Finger Keyboard

최종 보고서

팀 명	프레C맨			
담당교수	황기태 교수님			
	이 름	**학 번**	**E-Mail**	**전화번호**
팀 원	강 진 혁	1092046	jinhuk1313@gmail.com	010-9930-5477
	원 건 희	1092060	wonkh1213@gmail.com	010-3832-3960
	윤 성 민	1292077	sungmin7465@gmail.com	010-3668-7465
	한 다 혜	1292084	sianhan92@gmail.com	010-9088-2740
홈페이지	http://github.com/freCman			

목 차

1. 개요

1.1 프로젝트 수행 목적

CUI(Character User Interface)에서 이용되는 텍스트 베이스 입력은 Hardware적인 키보드 입력에서 Software적인 터치스크린 입력으로 발전되었다. 하지만 이제까지의 텍스트 베이스 입력은 공간을 차지하고 입력키가 고정적이라는 물리적 키보드의 단점과 스마트 기기의 종속적인 키보드 사이즈를 가질 수밖에 없는 소프트웨어 키보드의 단점 등 각각의 분야에서 여러 단점이 있었다.

우리는 이런 단점들을 해결하고자 보다 HCI(Human-Computer Interaction)를 지향하는 텍스트 베이스 입력을 만들기 위해, 쉽고 편리하게 어디서든 사용할 수 있다는 의미의 영상인식 키보드인 Finger keyboard를 개발하고자 한다. 사용자는 Finger keyboard를 통해 카메라와 쉽게 구할 수 있는 종이를 이용하여 물리적 키보드와 같은 입력 장치를 대신해 입력을 할 수 있을 것이다. 이 프로그램은 카메라가 사람의 손과 키보드 자판이 인쇄된 종이가 보이는 영상을 인식하여, 최종적으로 사용자의 손가락 움직임을 키 입력으로 전달해준다. 이 프로그램은 Windows뿐만 아니라 Linux, Embedded도 타깃으로 하여 스마트 TV, Tablet PC, Smart Phone과 같은 다양한 스마트 기기의 입력 도구로도 활용될 수 있을 것이다.

1.2 프로젝트 설명

1.2.1 Desktop 환경

Desktop과 Web Cam이 연결되어있고, Web Cam이 바라보는 쪽에 인쇄된 종이가 있다. 그리고 PC에서 실행되는 프로그램이 존재한다. 동작 시나리오는 카메라가 종이 키보드를 인식하고 난 뒤, 종이 키보드 위에 손을 올려 손을 인식하게 한다. 이로써 프로그램이 동작할 준비를 한다. 사용자가 종이 키보드에 대하여 원하는 키 값을 누르는 입력 동작을 취하면, PC에 연결된 Web Cam은 그 동작을 포착하여 프로그램에 영상데이터를 전달한다. 이후 Open CV의 영상처리 개념이 적용 된 프로그램이 영상처리를 하여 사용자의 입력 동작에 대해서 인식을 하고, 그 영상 데이터를 적절한 데이터로 가공하여 키 값을 발생시킨다. 즉, 사용자는 쉽게 구할 수 있는 평범한 종이를 키보드의 기능을 하는 입력 장치로써 사용할 수 있게 된다.

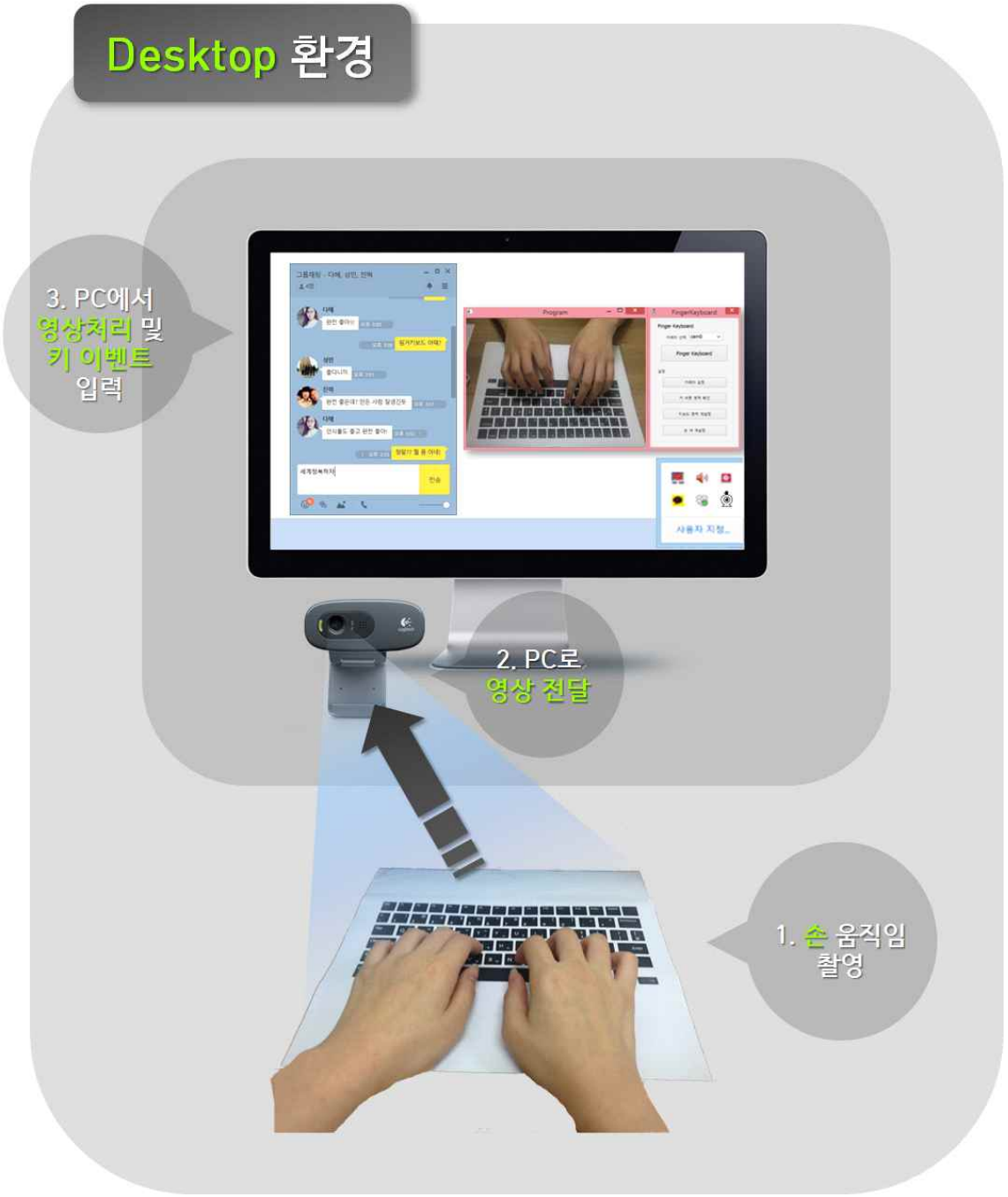
1.2.2 Embedded 환경

블루투스 통신이 가능한 Tablet PC나 Smart Phone과, 카메라모듈을 연결한 Raspberry Pi를 블루투스로 연결한다. Raspberry Pi에 사용되는 카메라는 Raspberry Pi 전용 카메라 모듈과 UVC를 지원하는 USB Web Cam을 이용할 수 있다. 카메라 모듈이 바라보는 쪽에 인쇄된 종이가 있다. 그리고 Raspberry Pi에서 실행되는 프로그램이 존재한다. 동작 시나리오는 카메라모듈이 종이 키보드를 인식하고 난 뒤, 종이 키보드 위에 손을 올려 손을 인식하게 한다. 이로써 프로그램이 동작할 준비를 한다. 사용자가 종이 키보드에 대하여 원하는 키 값을 누르는 입력 동작을 취하면, Raspberry Pi에 연결된 카메라모듈은 그 동작을 포착하여 프로그램에 영상데이터를 전달한다. 이후 Open CV의 영상처리 개념이 적용 된 프로그램이 영상처리를 하여 사용자의 입력 동작에 대해서 인식을 하고, 그 영상 데이터를 적절한 데이터로 가공하여 키 값을 발생시킨다. 발생한 키 값을 블루투스 통신을 하여 키 이벤트를 Tablet PC나 Smart Phone에 전송하여, 입력을 처리한다.

