The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 14, No. 2, pp.141-146, Apr. 30, 2014. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2014.14.2.141
JIIBC 2014-2-20

터치스크린을 이용한 필기체 문자 인식 알고리즘 설계 및 구현

Implementation and Design of Handwritten Character Recognition Algorithm Using Touch Screen

박상봉*

Sang-bong Park*

요 약 본 논문은 모바일 터치스크린을 이용한 필기체 문자 인식 알고리즘을 제안하고, 구현된 내용을 기술한다. 제안된 시스템은 PXA320 프로세서, 정전 용량 터치 패널과 QT4를 이용한 인터페이스로 구성하였다. C++ 언어를 사용하고 제안된 알고리즘은 문자의 특성을 직선, 좌호, 우호 특징을 추출하여 3진 트리 방식으로 입력되는 문자를 결정한다. 영문자에 대한 테스트를 통하여 성능을 검증하였다. 기존 방식보다 간단한 알고리즘으로 구성되므로, 모바일 터치스크린의 문자인식에 적용이 가능하다.

Abstract This paper describes the implementation and algorithm of handwritten character recognition using mobile touch screen. The system is consisted of PXA320 processor, capacitive touch panel and QT4 interface. The proposed algorithm extracts pattern characteristics with straight, left circle, right circle on the inputting character. The definition of character is determined by 3-way tree searching method. The performance of proposed algorithm is verified using alphabet character. It is suitable to apply the mobile touch screen because of simple algorithm.

Key Words : Touch Screen, Handwritten Character Recognition, 3-way Tree Searching Method

Ⅰ. 서 론

주변에서 스마트폰을 비롯한 각종 모바일 디바이스에서 터치스크린은 흔히 볼 수 있는 HID(Human Input Device) 로 사용되고 있다. 이러한 터치스크린에서 문자를 인식하는 방식은 키패드를 사용하거나 필기체 글자 인식 입력과 음성 인식 입력 방식으로 구성된다. 필기체 글자 인식은 실시간으로 입력되는 문자를 인식하는 온라인 인식과 사용자의 필기에 대한 인식이 필기한 이후에 진행되는 오프라인 방식으로 나누어진다. 필기체 문자 인식의 경우 사용자마다 글자를 쓰는 다양한 패턴이 존재하므로, 이러한 다양한 변형이 적용된 입력 데이터를 인식하기 위해서는 일정한 형태로 정규화 하는 것이 문자 인식의 정확도를 결정한다. 본 논문에서는 터치스크린을 이용한 정규화 된 온라인 필기체 문자 인식을 통하여 자동 문자 인식을 수행하는 알고리즘을 제안한다. 2장에서는 기존의 임베디드 리눅스 제품에서의 필기체 문자 입력 방식에 대한 알고리즘과 구현 방법을 설명하고, 3장

*정회원, 세명대학교 정보통신학부
접수일자 2014년 2월 28일, 수정완료 2014년 3월 28일
게재확정일자 2014년 4월 11일

Received: 28 February, 2014 / Revised: 28 March, 2014
Accepted: 11 April, 2014
*Corresponding Author: psbcom@semyung.ac.kr
Dept. Information&Communication, Semyung University, Korea

에서는 제안된 알고리즘을 설명한다. 4장에서는 제안된 필기체 문자 인식 시스템 개발 환경과 구현 내용을 기술한다. 실제 입력된 필기체에 대한 문자 인식을 실행한 검증 결과를 기술하고, 5장에서는 결론을 적었다.

Ⅱ. 기존 필기체 문자 인식 알고리즘

터치스크린이나 디지타이저와 같은 장치를 통해 필기되는 것을 실시간으로 인식하는 온라인 필기 인식에서는 사용자가 필기하는 터치스크린에 (x, y) 좌표가 시간에 따라 순차적으로 입력된다. 필기체 인식 과정은 일반적으로 다음 단계들로 구성된다.

