkim@handong.edu

Two Dimensional Barcode Recognition in Mobile Android Environment

Hey-Guen Cho*, In-Jung Kim**
Dept of Information Technology, Handong Global University

요 약

이차원 바코드는 인식이 편리하고 영상의 변이나 손실에 강인하기 때문에 널리 사용되고 있다. 본 논문에서는 모바일 장치에서 카메라로 촬영한 영상으로부터 이차원 바코드의 한 종류인 데이터매트릭스를 검출하고 인식하는 방법을 소개한다. 모바일 안드로이드 환경에서 빠른 속도로 검출 및 인식 과정을 수행하기 위해 JNI를 이용하여 C로 기능을 구현하였다. 먼저, 이차원 영상을 이진화한 후 연결 성분 탐색을 통해 바코드 후보영역을 추출한다. 그리고 직선을 검출하여 데이터매트릭스의 주요소인 L 인식 패턴과 타이밍 패턴을 찾는다. 각 패턴을 이용하여 바코드 영역의 꼭지점을 찾아낸 후, 회전이나 기울어짐을 정규화하여 정사각형 형태로 교정한다. 그 후, 타이밍 패턴을 기준으로 데이터 영역을 인식한다.

1. 서론

스마트폰을 비롯하여 카메라가 장착된 모바일 기기의 보급이 점점 확대되고 있다. 그에 따라 모바일 환경에서 카메라를 통해 입력받은 영상을 인식하는 기술의 응용분야도 넓어지고 있다. 모바일 단말기에서 촬영한 영상은 환경에 의한 변이가 매우 크다. 이차원 바코드는 이러한 환경에서도 인식하기 간편하며 오류 수정 코드를 포함해 영상의 변이나 손실에 강력하기 때문에 모바일 환경에서 사용하기에 적합하다.

본 논문에서는 표준 이차원 바코드 중 하나인 데이터매트릭스를 검출하고 인식하는 과정을 소개한다. 데이터매트릭스는 ISO/IEC 16022 표준으로 제정된 이차원 바코드로써 ASCII 128개 분자를 모두 표시할 수 있으며 약 2,300개의 문자 및 숫자를 극소화된 이미지에 저장할 수 있다. 특히, 리드 솔로몬 오류 정정 코드를 포함하고 있어 최대 20퍼센트의 영상 손실이 있더라도 정보를 판독할 수 있다.

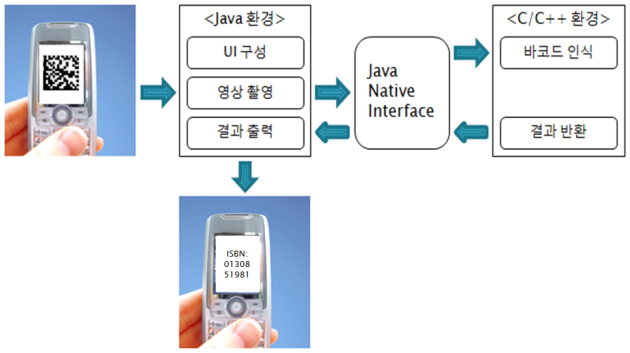
데이터매트릭스는 그림 1과 같이 검은색과 흰색의 정사각형이 일정한 규칙에 의해 배열된 형태를 갖는다. 데이터매트릭스를 이루는 작은 정사각형은 모듈이라고 불린다. 데이터매트릭스에 저장된 정보량에 따라 모듈의 수가 달라진다. 외부에는 모듈 하나 정도의 너비로 흰색의 여백이 있는데 데이터매트릭스와 주변 이미지를 구분해주는 역할을 한다. 좌측과 하단 경계부분에는 검은색 모듈들로 구성된 L 인식 패턴이 있다. L 인식 패턴은 바코드의 방향과 변이를 판단하는 기준이 된다. 우측과 상단 경계부분에는 검은색과 흰색의 모듈이 번갈아가며 나타나는 타이밍 패턴이 있다. 이 타이밍 패턴을 통해 모듈의 크기와 개수를 추정할 수 있다.