2. 구조

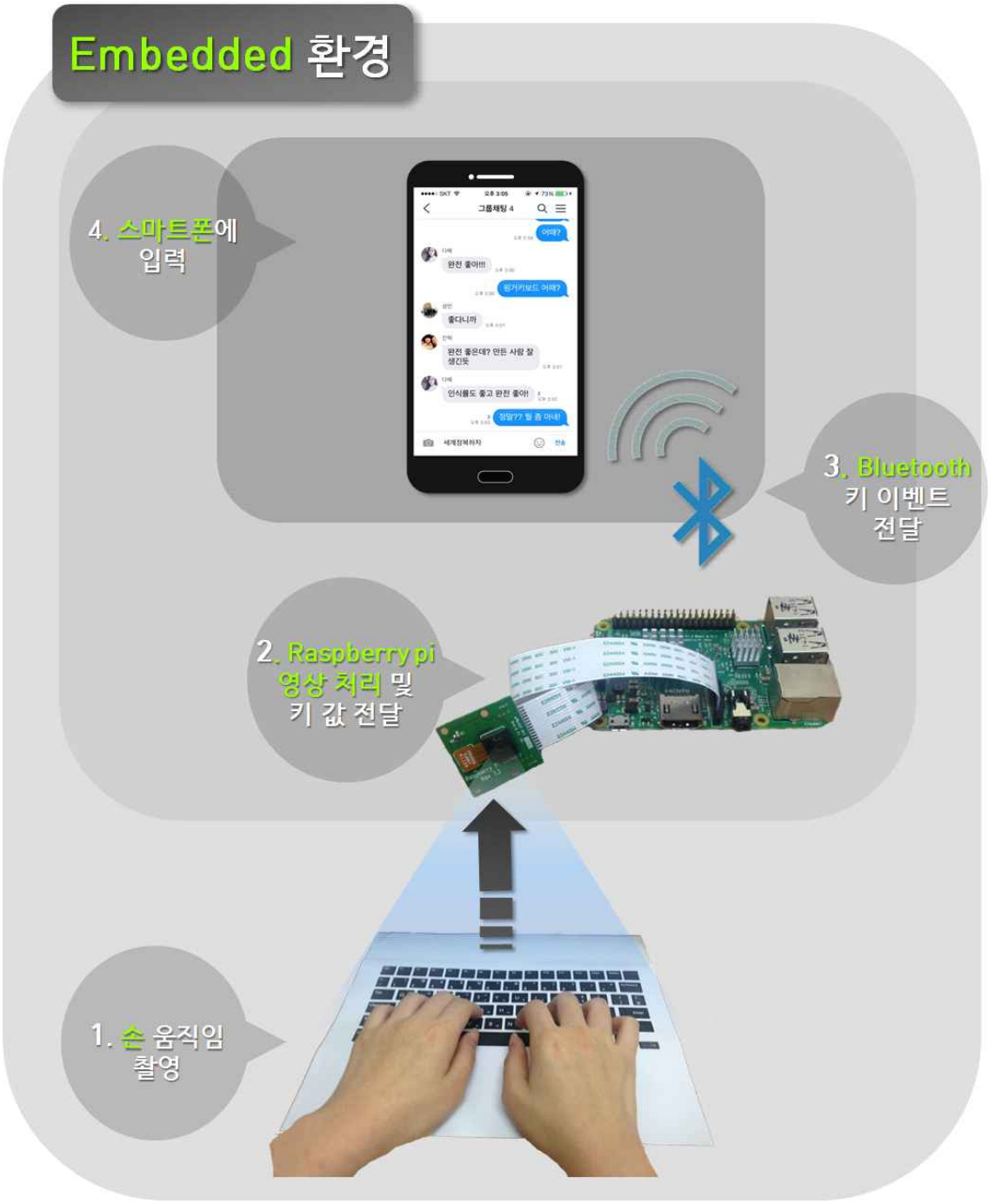
2.1 시스템 구조

2.1.1 Desktop 환경



사용자는 종이 키보드를 실제 키보드처럼 사용할 수 있다. 종이키보드와 사용자의 손이 Web Cam을 통해 촬영된다. PC에서는 Finger Keyboard 프로그램을 통해 촬영된 영상을 처리하여 종이 키보드와 사용자의 손을 인식하여 키 이벤트를 발생시킨다.

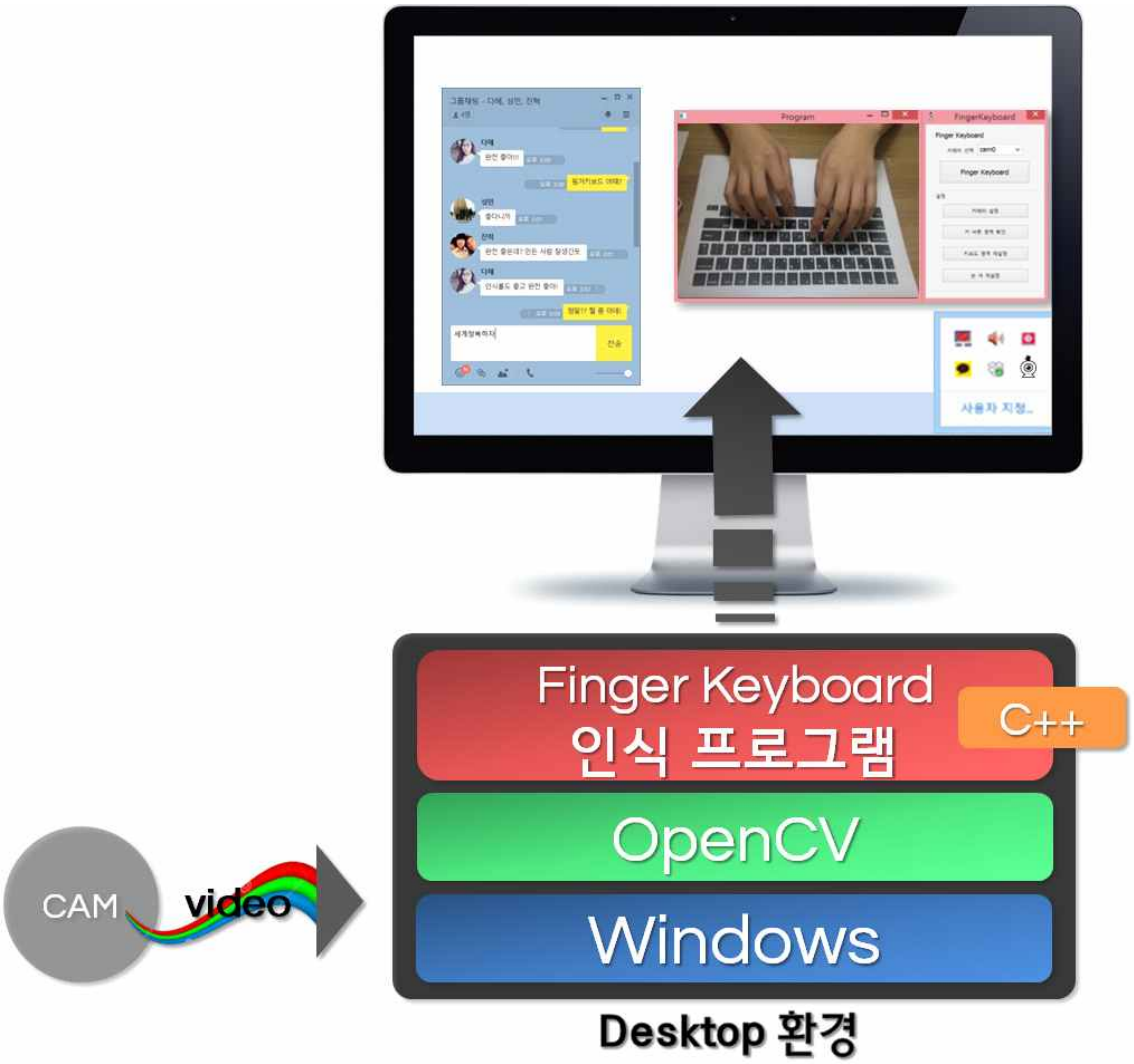
2.1.2 Embedded 환경



종이 키보드와 사용자의 손은, Raspberry Pi에 장착된 Pi Cam또는 UVC를 지원하는 USB Web Cam을 통해 촬영된다. Raspberry Pi에서는 Finger Keyboard 프로그램을 통해 촬영된 영상을 처리하여 종이 키보드와 손을 인식한다. 이때 사용자가 누른 키는 블루투스 통신을 통해 타깃 기기에 키 값을 전송하고, 키 값을 받은 타깃 기기는 키 값에 해당하는 키 이벤트를 발생시킨다.