분리 단계는 선, 단어를 분할해서 최종적으로 모든 문자를 분리한다. 이 단계에서는 문자의 경계를 찾아내고, 다음 처리를 위해서 문자들을 분리한다. 전처리 단계는 이미지의 초기 처리 과정이고, 인식 시스템의 입력으로 사용된다. 전처리 단계에서는 필기 방식의 다양한 변형을 가능한 줄여서 정확도를 높이기 위한 처리를 수행한다. 터치스크린에 따라서 동일한 글자를 필기하더라도, 필기 형태가 다르기 때문에 글자의 삐침, 크기, 속도 방향의 변형을 지닌다. 이러한 변형을 판단하여 글자의 기울기를 보정한다. 인식 단계는 이전 단계 처리 과정을 통하여 입력 이미지에 대한 보정이 완료되면 미리 정의된 기준이 되는 모델과 비슷한 필기 입력과 비교해서 정확한 문자를 찾아낸다. 기준 모델의 특징 값과 입력된 필기 데이터의 특징 값의 거리를 구하는 방식으로 여러 가지 알고리즘(Linear Matching, Elastic Matching, Neural Network, Hidden Markov Model)을 적용해서 사용자의 글씨를 인식한다. 기존의 EDK-320 내장 필기 문자 입력기의 경우에는 Qtopia 플랫폼을 사용하여 개발되었다. Qtopia는 리눅스 제품의 공통 플랫 폼으로 사용되며 사용자 정의 다중 응용 제품에 활용된다. Qtopia를 사용한 필기 문제 인식의 경우에는 우선 리눅스 기반의 터치스크린 라이브러리를 편집한다. 주로 작업 패턴 설치, 데이터 읽기 및 쓰기, 읽은 데이터 전송 등을 수행하는 인터페이스 함수를 제공한다. 최종적으로 동적 연결 라이브러리로 컴파일되고 상위의 응용프로그램으로부터 호출을 받게 된다. [1][2][3]

읽은 원시(raw) 데이터에 대한 변경, 교정, 배치 등의 처리를 위하여 Qt/Embedded 단계의 터치스크린 API를 수정한다. 이 때 터치스크린 API 파일은 qwsmouse_qws.cpp를 사용한다. Qtopia 단계에서는 입력기 API를 사용하여 터치스크린 입력 장치를 통해 사용자로부터 입력되는 필기 정보를 입력하고, 입력된 필기 정보를 디스플레이 한다. 또한 필기된 정보와 일치하는 식별 글자를 디스플레이하는 인터페이스를 제공한다. 사용된 입력기 플러그 인 인터페이스는 InputMethodInterface 클래스를 이용하며, 이 클래스를 이어받기 및 클래스의 멤버 함수를 다시 로그인을 통하여 필기 인식을 완성한다. 최종적으로 연결 라이브러리로 컴파일 되고 Qtopia 서버에게 자동적으로 호출을 제공한다. 그림 1은 Qtopia 임베디드 플랫폼을 이용한 애플리케이션을 나타낸다. 사용자에게 입력받은 필기 문자를 비트맵으로 저장 후, 컬러일 경우 필터링을 거쳐 흑백 이미지로 변환한다. 변환된 흑백 이미지는 필요에 따라 확대, 축소, 위치 이동, 회전시켜서 선형 변환을 완료하고, 기준 데이터 셋과 대조한다. 실제 입력된 문자를 기준 데이터 셋에 이미지와 유사도가 높은 이미지가 최종 식별 결과로 디스플레이 된다. 이러한 어플리케이션에서는 전처리 과정 정도에 따라 오차가 비교적 많이 발생하는 문제가 발생되고, 픽셀 단위의 이미지 데이터를 기반으로 하므로 연산 처리 과정이 복잡하다.