(그림 1) 데이터매트릭스의 형태

2. 안드로이드 환경에서 바코드 인식 어플리케이션 구현

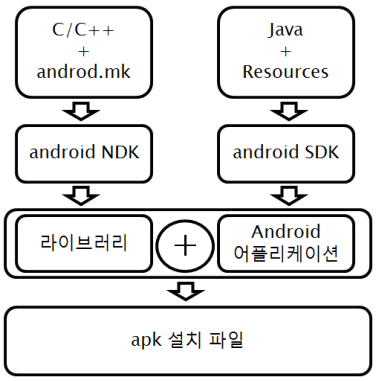
안드로이드 환경에서 어플리케이션은 자바로 구현된다. 그런데 자바환경에서는 복잡한 바코드 인식 연산을 수행할 경우 인식속도가 매우 저하된다. 따라서, 본 연구에서는 그림 2와 같이 JNI(Java Native Interface)를 이용하여 C/C++환경에서 연산을 수행한 후, 그 결과를 안드로이드의 자바환경으로 받아오게 하였다[1]. JNI를 통한 C/C++ 환경은 안드로이드 어플리케이션에서 라이브러리 형식으로 구성된다. 카메라를 구동하여 영상을 입력받는 부분은

자바 환경에서 이루어지며, 입력된 영상은 JNI를 통해 C/C++환경에 전달된다. C/C++ 환경에서 바코드 인식이 완료된 후, 다시 JNI를 통해 결과를 되돌려 준다. 자바환경에서는 돌려받은 결과값을 화면에 출력한다.



(그림 2) JNI를 사용한 바코드 인식 구현

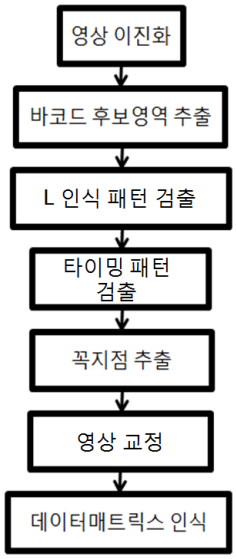
안드로이드 환경에서 JNI가 사용되는 과정은 그림 3과 같다. JNI를 사용하기 위해서는 JDK(Java Development Kit)와 안드로이드 환경에 맞는 크로스 컴파일러가 필요하다. 크로스 컴파일러로는 안드로이드 NDK가 무료로 제공되고 있다[2]. 라이브러리 파일을 생성하기 위해서 안드로이드 프로젝트 폴더 내부에 jni라는 폴더가 있어야하며, 그 안에 android.mk 파일이 있어야 한다. android.mk 파일은 C/C++환경을 구성하는 헤더 파일과 소스코드 파일에 대한 설정과 함께 C/C++환경을 빌드하기 위한 각종 정보를 포함한다. 안드로이드 NDK를 통해 라이브러리 파일이 생성되면 자바환경에서 생성된 안드로이드 어플리케이션과 함께 apk 파일에 포함된다. apk 파일은 안드로이드 환경에서 사용되는 설치 파일이다.



(그림 3) 안드로이드에서 JNI 활용

3. 데이터매트릭스 인식 알고리즘

데이터매트릭스는 인식과정을 위해 여백과 L 인식 패턴 그리고 타이밍 패턴을 통해 정보를 제공한다. 이를 통해 모듈의 크기와 같은 데이터매트릭스 자체의 정보뿐만 아니라 영상이 회전이나 기울어짐, 훼손 정도와 같은 변이에 대한 정보까지도 추정할 수 있다. 그러므로, 데이터매트릭스를 인식하기 위해서는 L 인식 패턴과 타이밍 패턴을 정확하게 검출할 필요가 있다. 데이터매트릭스 인식 단계는 그림 4와 같다.