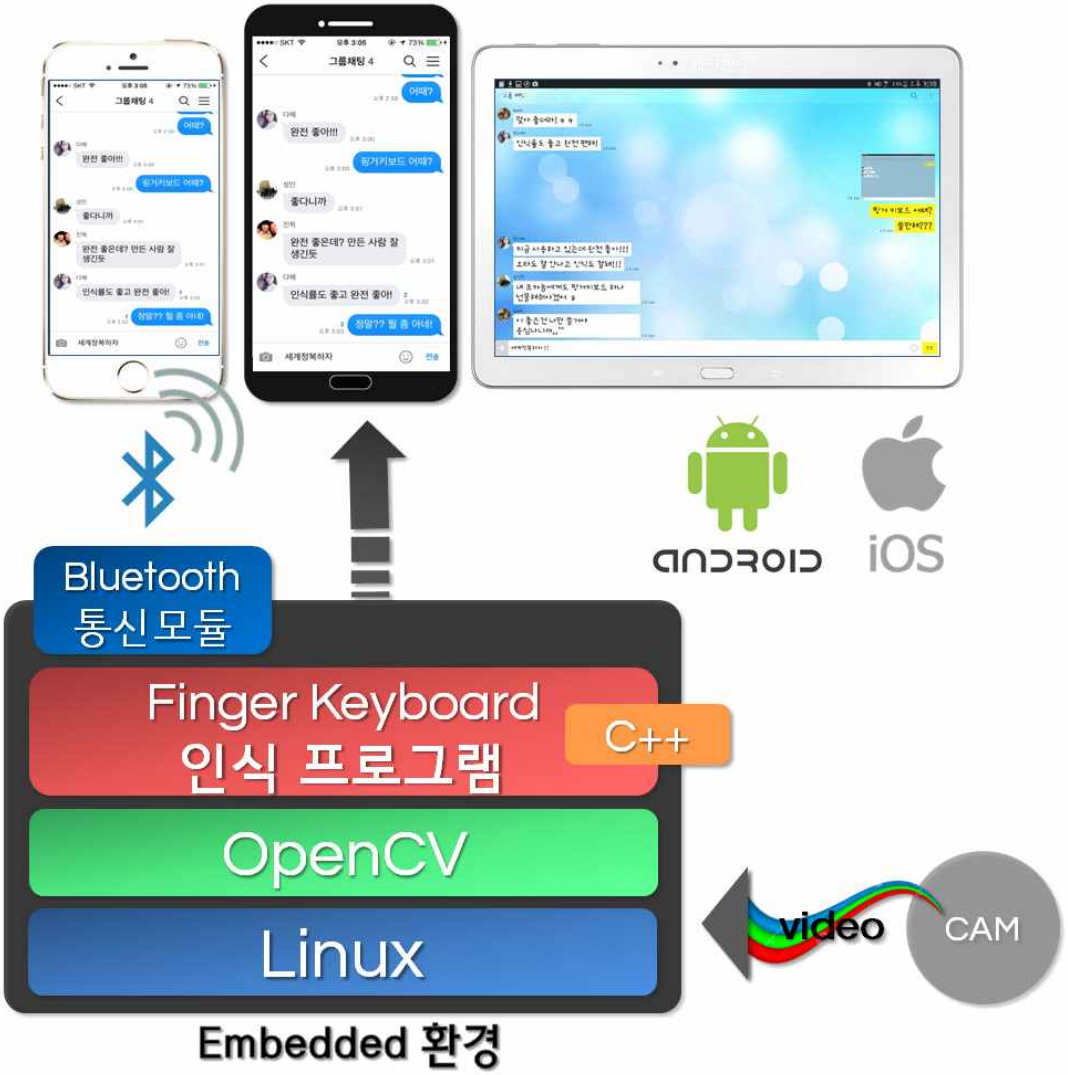
2.2 프로그램 구조

2.2.1 Desktop 환경



Web Cam을 통해서 종이 키보드와 사용자의 손을 촬영한다. 촬영된 동영상은 Windows 기반의 Desktop 으로 전달되고 우리가 만든 Finger Keyboard 프로그램으로 다루게 된다. 카메라가 종이 키보드를 인식하고 난 뒤, 종이 키보드 위에 손을 올려 인식하게 한다. 이 모든 작업은 Open CV의 영상 처리 개념이 적용 된 우리의 Finger Keyboard 프로그램이 영상데이터를 처리하여 일어나게 된다. Finger Keyboard는 실시간으로 영상처리를 하여 사용자의 입력 동작에 대해서 인식을 하고, 그 영상데이터를 적절한 데이터로 가공하여 키 값을 발생시킨다. 이 모든 작업을 사용자가 알기 쉽도록 하기 위해 Finger Keyboard는 MFC를 통해 제작한 GUI로 사용자에게 간편하고 쉬운 사용을 제공한다. Desktop 용 Finger Keyboard는 카메라 설정, 키 버튼 영역 확인, 키보드 영역 재설정, 손 색 재설정 등의 설정을 통해 인식 과정에서 사용자가 가질 수 있는 부담을 덜었다.

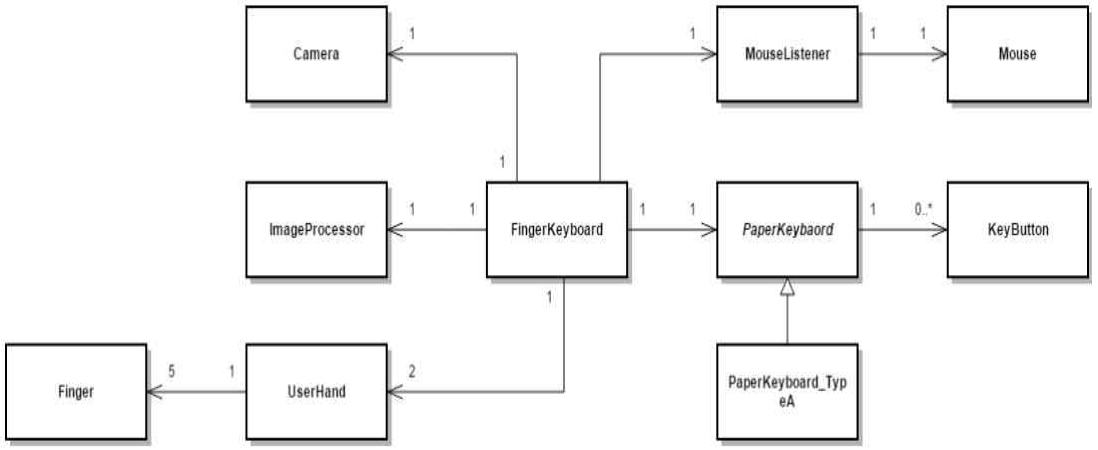
2.1.2 Embedded 환경



Raspberry Pi 전용 카메라 모듈 또는 UVC를 지원하는 USB Web Cam을 통해 종이키보드와 사용자의 손을 촬영한다. 촬영된 영상데이터는 Raspberry Pi 2의 Raspbian에서 동작하는 Finger Keyboard 프로그램으로 다루게 된다. 카메라가 종이 키보드를 인식하고 난 뒤, 종이 키보드 위에 손을 올려 인식하게 한다. 이 모든 작업은 Open CV의 영상처리 개념이 적용된 우리의 Finger Keyboard 프로그램이 영상데이터를 처리하여 일어나게 된다. Finger Keyboard는 실시간으로 영상처리를 하여 사용자의 입력 동작에 대해서 인식을 하고, 그 영상 데이터를 적절한 데이터로 가공하여 키 값을 발생시킨다. 발생한 키 값을 블루투스 통신모듈을 통해 키 이벤트를 Tablet PC나 Smart Phone에 전송하여 입력을 처리한다. 타깃 기기는 Bluetooth 통신이 가능한 모든 기기에서 사용할 수 있다. Desktop 환경과는 다르게 일련의 모든 과정을 따로 GUI로 제공하지 않으며 좀 더 직관적이고 간편한 사용을 추구하여, 실제로 One chip화 되었을 때의 모델을 고려하여 제작하였다.

3. 프로젝트 기능 모듈

3.1 클래스 다이어그램



<클래스 다이어그램>

3.2 클래스 명세서

3.2.1 FingerKeyboard

전체 프로그램인 FingerKeyboard가 정의된 클래스이다. FingerKeyboard는 다음과 같은 속성과 행위를 갖는다.

3.2.1.1 속성

이름	IplImage* dstImage;
설명	카메라가 가져오는 이미지를 저장한다.

이름	ImageProcessor* imageProcessor;
설명	이미지 처리를 위한 객체

이름	Hand* userHand[2];
설명	사용자 손 객체

이름	MouseListener* mouseListener;
설명	사용자의 마우스 이벤트에 대한 처리 객체

이름	Camera* camera;
설명	카메라 객체

이름	static int captureFrame;
설명	현재 시간을 나타낸다.

이름	static CvRect selectedPaperArea;
설명	사용자가 선택한 종이 키보드 영역을 저장한다.

이름	static PaperKeyboard* paperKeyboard;
설명	종이 키보드 객체

이름	static CvHistogram* pHist;
설명	사용자 피부색에 대한 Histogram

이름	static IplImage* bgImage;
설명	배경이미지

3.2.1.2 행위

이름	int programSetUp(int camIndex);
설명	프로그램을 초기화 한다.

이름	void programRun();
설명	프로그램을 실행 시킨다.

이름	void programExit();
설명	프로그램을 종료한다.

3.2.2 PaperKeyboard

종이 키보드를 정의한 추상 클래스이다. PaperKeyboard를 상속받아 다른 형태의 키보드를 구현할 수 있다.

3.2.2.1 속성

이름	int pressedKey;
설명	종이 키보드에서 누른 키를 저장하기 위한 변수

이름	int holdKey;
설명	종이 키보드에서 누르고 있는 키를 저장하기 위한 변수

이름	KeyButton* keyButton;
설명	키 버튼을 저장하기 위한 변수

이름	static CvPoint2D32f keyboardCorner[4];
설명	종이 키보드의 코너를 저장하기 위한 변수

3.2.2.2 행위

이름	CvPoint2D32f getKeyButtonCorner(int index);
설명	종이 키보드의 키 버튼의 개수를 가져온다.

이름	int getKeyButtonCornerCount();
설명	종이 키보드에 존재하는 키 버튼의 모든 코너의 개수를 가져온다.

이름	void setKeyboardCorner(CvPoint2D32f* corner);
설명	검출한 종이 키보드의 코너를 저장한다.

이름	void setPaperKeyboardCorner(IplImage* srcImage, CvRect paperArea, MouseListener* mouseListener);
설명	종이 키보드의 코너를 검출한다.

이름	virtual void setKeyButton(IplImage* srcImage)=0;
설명	종이 키보드 종류에 따라 정의 가능한 가상 함수.

3.2.3 PaperKeyboard_TypeA

PaperKeyboard 클래스를 상속받아 PaperKeyboard_TypeA 클래스를 정의하였다. 종이 키보드 종류에 맞게 setBeyButton() 메서드에서 정의가 가능하다.

3.2.3.1 행위

이름	PaperKeyboard_TypeA();
설명	종이 키보드의 키 버튼의 개수를 초기화한다.

이름	void setKeyButton(IplImage* srcImage);
설명	종이 키보드를 정의하고 저장한다.

이름	void sortPaperKeyboardCorner();
설명	종이 키보드의 코너를 시계 방향으로 정렬한다.

이름	void cornerVerification(IplImage* srcImage);
설명	2차 원근변환을 실시 후, 검출한 코너를 1차 원근변환 이미지에 대응시킨다.

이름	void initKeyButtonCorner()
설명	정렬된 코너를 저장한다.

이름	void cornerSortingY(int startIndex, int cornerCount);
설명	코너를 Y를 기준으로 정렬한다.

이름	void cornerSortingX(int startIndex, int cornerCount);
설명	코너를 X를 기준으로 정렬한다.

이름	void detectKeyButtonCorner(IplImage* srcImage);
설명	이미지에서 특징이 되는 점을 검출한다.

이름	void setKeyButtonArea(CvPoint2D32f* corners, int startIndex, int keyCount);
설명	키 버튼의 위치, 영역을 설정하고 저장한다.

이름	void setDirectionKeyButtonArea(CvPoint2D32f* corners, int startIndex, int next, int index);
설명	방향키 버튼의 위치, 영역을 설정하고 저장한다.

이름	void showKeyButton(IplImage* srcImage);
설명	설정된 키 버튼을 출력한다.

3.2.4 ImageProcessor

영상을 변환하고 처리해주는 클래스이다. 대부분의 연산이 이 클래스의 객체가 수행한다.

3.2.4.1 속성

이름	bool inputAvailable;
설명	입력 가능 시간을 나타내는 변수

이름	double inputStartPoint;
설명	입력을 시작한 시간을 저장한다.