1. 기존의 Qtopia 내장 필기 문자 인식기
Fig. 1. Qtopia Embedded handwritten character recognizer

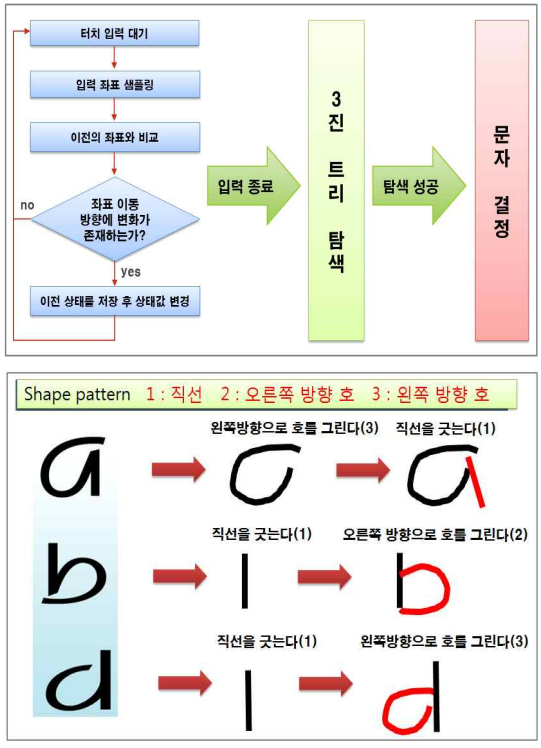
Ⅲ. 제안된 필기체 문자 인식 알고리즘

제안된 필기체 문자인식 알고리즘은 분할, 전처리 및 인터페이스는 기존의 Qtopia에서 제공하는 API와 라이브러리를 사용하였다. 문자 인식에 대한 알고리즘은 기

존의 비트맵 방식의 오류와 복잡성을 개선하기 위하여 그림 2와 같은 흐름도를 지닌다. 그림 2에서 첫단계로 인터페이스를 통하여 터치 입력 대기 상태에서 좌표 이동 방향에 변화가 존재할 때 이전 상태를 저장한 후 상태 값을 변경해준다. 변화가 존재하지 않을 때는 다시 터치 입력 대기 상태로 돌아간다. 스크린의 입력좌표 (x, y)를 이용하여 라디안 변화를 산출하고 지속적으로 변화량을 추출한다. 입력되는 좌표의 각도 변화가 정의된 임계치를 초과하는지를 모니터링 한다. 만약 초과했다면 그 값이 양인지 음인지를 검사하여 결과에 따라서 좌회전하는 호, 우회전하는 호로 판별한다. 일정 기간 동안 임계치 내에서만 좌표 값이 변화하면 직선으로 판별한다. 이러한 지속적인 상태 변화를 검출하고, 입력되는 문자 특성에서 다른 형태로의 변화가 발생하게 되면 이전 상태를 저장하는 동작을 반복 실시한다. 문자 입력이 종료되면 3진 트리 탐색을 통하여 취합된 좌호, 우호, 직선에 대한 조합을 기준 문자 데이터 셋과 비교하여 일치하는 문자를 출력한다. 문자 a와 같은 경우는 좌호와 직선으로 구성되어 있고, b는 직선과 우호로 구성되어 탐색과정이 진행된다.

그림 3은 제안된 알고리즘에 대한 설계 구조를 나타낸다. Qt4에서 제공하는 QApplication 객체를 생성한 후, Paint 클래스를 상속받는다. 상속받은 Paint 클래스에서는 Mouse Event를 이용하여 사용자가 터치한 좌표를 얻는다. Paint 클래스 내부에는 문자 패턴 처리에 대한 PATT 클래스를 선언하고, 마우스 이벤트가 발생 시 Insert 메소드를 호출하여 좌표 값을 전달한다. 전달받은 좌표 값에 따라 사용자의 입력 변화를 산출하여 연결리스트를 생성하고, 기준 데이터 셋과 각각 대조하여 일치하는 문자를 탐색한다.