(그림 4) 바코드 인식 순서

1) 이진화

L 인식 패턴 및 타이밍 패턴 검출을 위해 우선 수집 영상을 이진 영상으로 변환한다. 모바일 장치의 사용 환경은 매우 다양하므로 영상의 명도가 균일하지 않다. 조명의 영향에 따른 영상의 변이를 최소화하기 위해 지역적 이진화 방법 중 하나인 수정된 Niblack 알고리즘을 사용하였다[3].

2) 후보영역 추출

이진화가 완료된 후 전체 영상 중에서 데이터매트릭스가 포함된 바코드 후보영역을 추출한다. 데이터매트릭스는 검은색과 흰색 모듈이 일정한 규칙에 의해 배열되어 있는 형태로 대부분의 흑화소는 서로 연결되어 있다. 또한 여백으로 둘러싸여 있어 배경의 노이즈와 분리되어 있다. 그러므로 전체 영상에 대해 서로 연결된 흑화소 성분들을 추출해내면 데이터매트릭스를 포함하는 바코드 후보영역을 추출할 수 있다. 바코드 후보영역들이 서로 겹치는 경우에는 하나의 큰 영역으로 합성한다. 데이터매트릭스 이외에도 문자나 숫자, 혹은 작은 이미지와 같은 노이즈들이 연결 성분 영역으로 추출될 수 있으므로 추출된 영역의 크기와 데이터매트릭스의 최소 크기[1]를 비교하여 작은 크기의 영역들을 제거한다. 작은 크기의 후보 영역들이 제거된 후, 남은 후보 영역들에 대해서 L 인식 패턴을 검출하여 데이터매트릭스가 포함되었는지 판단한다.

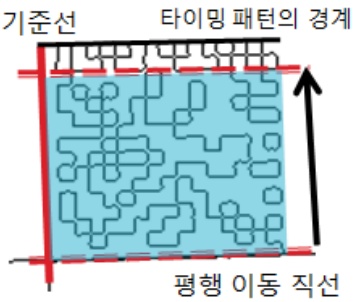
1) 데이터 매트릭스의 최소 권장 밀도는 0.25 mm 이며 각각의 행과 열은 최소 10개 이상의 모듈로 구성된다.

3) L 인식 패턴 검출

각 후보영역에 대해 데이터매트릭스가 포함되어 있는지 여부를 검증하기 위해서 L 인식 패턴을 찾는다. L 인식 패턴은 직선의 형태이므로 외곽선을 추출한 후 직선 검출을 통해 L 인식 패턴을 검출하였다. 직선 검출에는 Ramers Douglas Peucker 알고리즘을 사용하였다[4]. 후보영역 내의 모든 직선을 검출해낸 후, L 인식 패턴을 찾는다. L 인식 패턴은 데이터매트릭스에서 가장 긴 두 개의 직선으로 동일한 길이를 갖고 있으며 서로 수직으로 연결된 형태이다. 그러므로, 후보영역 내부에서 가장 긴 두 직선을 추출한 후 두 직선의 길이와 연결 여부, 그리고 내각의 크기를 기준으로 L 인식 패턴인지를 판단한다. 영상 취득 시의 환경과 영상 처리 과정에서의 변이에 따라 각 기준의 값이 달라질 수 있으므로 실제 구현 시 ±10퍼센트 정도의 오차를 허용하였다.

4) 타이밍 패턴 검출

L 인식 패턴을 찾아낸 후 패턴을 구성하는 두 직선을 직선의 식을 찾는다. 그 후, 그림 4과 같이 두 직선 중 하나의 직선을 기준으로 다른 직선을 여백에 도달할 때까지 평행이동시켜 타이밍 패턴의 경계를 찾는다. 직선이 이동할 때마다 직선상의 흑화소의 개수를 센다. 영상의 변이로 인해 실제 촬영된 L 인식 패턴과 타이밍 패턴이 평행하지 않을 수도 있으므로 실제 구현 시에는 이동된 후의 직선상의 흑화소의 개수가 L 인식 패턴을 구성하는 흑화소의 개수의 10퍼센트 이하일 때를 경계값으로 설정하였다. 또한 타이밍 패턴의 경계를 이루는 직선에 대해 RANSAC 알고리즘을 적용하여 정확도를 높인다[5].