이름	int pressFinger;
설명	사용자가 키를 누른 손가락을 저장한다.

3.2.4.2 행위

이름	void paperAreaDraggingImage(IplImage* srcImage, MouseListener* mouseListener);
설명	사용자가 마우스 Dragging을 할 때, Window에 Dragging 되고 있는 영역을 나타낸다.

이름	void createSkinColorHistogram(IplImage* srcImage);
설명	영역 안에 있는 색에 대한 Histogram을 생성한다.

이름	void detectHandContour(IplImage* srcImage, PaperKeyboard* paperKeyboard, Hand* userHand[]);
설명	손의 윤곽선을 찾고, 사용자가 누른 키를 결정한다.

이름	void getHandBinaryImage(IplImage* srcImage);
설명	사용자 손에 대한 이진 이미지를 얻는다.

이름	void determinSingleHandFingerTip(Hand* userHand);
설명	한 손이 검출됐을 때, 손 끝점을 찾아 저장한다.

이름	void determinDoubleHandFingerTip(Hand* userHand[]);
설명	두 손이 검출됐을 때, 손 끝점을 찾아 저장한다.

이름	bool keyEvent(CvPoint fingerLocation,int index, PaperKeyboard* paperKeyboard);
설명	손가락 위치에 해당하는 키 버튼의 이벤트를 발생시킨다.

3.2.5 Cemera

카메라를 정의한 클래스이다. 카메라의 해상도, 카메라의 밝기, 노출, 대비등을 설정할 수 있다.

3.2.5.1 속성

이름	int resolutionWidth;
설명	해상도의 폭을 저장한다.

이름	int resolutionHeight;
설명	해상도의 높이를 저장한다.

이름	CvCapture* camCapture;
설명	카메라로부터 입력 받은 영상의 속성들을 설정하기 위한 구조체이다.

3.2.5.2 행위

이름	CvSize getResolution();
설명	현재 영상의 해상도를 얻어온다.

이름	void setResolution(int width, int height);
설명	영상의 해상도를 매개 변수 width와 height로 설정한다.

이름	void setCamera(int deviceIndex);
설명	매개변수 deviceIndex에 해당하는 카메라로부터 입력받도록 설정한다.

이름	IplImage* getQueryFrame();
설명	영상의 한 프레임을 얻어서 IplImage* 타입으로 리턴한다.

이름	void setVideo(char* fileName);
설명	매개변수 fileName의 주소에 해당하는 동영상파일로부터 입력받도록 설정한다.

3.2.6 Mouse

마우스를 정의한 클래스이다.

3.2.6.1 속성

이름	bool bLButtonDown;
설명	이 속성의 값이 true인 경우에는 마우스 좌 클릭이 눌린 상태를 나타내고, false인 경우에는 마우스 좌 클릭이 눌리지 않은 상태를 나타낸다.

이름	bool bRButtonDown;
설명	이 속성의 값이 true인 경우에는 마우스 우 클릭이 눌린 상태를 나타내고, false인 경우에는 마우스 우 클릭이 눌리지 않은 상태를 나타낸다.

3.2.6.2 행위

이름	void setLButtonDown();
설명	bLButtonDown의 값을 true로 설정한다.

이름	void setLButtonUp();
설명	bLButtonDown의 값을 false로 설정한다.

이름	bool getLButtonDownState();
설명	bLButtonDown의 값을 리턴한다.

이름	bool getRButtonDownState();
설명	bRButtonDown의 값으 리턴한다.

이름	void setRButtonUp();
설명	bRButtonDown의 값을 false로 설정한다.

이름	void setRButtonDown();
설명	bRButtonDown의 값을 true로 설정한다.

3.2.7 MouseListener

사용자의 마우스 이벤트를 처리하기 위한 클래스이다. 마우스 객체를 갖으며, 프로그램 상태에 따라 동작을 처리한다.

3.2.7.1 속성

이름	static Mouse* mouse;
설명	마우스 객체

이름	static CvRect mouseDragArea;
설명	사용자가 선택한 영역을 저장하기 위한 변수

이름	static CvPoint originPoint;
설명	사용자가 처음에 클릭한 점을 저장하기 위한 변수.

3.2.7.2 행위

이름	MouseListener();
설명	마우스 객체를 생성하여 초기화 한다.

이름	CvRect getMouseDragArea();
설명	사용자가 선택한 영역을 가져온다

이름	Mouse* getMouse();
설명	마우스 객체를 가져온다.

이름	bool isMouseDragSize();
설명	사용자 선택한 영역이 유효한지 판단한다.

이름	void resetMouseDragArea();
설명	사용자가 선택한 영역을 초기화 시킨다.

이름	static void mouseClickEvent(int mEvent, int x, int y, int flags, void* param);
설명	사용자가 Window에 발생시키는 마우스 이벤트를 처리하기 위한 Call Back Method.

3.2.8 Keybutton

3.2.8.1 속성

이름	KeyId keyId;
설명	키버튼의 ID를 나타내는 enum 타입의 속성이다.

이름	KeyState keyState;
설명	키버튼의 상태를 나타내는 enum 타입의 속성이다.

3.2.8.2 행위

이름	void setKeyId(KeyId keyId);
설명	키버튼의 ID를 매개변수 keyId 값으로 설정한다.

이름	void setKeyLocation(int x, int y, int width, int height);
설명	키버튼의 영역에 대해 왼쪽 상단을 기준으로 x 좌표를 매개변수 x값으로 설정하고, y 좌표를 매개변수 y값으로 설정하고, 폭을 매개변수 width값으로 설정하고, 높이를 매개변수 height값으로 설정한다.

이름	void keyAction();
설명	키 이벤트를 발생시키는 함수이다.

이름	void setPress();
설명	키버튼의 상태를 눌린 상태로 설정한다.

이름	void setNone();
설명	키버튼의 상태를 눌리거나 잡혀있거나 떼어지지도 아닌 상태로 설정한다.

이름	void setHold();
설명	키버튼의 상태를 잡힌 상태로 설정한다.

이름	KeyState getKeyState();
설명	키버튼의 현재 상태를 리턴한다.

이름	KeyId getKeyId();
설명	키버튼의 ID를 리턴한다.

이름	void setRelease();
설명	키버튼의 상태를 떼어진 상태로 설정한다.

3.2.9 Finger

손가락에 대한 클래스를 정의하였다. 손가락의 위치와 이전위치, 방향성, 유효성, 움직임 정도, 움직임 누적치를 속성으로 갖으며, 이 속성들은 키 이벤트를 결정하는 요소가 된다.

3.2.9.1 속성

이름	CvPoint currFingerTip;
설명	현재 프레임에 대한 손 끝점의 x, y 좌표를 갖고 있는 CvPoint 구조체를 저장한다.

이름	Finger* prevFinger;
설명	이전 프레임에 대한 손가락 객체의 주소 값을 저장한다.

이름	bool downOrientMotion;
설명	이 속성의 값이 true인 경우 모션벡터의 방향이 아래임을 나타내고, false인 경우 모션벡터의 방향이 위임을 나타낸다.

이름	bool validMotion;
설명	이 속성의 값이 true인 경우 모션벡터의 값이 유효함을 나타내고, false인 경우 유요하지 않음을 나타낸다.

이름	double motionVector;
설명	모션벡터의 크기를 저장한다.

이름	double motionVectorSum;
설명	이전 프레임의 모션벡터와 현재 프레임의 모션벡터의 합을 저장한다.

3.2.9.2 행위

이름	bool inButton(KeyButton button);
설명	이 값이 true인 경우 현재 손가락의 좌표가 매개 변수 button에 해당하는 키버튼의 영역에 있음을 나타내고, false인 경우 그렇지 않음을 나타낸다.

이름	bool isMotion();
설명	손가락이 입력동작을 취했는지 아닌지를 판단하여 진리값을 리턴한다.

이름	void setMotionVector();
설명	모션벡터의 크기와 누적된 모션벡터의 크기와 모션벡터의 방향을 설정한다.

이름	void setValidMotion();
설명	모션벡터가 유효한지 아닌지를 나타내는 validMotion 속성의 값을 설정한다.

이름	void initAttribute();
설명	이전 프레임의 손가락 객체의 현재 손 끝점 좌표와 모션 벡터에 대한 속성 값들을 초기화한다.

이름	void setPrevFinger();
설명	이전 프레임의 손가락 객체를 생성한다.

이름	void fingerCopy(Finger finger);
설명	손가락 객체의 현재 손 끝점 좌표와 모션 벡터에 대한 속성 값들을 이전 손가락 객체에 그대로 복사한다.

이름	void resetFinger();
설명	이전 프레임의 손가락 객체의 손 끝점 좌표와 모션 벡터에 대한 속성 값들을 맨 처음 상태의 값으로 설정한다

3.2.10 Hand

손에 대한 클래스를 정의하였다. 손은 다섯 개의 손가락 객체를 갖으며, 이전에 발견한 손가락의 개수, 현재 발견한 손가락의 개수를 속성으로 갖는다.

3.2.10.1 속성

이름	Finger* finger[5];
설명	손에 대한 손가락 객체 5개를 갖는다.

이름	int detectFingerCount;
설명	검출된 손가락의 개수를 저장한다.

이름	int prevDetectFingerCount;
설명	이전 Frame에서 검출된 손가락의 개수를 저장한다.

3.2.10.2 행위

이름	bool isPressKey(KeyButton button);
설명	button을 손이 누르고 있는지 판별한다.

이름	int determinMotion();
설명	키 버튼을 누른 움직임을 한 손가락의 인덱스 번호를 반환한다.

이름	void setFingerMotionVector();
설명	손가락의 움직임 정도를 구한다.

이름	bool getAllFingerDownMotion();
설명	손 자체가 움직이고 있는지 판별한다.