입력되는 좌표 값에 대한 각 변화량을 라디안으로 추출하기 위하여 math.h 라이브러리를 활용한다. atan2() 함수를 이용하여 라디안을 π ~ -π 까지 사이의 값으로 변환한다. 이때 반환하는 값에 π를 더해서 실제로는 0부터 2π 까지 라디안으로 출력된다. 그림 4와 5는 좌회전하는 호와 우회전하는 호의 라디안 변화를 나타낸다. 그림과 같이 필기 글자가 좌회전하는 호를 그릴 때에선 라디안이 감소하고, 우회전하는 호를 그릴 때에는 라디안이 증가한다. 직선을 그릴 때의 라디안은 일정 범위 내에서 크게 벗어나지 않는다. 수집된 좌호, 우호 샘플들의 경우에 대부분의 표본이 예상된 방향으로 움직임을 나타낸다. 그림 4에서 라디안 12, 13은 변화량이 양인데, 이것은 터치스크린의 샘플링 시간 내에서 발생하는 오차이므로 필터링을 통하여 개선하였다. 그림 6은 직선, 좌호, 우호를 결정하는 C++ 로 구현된 소스 코드이다. 소스 코드는 라디안 변화가 임계치를 넘지 않으면 직선으로 인식되고, 라디안 변화가 이전 값에 비해 양으로 증가하면 우호로 인식된다. 마찬가지로 라디안 변화가 음으로 감소하면 좌호로 인식된다. 그림 7은 추출된 직선과 좌호, 우호에 대해 탐색을 진행하는 3진 트리 탐색 코드이다.



2. 제안된 필기체 인식 알고리즘 순서도
Fig. 2. Flowchart of the proposed handwritten character recognition algorithm

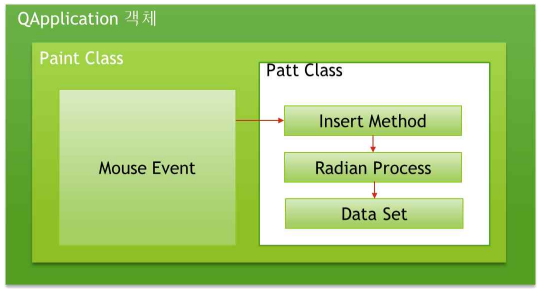
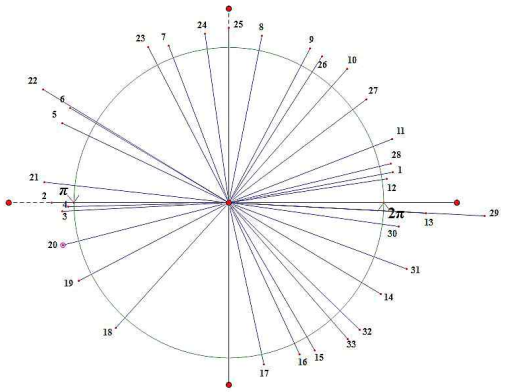