(그림 5) 직선 이동을 이용한 타이밍 패턴 검출

5) 직선의 식을 이용한 꼭지점 추출

L 인식 패턴과 타이밍 패턴을 구성하는 총 네 개의 직선을 찾아낸 후 변이로부터 영상을 교정하기 위해 데이터매트릭스의 꼭지점을 추출한다. 꼭지점은 데이터매트릭스의 형태를 나타내는 중요한 정보일 뿐만 아니라 영상 교정과정에도 중요하게 사용되므로 정확하게 추출되어야 한다. 꼭지점의 정확도를 높이기 위해 RANSAC 알고리즘을 적용하여 직선의 식의 정확도를 높였다. 꼭지점은 두 직선이 수식 1의 형태로 나타날 때 수식 2의 교점공식을 사용하여 구할 수 있다.

$$Eq1: ax+by+c=0$$
$$Eq2: dx+ey+f=0 \quad (1)$$

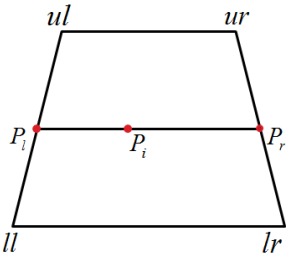
$$\text{교점 } I(x,y)=\left(\frac{b\times f-c\times e}{a\times e-b\times d}, \frac{a\times f-c\times d}{b\times d-a\times e}\right) \quad (2)$$

6) 영상 교정

기본적으로 데이터매트릭스는 정사각형이지만 영상 취득 시 영상의 회전이나 기울어짐이 발생하여 그 모양이 달라질 수 있다. 이러한 변이를 해결하기 위해 입력 영상의 좌상단, 우상단, 좌하단, 우하단의 네 꼭지점을 찾아서 영상을 교정한다. 각각의 꼭지점은 ul, ur, ll, lr으로 나타낸다. 영상을 교정하기 위해 교정된 영상의 좌표 $P_d=(x_d, y_d)$에 대응되는 입력 영상의 좌표 $P_i=(x_i, y_i)$를 추정하여 그 화소값을 교정 영상에 할당한다.

$$P_l = ul+(ll-ul)\frac{y_d}{destHeight} \quad (3)$$

$$P_r = ur+(lr-ur)\frac{y_d}{destHeight} \quad (4)$$



(그림 6) 입력 영상에서 각 좌표의 위치

수식 3의 P_l과 P_r은 그림 6에서와 같이 입력 영상의 좌측과 우측에서 교정 영상의 좌표 y_d에 대응하는 입력 영상의 위치를 나타낸다. $destHeight$는 교정 영상의 높이를 나타내고 $destWidth$는 너비를 나타낸다. 수식 3의 $ll-ul$은 ul을 기준으로 영상의 좌측에서의 변이를 벡터로 나타낸 것이며 교정 영상의 좌표의 비율인 $\frac{y_d}{destHeight}$를 곱하여 입력 영상에서 교정 영상의 좌표 중 y_d값에 해당하는 위치를 찾는다. 수식 4를 이용해 영상의 우측에서의 변이에 대해서도 같은 연산을 수행한다. 마지막으로 수식 5를 이용해 교정 영상의 좌표 x_d에 대응되는 위치를 추출하여 (x_d, y_d)에 대응되는 좌표 P_i를 구할 수 있다.

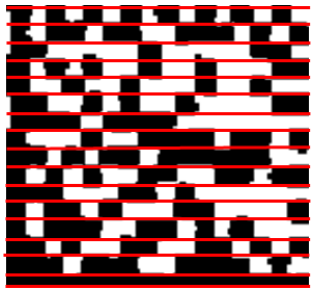
$$P_i = P_l+(P_r-P_l)\frac{x_d}{destWidth} \quad (5)$$

바코드 영상이 교정되면 기울어지거나 회전되었던 영상

이 정사각형 모양으로 교정된다. 이 과정에서 직선상의 픽셀이 톱니형태로 돌출되거나 모듈 사이에 여백이 발생하는 경우가 있는데, 팽창 연산을 통해 그림 7과 같이 노이즈와 여백을 제거한다.