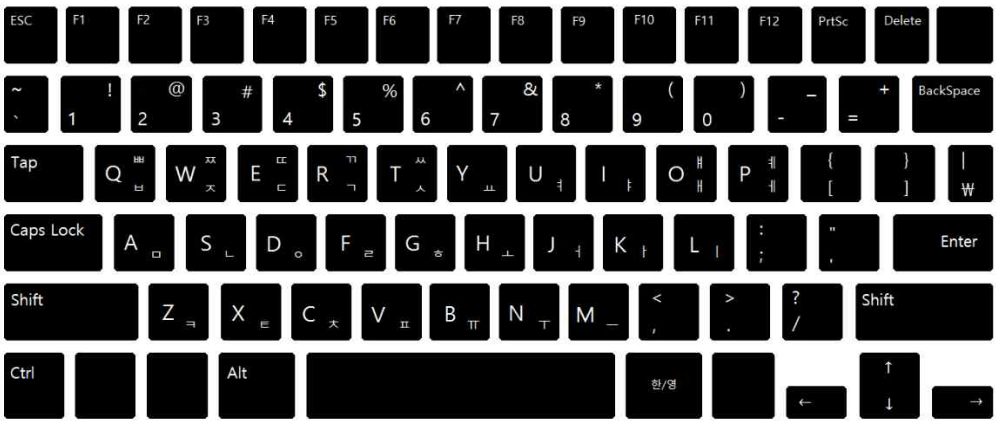
이름	void getHandDefect();
설명	이미지의 결함에 대해서 시작점, 끝점, 결점을 구한다.

이름	int getDefectTotal();
설명	윤곽선에서 검출한 결점의 개수를 반환한다.

이름	void convertArray();
설명	cvSeq 타입의 데이터를 배열로 변환한다.

4. 장비

4.1 종이 키보드



<종이 키보드>

위와 같이, A4용지 규격의 종이에 일반적인 노트북 키보드를 본떠 종이 키보드를 제작하였다. 키 버튼의 개수는 총 79개이며, 영어와 한글을 지원한다.

4.2 웹캠

Logitech C270 모델을 이용했다. USB 2.0을 이용하여 타깃 머신과 통신하며, 1280 x 720 해상도와 300만 화소를 지원한다.

4.3 라즈베리 파이

영국의 라즈베리 파이 재단이 학교에서 기초 컴퓨터 과학 교육을 증진시키기 위해 만든 싱글 보드 컴퓨터이다. 새로 출시한 2 모델 B는 ARM Cortex-A7 0.9GHz프로세서와 램 용량이 1GB로 성능이 업그레이드되어 출시되었다.

4.4 라즈베리 전용 캠

라즈베리 파이 전용 카메라 모듈인 Pi Cam을 이용하여 Embedded 환경에서의 프로그램을 동작시켰다. CSI(Camera Serial Interface) 소켓을 통해 연결되기 때문에 USB 통신보다는 빠른 영상 처리를 할 수 있으며, 500만 화소 640x480 해상도를 지원한다.

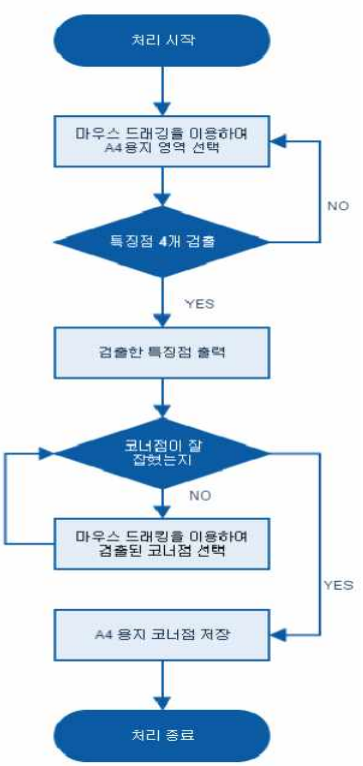
5. 프로그램 구동 절차

FingerKeyboard 프로그램은 카메라로부터 영상 이미지를 받아오며, 받아온 영상이미지를 현재 프로그램의 상태에 따라 처리하고, 출력하는 동작을 한다. 프로그램의 동작은 크게 종이 키보드를 인식하는 작업과, 사용자의 손을 인식하고, 사용자의 움직임을 분석하여 키 이벤트를 발생시키는 작업으로 이뤄져있다.

이 장에서는 작업에 따른 처리 절차와, 그 과정에서 얻을 수 있는 이미지, 객체간의 통신 다이어그램으로 프로그램 구동 절차를 설명한다.

5.1 종이 키보드 영역 검출

카메라에서 가져온 이미지로부터 종이 키보드 영역을 검출하는 작업을 수행한다. 이 작업에서는 사용자가 마우스를 이용하여 종이 키보드 영역을 선택하고, 선택된 영역에서 종이 키보드의 코너를 구하여 종이 키보드 영역을 인식한다.



<종이 키보드 영역 검출에 대한 Flow Chart>

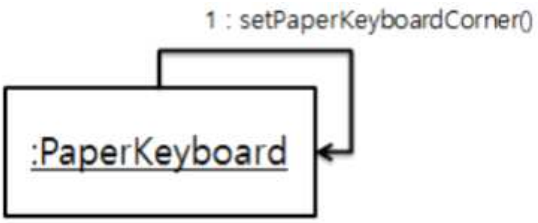
5.1.1 사용자의 A4용지 영역 선택

먼저, 사용자가 마우스 Dragging을 이용하여 종이 키보드가 위치한 영역을 선택한다. 카메라가 잡는 영역이 넓기 때문에, 환경에 종속적이고, 종이 키보드만 온전히 검출하는데 어려움이 따른다. 따라서 다른 방해물을 고려하지 않기 위해, 사용자가 종이 키보드 영역을 선택한다. 어차피 이후 키 버튼을 추출하는 작업, 사용자의 키 이벤트를 처리하는 작업 모두 종이 키보드 영역 안에서 일어나기 때문에 종이 키보드에만 관심을 갖는다.

<사용자는 마우스 Dragging을 통해 종이 키보드가 존재하는 영역을 선택>

5.1.2 종이 키보드 코너 검출

선택된 종이 키보드 영역에서 특징이 되는 점, 즉 용지의 각 코너를 검출하는 작업을 수행한다. 이 작업은 이후, 종이 키보드를 원근 변환하는 작업에서 필요한 데이터이므로 올바르게 검출해야한다.



<종이 키보드 코너 검출을 위한 통신 다이어그램>

PaperKeyboard 클래스의 객체는 setPaperKeyboardCorner() 메서드를 이용하여, 전 작업에서 선택된 종이 키보드 영역 안에서 특징이 되는 점 4곳을 검출하여 저장한다. 만약 4곳이 검출되지 않았다면, 다시 키보드 영역을 설정한다.


<올바르게 검출>


<한 코너를 올바르게 검출하지 못함>

선택된 종이 키보드 영역에서 검출된 코너는 위와 같이 나타난다. 환경에 따라 코너가 올바르게 검출될 수도, 안될 수도 있기 때문에 종이 키보드의 코너를 사용자가 검증하는 작업이 필요하다.

5.1.3 종이 키보드 코너 검증

환경에 따라 종이 키보드 코너가 부정확하게 검출될 수 있다. 이때는 사용자가 마우스 Dragging을 이용하여 코너를 선택한 후, 올바른 위치로 옮길 수 있다.


<코너 선택>


<올바른 곳으로 이동>

5.2 키 버튼 검출

설정된 종이 키보드 영역에 존재하는 키 버튼의 위치와 영역 정보를 얻기 위한 처리를 한다. 이 작업에서 종이 키보드에 대한 이미지를 수차례 변환하고 연산하여, 이미지 데이터를 가공한다. 그리하여 키 버튼에 대한 위치, 영역 정보를 얻을 수 있게 되고, 키 버튼의 정보를 저장한다.

5.2.1 키보드 영역 영상 처리

5.2.1.1 1차 원근 변환

이전 종이 키보드를 검출하는 과정에서 코너 4곳을 검출했다. 이 코너는 원근 변환을 위해 사용된다. 아래 원본 이미지를 보면 사다리 꼴 형태로 나타난다. 이유는 카메라와 물체사이의 거리 때문인데, 멀리 있는 것은 작게 보이고, 가까이 보이는 것은 크게 보이는 원리이다. 이러한 이미지를 이용하여 키 버튼 영역을 검출한다면, 카메라 거리에 따라 이미지가 항상 바뀌기 때문에 일관성이 떨어진다.

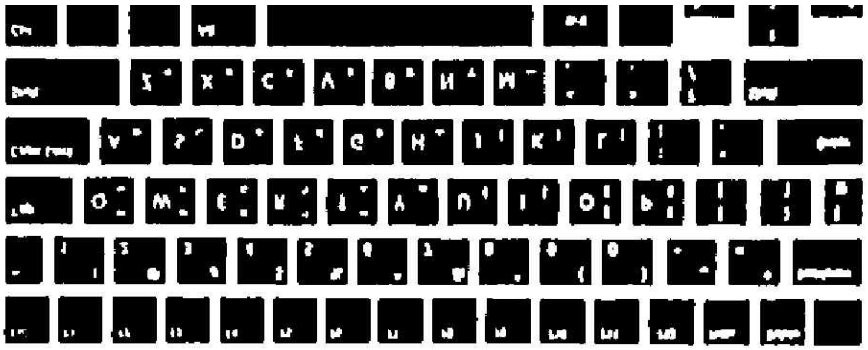
따라서 이미지를 일관성 있게 다루기 위해서 아래와 같이 원근 변환을 실시한다. 이는 카메라 거리에 영향을 받지 않고 키보드의 영역을 얻어낼 수 있기고, 종이 키보드가 기울어져 있더라고 일관성 있는 데이터를 얻을 수 있기 때문에 키 버튼을 검출하는 데에 효율적이다.