그림 3. 제안된 필기 인식 설계 구조
Fig. 3. Design construction of the proposed algorithm



4. 좌호를 쓰는 경우 좌표 및 그림
Fig. 4. Coordinate and plot of left c

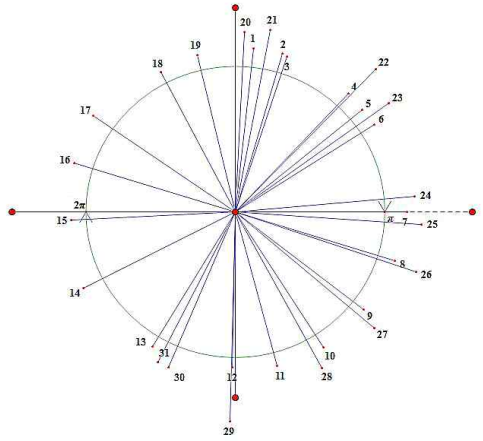


그림 5. 우호를 쓰는 경우 좌표 및 그림
Fig. 5. Coordinate and plot of right arc

```
void PATT::decision_line(struct coor_link * now){
	static int status=NONE_ST;	// 현재 어느 형태와 가장 가까운가?
	static int doubt_status=NONE_ST;
	static int count=0;	// 몇개의 데이터를 카운트했는가.
	static double rad_temp;

	if(count++<1){  // 아직 카운트된 요소가 하나도 없을 시
		doubt_status=1; // 요주관찰대상
		rad_temp=now->angle;
		return;
	}
	if(fabs(now->angle-rad_temp)<CRIT){	// 각변화가 임계치를 넘지 않으므로 직선.
		if(status!=_line){	// 이전 상태가 선이 아니라면
			if(doubt_status==0){
				doubt_status=_line;
			}
			else if(doubt_status==_line){
				status=_line;
				dt.insert(status,rad_temp);
				doubt_status=0;
			}
		}
		rad_temp=now->angle;
	}
	else{
		if((now->angle-rad_temp)>0){	// 우회전 -> radian increase
		// 2PI -> 0 을 어떻게 처리할 것인가.
			if(status!=_right_arc){ // 상태변화.
				if(doubt_status==0){
					doubt_status=_right_arc;
				}
				else if(doubt_status==_right_arc){
					status=_right_arc;
					dt.insert(status,rad_temp);
					doubt_status=0;
				}
			}
		}
		else{				// 좌회전 -> radian decrease
		// 0 -> 2PI 를 어떻게 처리할 것인가.
			if(status!=_left_arc){  // 좌회전인데 이전 상태가 좌회전이 아니고
				if(doubt_status==0){	// 의심치가 없을 때
					doubt_status=_left_arc; // 의심치 추가.
				}
				else if(doubt_status==_left_arc){	// 의심치가 있을 때
					status=_left_arc;
					dt.insert(status,rad_temp);
					doubt_status=0;
				}
			}
		}
		rad_temp=now->angle;
	}
	printf("now status : %d, rad_temp : %lf\n",status,rad_temp);
}
```

그림 6. 직선, 좌호, 우호 결정 소스 코드
Fig. 6. Source code of straight, left arc, right arc

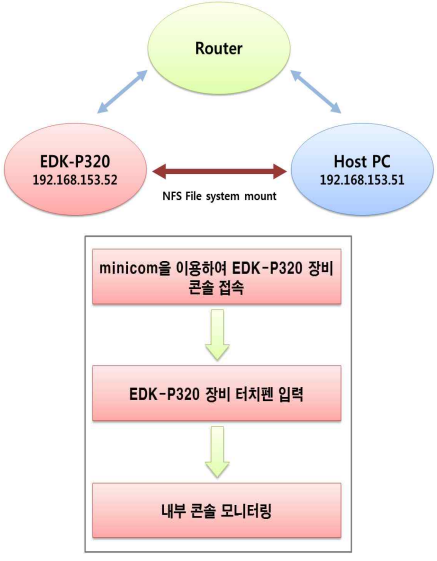
```
void PATT::decision_line(struct coor_link * now){
	static int status=NONE_ST;	// 현재 어느 형태와 가장 가까운가?
	static int doubt_status=NONE_ST;
	static int count=0;	// 몇개의 데이터를 카운트했는가.
	static double rad_temp;

	if(count++<1){  // 아직 카운트된 요소가 하나도 없을 시
		doubt_status=1; // 요주관찰대상
		rad_temp=now->angle;
		return;
	}
	if(fabs(now->angle-rad_temp)<CRIT){	// 각변화가 임계치를 넘지 않으므로 직선.
		if(status!=_line){	// 이전 상태가 선이 아니라면
			if(doubt_status==0){
				doubt_status=_line;
			}
			else if(doubt_status==_line){
				status=_line;
				dt.insert(status,rad_temp);
				doubt_status=0;
			}
		}
		rad_temp=now->angle;
	}
	else{
		if((now->angle-rad_temp)>0){	// 우회전 -> radian increase
		// 2PI -> 0 을 어떻게 처리할 것인가.
			if(status!=_right_arc){ // 상태변화.
				if(doubt_status==0){
					doubt_status=_right_arc;
				}
				else if(doubt_status==_right_arc){
					status=_right_arc;
					dt.insert(status,rad_temp);
					doubt_status=0;
				}
			}
		}
		else{				// 좌회전 -> radian decrease
		// 0 -> 2PI 를 어떻게 처리할 것인가.
			if(status!=_left_arc){  // 좌회전인데 이전 상태가 좌회전이 아니고
				if(doubt_status==0){	// 의심치가 없을 때
					doubt_status=_left_arc; // 의심치 추가.
				}
				else if(doubt_status==_left_arc){	// 의심치가 있을 때
					status=_left_arc;
					dt.insert(status,rad_temp);
					doubt_status=0;
				}
			}
		}
		rad_temp=now->angle;
	}
	printf("now status : %d, rad_temp : %lf\n",status,rad_temp);
}
```