(그림 7) 팽창연산을 통한 노이즈/여백 제거

팽창 연산 이후, 대부분의 여백은 제거 되었지만 여전히 약간의 노이즈가 남아있기 때문에 같은 행과 열의 모듈이라도 크기가 조금씩 다른 경우가 발생한다. 그래서 타이밍 패턴에서 픽셀값이 변화하는 지점을 직접 추출하여 모듈 수치화를 위한 행과 열에 대한 구분선을 적용한다. 그림 8과 같이 구분선을 적용한 후 각 구분선에 의해 나뉘어진 영역에서 흑화소의 수가 분할 영역의 60퍼센트 이상인 경우를 1로 아닌 경우를 0으로 결정하여 모듈의 수치데이터를 추출한다.



(그림 8) 모듈 수치화를 위한 행 구분선 적용

4. 실험 및 결과

구현된 바코드 인식기를 모바일 안드로이드 환경에서 구동하여 인식률을 실험해보았다. 실험에 사용된 모바일 안드로이드 환경은 HTC 디자이어 단말기와 android 2.2 프로요 운영체제를 사용하였다. 그림 9는 인식에 사용된 샘플 영상 중 일부이다.

(그림 9) 인식 실험 샘플 영상

총 102개의 샘플 영상으로 인식 실험을 진행한 결과 100퍼센트의 인식 성공률을 보였다. 바코드 패턴 추출과정에서 3.9 퍼센트의 오류가 발생하였으나 오류 정정 코드에 의해 자동으로 교정되었다.

5. 결론

본 논문에서는 모바일 안드로이드 환경에서 이차원 바코드 중 데이터매트릭스를 검출하고 인식하는 과정을 소개하였다. 안드로이드 모바일 환경에서 어플리케이션은 자바로 구현된다. 그러나, 바코드 인식과정을 자바로 구현하면 속도가 느려지므로 JNI를 이용하여 C/C++환경에서 인식한 후 자바환경에서 결과를 출력하도록 구현하였다. 먼저 연산량을 줄이기 위해 입력 영상으로부터 바코드 후보영역을 추출하였다. 그리고, 직선 검출 알고리즘을 사용하여 데이터매트릭스검출의 기본이 되는 L 인식 패턴과 타이밍 패턴을 찾아내었다. 이 과정에서 직선 추출의 정확도를 높이기 위해 RANSAC 알고리즘을 적용하였다. 영상의 변이를 고려하여 영상의 네 꼭지점을 기준으로 정사각형 형태로 영상을 교정한 뒤 교정된 영상을 사용하여 데이터매트릭스를 인식하였다.

실제 모바일 기기를 이용하여 다양한 환경에서 인식 실험을 해본 결과 신뢰할 만한 인식률을 얻을 수 있었다. 앞으로 모바일 환경이 더욱 확대됨에 따라 본 논문에서 소개된 이차원 바코드 인식도 더욱 넓은 분야에서 활용될 것으로 기대된다.

*본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신 인력 양성사업으로 수행된 연구결과임

참고문헌

[1] Sheng Liang (1999), The JavaTM Native Interface Programmer's Guide and Specification. Addison-Wesley

[2] Google Inc., Android NDK | Android Developers, Retrieved September 28, 2010, from http://developer.android.com/sdk/ndk.html

[3] 김인중, "화질 분석을 통한 카메라 문서 영상의 적응적 이진화" 한국정보과학회, 정보과학회논문지 : 소프트웨어 및 응용, Vol.34 No.9, pp. 785-796, 2007

[4] J. Hershberger and J. Snoeyink, "Speeding up the Douglas-Peucker line simplification algorithm." In Proc. 5th Intl. Symp. Spatial Data handling. IGU Commission on GIS, pp. 134-143, 1992

[5] David A. Forsyth and Jean Ponce (2003). Computer Vision, a modern approach. Prentice Hall. ISBN 0-13-085198-1