<원본 이미지>

<원근 변환 이미지>

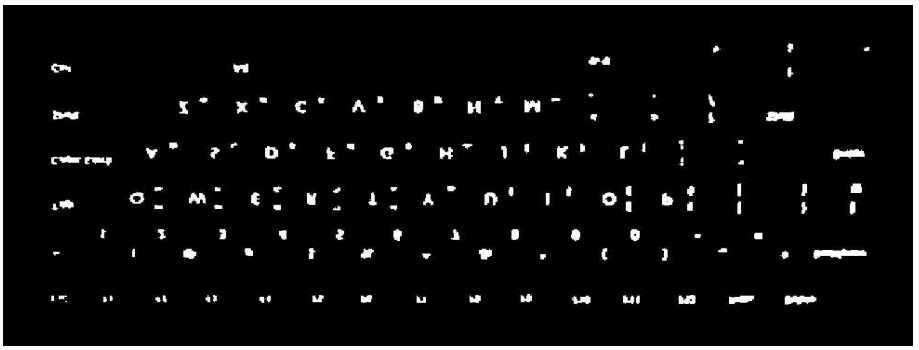
5.2.1.2 키 버튼 내부의 레이블 제거

원근 변환 된 이미지를 아래와 같이 이진 이미지로 처리하였다.



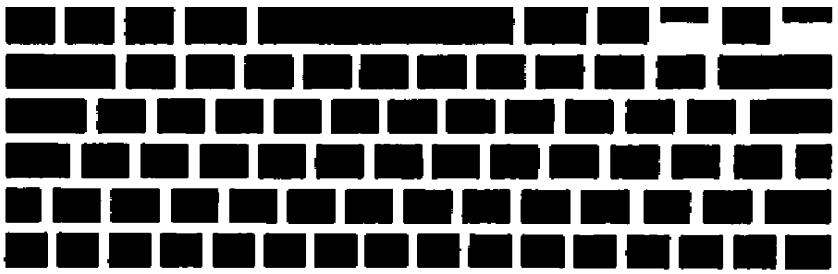
<종이 키보드의 이진 이미지>

하지만 키 버튼 안의 레이블로 인해 특징이 되는 점을 검출하는 데에 어려움이 있다. 그렇기 때문에 키 버튼 안의 하얀 부분을 제거할 필요가 있다.



<채움 연산 이후, 종이 키보드의 이진 이미지>

위와 같이 종이 키보드의 여백 부분을 cvFloodFill() 연산을 이용하여 제거하였다. 이미지는 키 버튼의 레이블에 대한 정보를 갖게 되었다. 이제 종이 키보드의 이진 이미지와, 키 버튼의 레이블에 대한 이미지를 Xor 연산을 하면 다음과 같은 이미지를 얻을 수 있다.

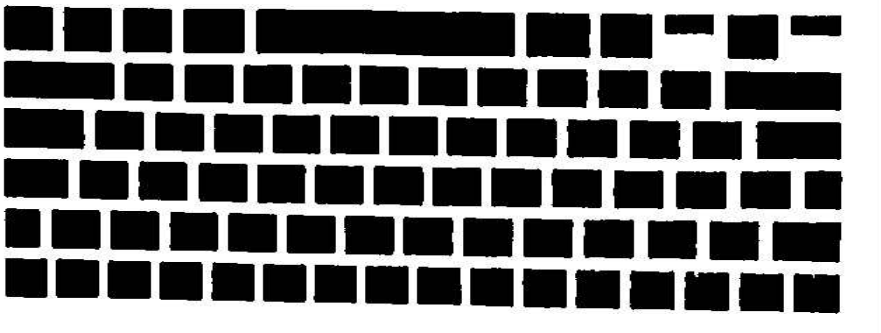


<두 이미지의 Xor 연산으로 키 버튼의 레이블이 제거된 종이 키보드의 이진 이미지>

이 이미지를 입력으로 하여 특징이 되는 점을 검출 할 수 있고, 이것은 종이 키보드의 키 버튼에 대한 정보를 얻을 수 있게 된다.

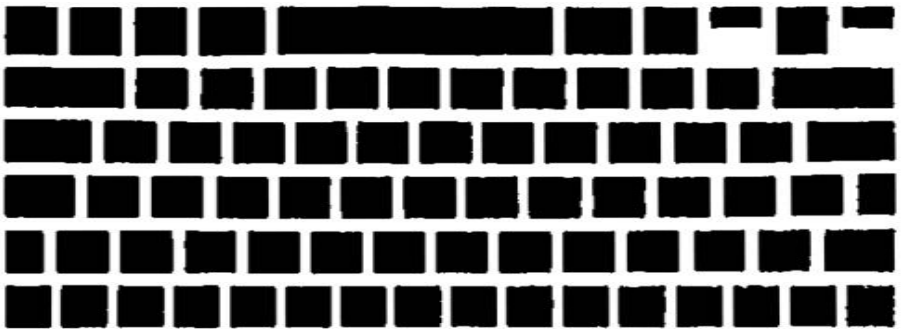
5.2.1.3 2차 원근 변환

사용자가 종이 키보드의 코너를 올바르게 설정하지 않은 경우, 원본 이미지에서 원근 변환 실시하면 다음과 같은 이미지를 얻게 된다.



<올바르지 않은 종이 키보드의 코너로 원근 변환 했을 경우>

그림에서 보이는 것처럼 키 버튼이 약간 기울어 진 것을 확인 할 수 있다. 위처럼 기울어 진 이미지에서는 키 버튼의 코너가 올바르지 못하게 정렬될 수 있다. 그렇기 때문이 기울어 진 이미지를 보정할 필요가 있고, 한 번 더 원근 변환을 실시한다.



<2차 원근 변환 이후 기울어짐이 제거된 종이 키보드>

이와 같이 원근 변환을 한 번 더 적용하면 기울어짐이 제거된 종이 키보드 이미지를 얻을 수 있다. 이제 이 이미지를 통해서 키 버튼의 코너를 검출한다.

5.2.2 실제 이미지에 대응

위에서 검출한 키 버튼의 코너는, 변환된 이미지에서의 좌표이기 때문에 실제 사용자가 보는 이미지에서의 키 버튼의 코너와 대응하지 않는다. 그렇기 때문에 검출한 코너를 실제 사용자가 보는 이미지에 대응시킬 필요가 있다.

$$\begin{bmatrix} t_i x_i' \\ t_i y_i' \\ t_i \end{bmatrix} = \mathtt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

<원근 변환 식>

앞에 원근변환의 원리는 원본 점과 변환 행렬의 행렬 곱 연산을 통해 변환 점을 얻을 수 있는 것이 원리이므로, 역으로 변환 점과 변환 행렬의 역행렬의 행렬 곱 연산을 이용하여 원본 점을 얻을 수 있고, 이것을 통해 실제 사용자가 보는 이미지에 대응하는 코너를 찾을 수 있다.

5.2.3 키 버튼 영역 저장

위에서 검출한 점을 이용하여 버튼 영역을 저장할 수 있다.

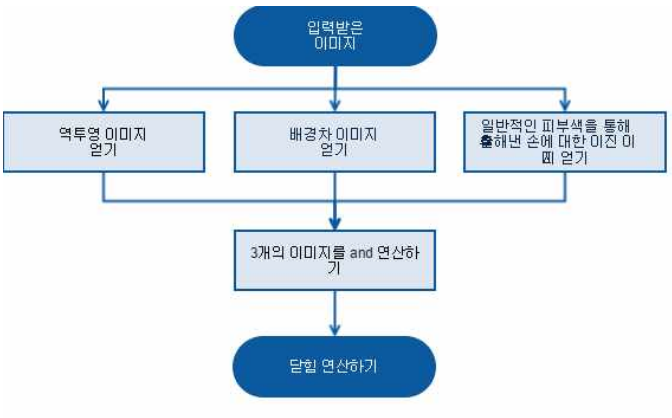
<검출된 키 버튼 출력>

5.3 사용자 손 검출

사용자의 피부색, 배경, 일반적인 사람의 피부색을 이용하여 손을 검출한다. 이후, 검출된 사용자의 손에서 손 끝점을 찾아내기 위한 작업을 수행한다.

5.3.1 손에 대한 영상 처리

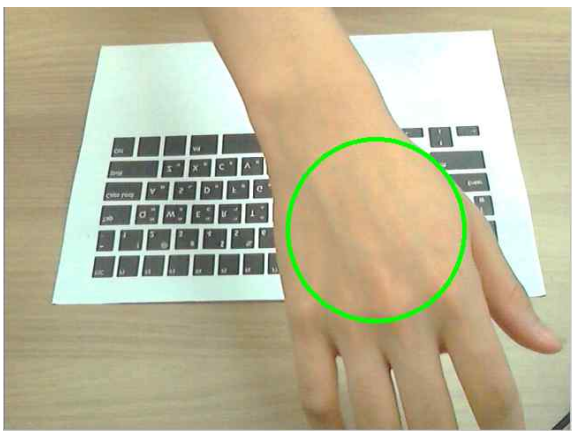
사용자의 피부색을 인식하여 사용자 피부색, 배경 이미지, 일반적인 사람의 피부색을 이용하영 각각의 손에 대한 이진 이미지를 얻고, 3가지의 이미지를 모두 And 연산을 하여, 사용자 손을 얻어낼 수 있다.



<사용자 손 검출을 위한 Flow Chart>

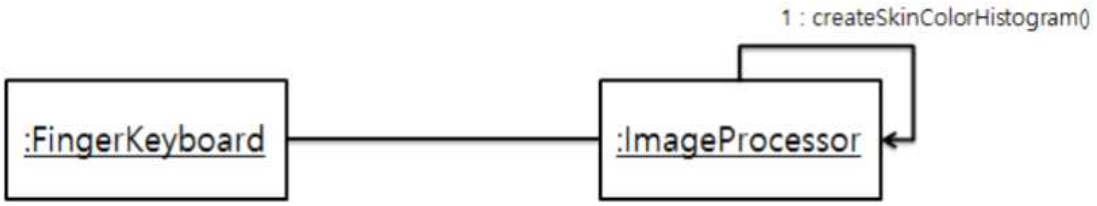
5.3.1.1 히스토그램을 이용하여 손 검출

프로그램 동작 중에 사용자의 피부색을 인식시킨다.