그림 7. 3진 트리 탐색 소스 코드
Fig. 7. Source code of 3 way tree search

Ⅳ. 구현 및 테스트 결과

그림 8은 제안된 알고리즘의 테스트를 위한 개발 환경이다. 한백전자의 EDK-P320 툴 킷을 사용하여 구현하였고, 터치 패널의 입력을 사용하기 위해 linuxtp 라이브러리를 사용하였다. QT4를 이용하여 인터페이스를 구현하였으며, 컴파일러는 arm-linux-gcc 4.1.1을 이용하여 크로스 컴파일 하였고, 언어는 C++을 사용하였다. 터치스크린 장비와 실제적인 프로그램이 존재하는 PC를 네트워크로 연결 후, NFS 마운트하여 실행 파일을 장비에 직렬 통신을 이용하여 다운로드 후 장비 콘솔에서 결과를 확인하였다. 그림 9는 터치스크린에 a를 필기체 입력과 필기체 a에 대한 연산 처리 과정을 나타내는 좌표 값과 a를 인식한 테스트 결과를 나타낸다.



8. 개발 환경 결과
Fig. 8. Development Environment

그림 9. (a)필기체 입력

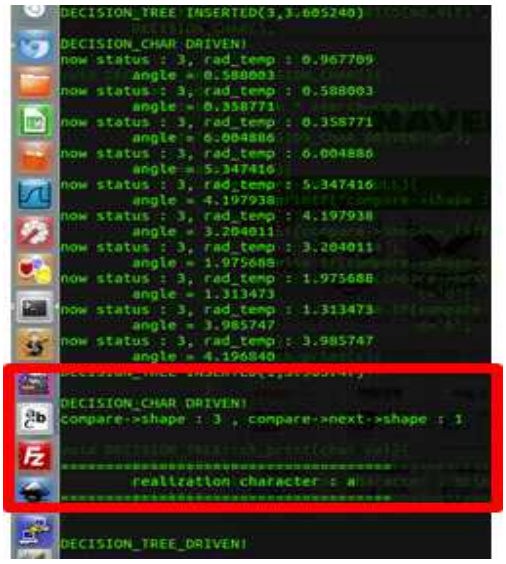


그림 9. (b) 식별 과정 좌표 및 결과
Fig. 9. (a) handwritten input (b) recognition coordinate and result

표 1. 종전 알고리즘과 비교
Table 1. Comparison of algorithms

성능비교	기존의 알고리즘	제안하는 알고리즘
문자 입력 기준선 사용	기준선에 맞게 입력하지 않으면 인식 불가	기준선 자체가 없음
이미지 프로세싱 사용 여부	이미지 프로세싱을 사용하고 연산이 많고 복잡	단순히 좌표값과 각도만을 계산하므로 연산이 적고 단순
다양한 글씨체의 수용성	수용하기 어려움	수용이 용이함
범용성	이미지 입력 장치만 존재한다면 우수	터치패널에 대해서만 적용
알고리즘 복잡성	복잡	단순

문자 a는 좌호와 직선의 형태로 식별하였고, 다른 글자에 대해서도 마찬가지로 장비에 직접 입력하여 실제 의도했던 문자와 같은지 테스트하였다. 모니터링 되는 데이터는 좌표 값, 각도 변화량, 연결 리스트에 추가된 패턴과 최종적으로 검출된 문자를 정상적으로 확인할 수 있었다.

표 1은 종전 방식과 제안된 알고리즘의 성능을 간단한 표로 나타내었다. 터치 패널에 한정적이라는 단점을 제외한 모든 영역에서 기존 알고리즘보다 우수함이 검증되었다.

Ⅴ. 결 론

본 논문에서는 모바일 터치스크린을 이용한 필기체 문자 인식 알고리즘을 제안하고, 구현된 내용을 기술한다. PXA320 프로세서, 정전 용량 터치 패널과 QT4를 이용한 인터페이스로 구성하였다. C++ 언어를 사용하고 제안된 알고리즘은 문자의 특성을 직선, 좌호, 우호 특징을 추출하여 3진 트리 방식으로 입력되는 문자를 결정한다. 영문자에 대한 테스트를 통하여 성능을 검증하였다. 제안하는 알고리즘의 기존의 알고리즘보다 연산량이 적고, 다양한 글씨체에 대해서 수용해낼 수 있다. 알고리즘이 단순하기에 구현이 용이하며, 문자 입력의 기준선이 필요 없으므로, 어떠한 각도에서도 문자 식별이 가능하다. 그러나 터치스크린을 기반으로 한 필기 문자 인식만 적용되므로 범용성 면에서는 향후 연구가 필요하다고 본다. 최근 모바일용 터치스크린의 필기 문자 인식으로 적용이 가능하다고 사료된다.

References

[1] M. C. Jung, "Character Segmentation Using Side View Feature in Machine-Printed Optical Character Recognition", Journal of Korean Institute of Information Technology, Vol. 8, Issue 12, pp. 271-280, Dec 2010

[2] K. C. Hong, I. S. Oh, "Extraction of Skeletons from Handwritten Hangul Characters using Shape Decomposition", Journal of Korean Institute of Information Scientists and Engineers, Vol. 27, No 6, pp. 583-594, June 2000

[3] B. Y. Choi, S. Y. Bang, "A Design of Handwriting Hangul to Improve Machine Readability", Journal of the Institute of Electronics Engineers of Korea, Vol. 22, No 3, pp. 431-400, Mar 1995

[4] J. H. Jung, T. Y. Choi, "Recognition of Printed Hangul Text Using Circular Pattern Vectors", Journal of Korean Institute of Information Scientists and Engineers, Vol. 38, No 3, pp. 33-45, Mar 2001

[5] H. H. Song, S. W. Lee, "Optimal Design of Reference Models for Large-set Handwritten Character Recognition using Simulated Annealing Combined with an Improved LVQ3", Journal of Korean Institute of Information Scientists and Engineers, Vol. 21, No 6, pp. 1059-1067, June 1994

[6] Y. H. Park, H. J. Lee, H. H. Baek, "The Character Recognition System of Mobile Camera Based Image", Journal of the Korea Academia-Industrial cooperation Society, Vol. 11, No. 5, pp. 1677-1684, May 2010

소개

상 봉(정회원)

- 1992년 2월 : 고려대 전자공학과 공학박사
- 1992년 3월~1992년 2월 : 삼성전자 선임 연구원
- 1999년 3월~현재 : 세명대학교 정보통신학부 부교수
- 2000년 7월 ~ 현재 : @lab(주) Digital 설계팀 기술고문
- 관심분야 : RFID/USN 기술, ASIC 설계, 광센서, 멀티터치