<사용자 피부색 인식>

초록색 원 안의 색의 분포를 이용하여 Histogram 생성하고, 히스토그램을 이용하여 손에 대한 역 투영 이미지를 얻는다.



<통신 다이어그램>

ImageProcessor 클래스의 객체는 createSkinColorHistogram() 메서드를 호출하여, 초록색 원 안의 색의 분포를 계산하여 FingerKeyboard 클래스의 객체에게 넘겨준다. 얻은 Histogram을 이용하여 색에 대한 분포를 나타낸 이미지는 다음과 같다.

<Histogram에 대한 색 분포 이미지>

5.3.1.2 일반적인 피부색으로 손 검출

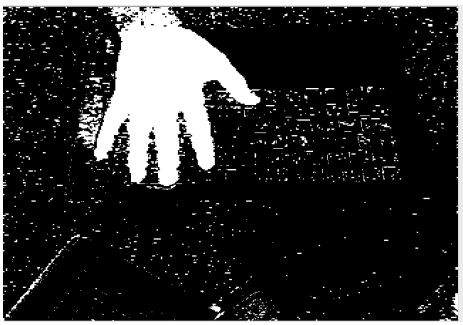
입력받은 영상에서 일반적인 사람의 피부색 범위에 있는 색을 다음과 같이 나타낸다.

<일반적인 사람의 피부색으로 검출>

그러나 종이 키보드 외부의 배경이 사용자 피부색과 비슷하기 때문에 위와 같은 이미지가 얻어진다.

5.3.1.3 배경과의 차이를 이용하여 손 검출

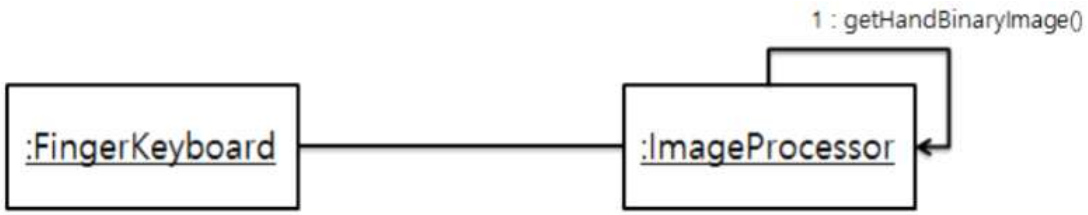
기준이 되는 배경 이미지와 현재 이미지와의 차이를 이용하여 배경 차 이미지를 얻는다. 이것은 배경이 사용자의 피부색과 비슷한 색일 수 있기 때문에, 필요한 작업 배경을 제거해 주기 위해서 생성한다.



<배경과의 차이를 이용한 손 이미지 검출>

5.3.1.4 손 검출

위에서 얻은 3가지 이미지를 이용하여 손에 대한 이미지를 얻을 수 있다. 그리고 노이즈를 없애기 위해 닫기 연산(팽창연산 후 침식연산) 한 번 더 수행한다.



<통신 다이어그램>

ImageProcessor 객체가 getHandBinaryImage() 메서드를 이용하여, 사용자 손에 대한 이진 이미지를 생성한다.

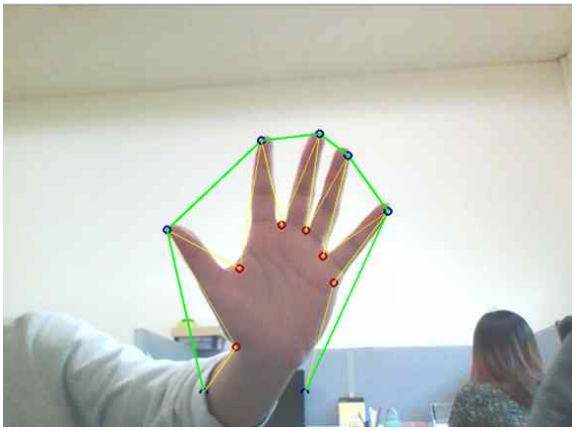
<추출한 손에 대한이진 이미지>

5.3.2 손 끝점 검출

생성한 이진 이미지에서 사용자의 손 끝점이라고 판단되는 점을 검출한다.

5.3.2.1 손 외곽선

손의 이진 이미지로부터 손에 대한 외곽선을 얻어낸다. 이후 윤곽선 근사화 작업, 손의 결점 검출 작업을 수행한 이후에 아래와 같은 이미지를 얻어 낼 수 있다.



<손에 대한 윤곽선>

5.3.2.2 손 끝점 결정

계산된 손의 영역의 크기의 범위에 따라 정상적으로 한 손이 잡혔는지, 두 손이 잡혔는지 또는 손이 겹쳐졌는지, 완전히 손이 영역을 벗어났는지 판단한다. 그리고 일차적으로 얻은 결점들에 대해서 손가락 끝 부분으로부터 일정 크기 이상 떨어진 경우, 그 점을 실제 손의 결점으로 결정하고 그 점에 대한 시작점을 손 끝점으로 결정한다.

<검출된 손 끝점>

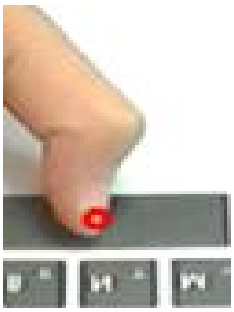
5.4. 사용자 움직임

위와 같이 손에 대한 윤곽선에 대해서 손 끝점을 검출하여 각 위치를 저장하였다. 이제 손 끝점의 움직임을 계산하여, 사용자가 눌렀다고 판단되는 키 버튼의 이벤트를 발생시킨다.

5.4.1. 손가락

5.4.1.1 유효한 손가락의 움직임

검출한 손 끝점의 위치가 유효하지 않게 되는 상황 발생.

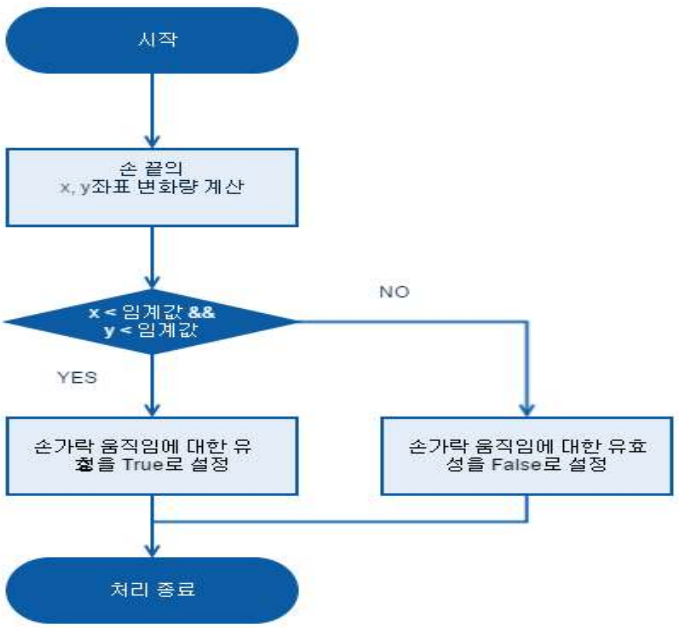

<t = 1>


<t = 2>


<t = 3>

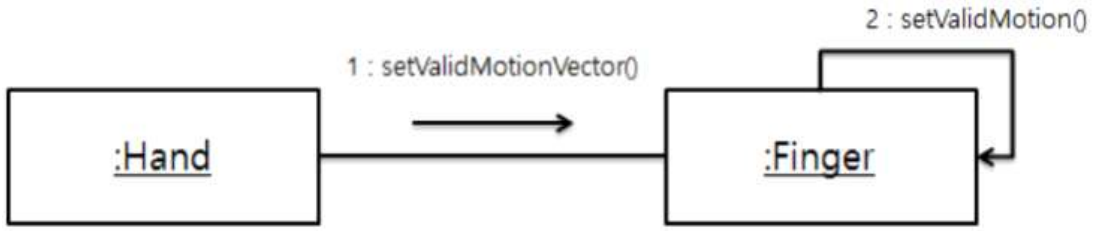
손가락의 끝 점이 올바르게 추출된다면 좋겠지만, 손끝이라고 판단되는 다른 것이 물체가 존재한다면, 손가락의 끝 점을 올바르게 추출하지 못할 수 있다. 그렇기 때문에 위와 같이 손가락의 끝점이 잘못 검출될 수가 있고 이런 손끝의 변화로 인해 의도하지 않은 키 이벤트가 발생할 수도 있다.

또 화면 밖에서 사용자의 손이 들어올 경우, 손 자체가 빠르게 들어오기 때문에 움직임 정도가 유효하지 않을 수 있다.. 이것을 제거하기 위해, 유효한 손가락의 움직임을 판단하는 작업이 필요하다.



<유효한 손가락 움직임 판별을 위한 Flow Chart>

Hand 클래스의 Instance인 userHand에는 Finger 클래스의 instance가 5개 존재한다. 각각의 Finger 객체는 setValidMotion() 메서드를 호출한다. setValidMotion() 메서드는 손가락의 움직임이 유효한지, 유효하지 않은지 판별하는 절차로, x의 변화량, y의 변화량이 특정 임계값보다 많이 움직였다면, 유효하지 않은 움직임이라고 판단하고 Finger 클래스의 객체의 속성인 validMotion은 false로 설정한다. 유효한 움직임이라면 validMotion을 true로 설정한다.



<통신 다이어그램>

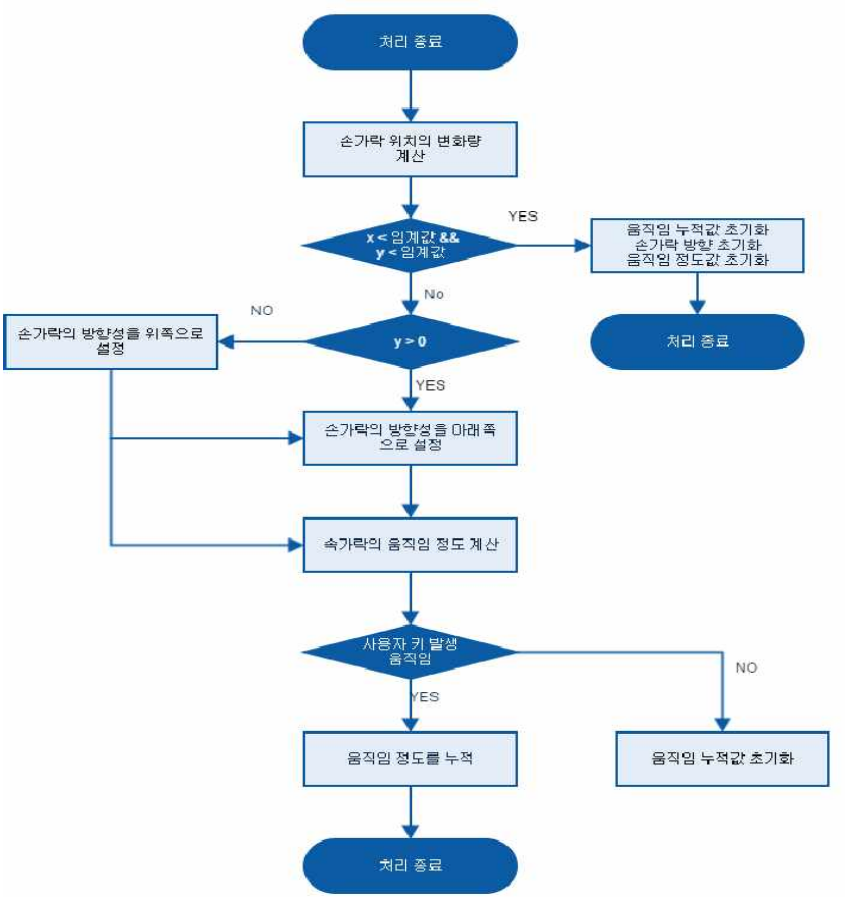
5.4.1.2 움직임 정도 계산

이전 Frame에서의 손가락의 좌표와 현재 Frame에서의 손가락의 좌표를 이용하여 손가락의 움직임 정도를 계산한다. 하지만 환경으로 인한 여러 가지 잡음으로 인하여 손끝의 위치가 조금씩 변경될 수 있다.

t = 1

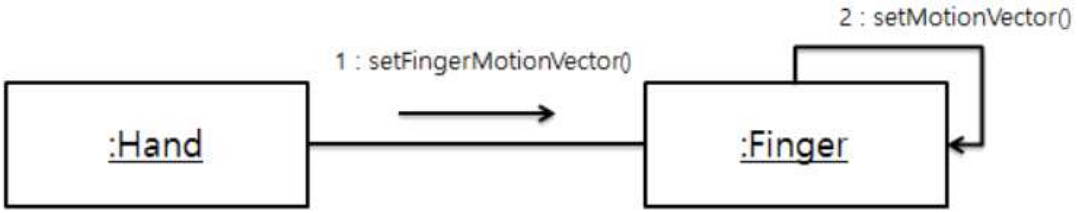
t = 2

위의 이미지와 같이 손을 움직이지 않았지만, 그림에서 보이는 첫 번째 손 끝점의 위치가 약간 변경된 것을 확인할 수 있다. 이런 잡음이 발생하면, 의도하지 않은 결과를 초래하기 때문에 먼저 잡음을 제거해준 후 처리하여야 한다.



<움직임 정도를 계산하기 위한 Flow Chart>

Hand 클래스의 Instance인 userHand에는 Finger 클래스의 instance가 5개 존재한다. 각각의 Finger 객체는 setMotionVector() 메서드를 호출한다. setMotionVector()에서 이전 Frame의 손가락 위치와 현재 Frame의 손가락 위치를 이용하여, 손가락 위치의 변화량을 구한다. 움직임 정도가 임계값보다 작다면 잡음으로 인한 값이라고 판단하고 작업을 종료한다. 아니라면 y의 변화량에 따라 움직임의 방향성을 판별하고, 움직임 정도를 계산한다. 이 움직임이 사용자가 키 버튼을 누른 움직임이라고 판단된다면, 움직임 정도를 누적시키고, 아니라면 이전에 누적되었던 움직임 누적치를 0으로 초기화 한다.

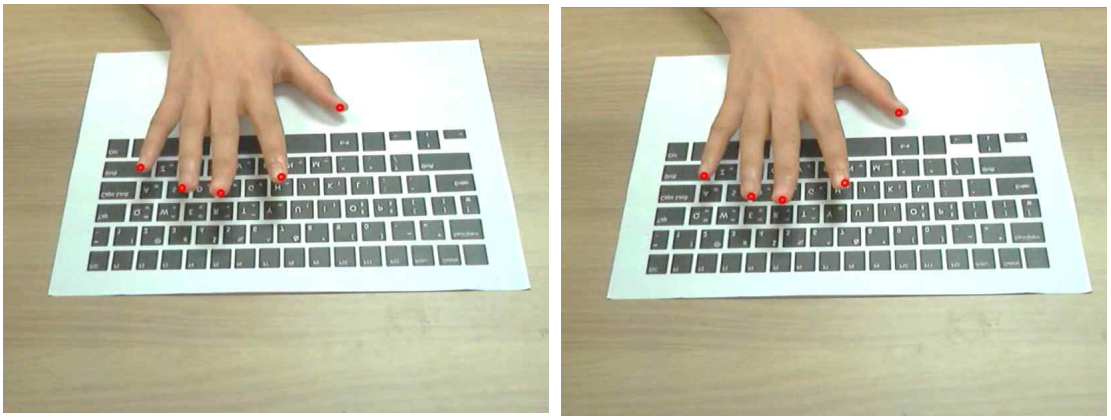


<움직임 정도 계산을 위한 통신 다이어그램>

5.4.2 손

앞의 과정에서는 손가락 하나하나를 고려했다면, 이제는 손에 대한 움직임을 판단한다. 앞에서 계산된 움직임의 유효성, 손가락의 방향성, 움직임 정도, 움직임 누적치가 판단하는 기준이 된다.

5.4.2.1 손의 움직임

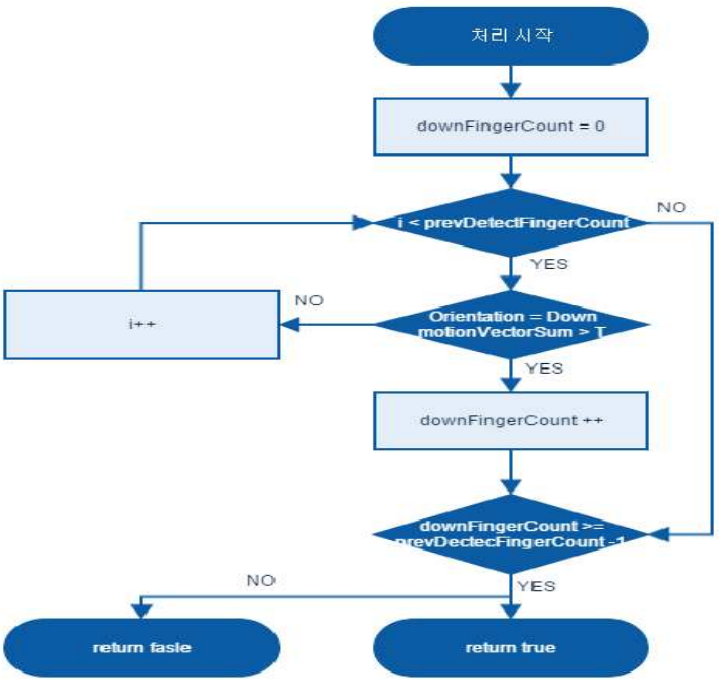


t = 1 t = 2

위와 같이 사용자가 입력을 위해 손을 이동하는 과정을 생각해보면, 손을 키보드에 올려놓는 작업을 할 것이다. 그 과정에서 손가락 전체가 움직이기 때문에, 프로그램은 사용자가 키 버튼을 눌렀다고 판단하여, 사용자가 의도하지 않은 키 이벤트가 발생될 수도 있다.

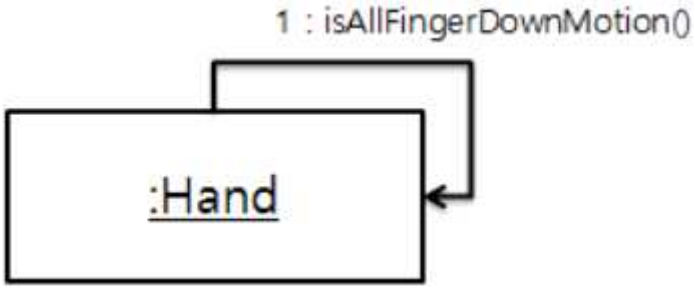
그렇기 때문에 손 자체가 움직이는 경우는 키 입력 과정에서 제외 시켜주어야 하며, 유효한 움직임을 한 손가락이 하나일 경우만 키 이벤트를 결정해야 한다.

다음은 손이 움직이는 경우라고 판단하는 절차이다.



<손이 움직이는지 판별하기 위한 Flow Chart>

Hand 클래스의 Instance인 userHand가 isAllFingerDownMotion() 메서드를 호출한다. 이 함수는 검출된 손가락이 대부분 움직일 경우라면, 키 이벤트를 처리해주지 않기 위한 메서드이다. 여기서 대부분의 손가락이란 이전 Frame에서 검출된 손가락 개수에서 하나 적은 수이며, 이 개수보다 크거나 같다면 손자체가 움직인 것이라고 판단하여, 키 이벤트를 고려하지 않는다.



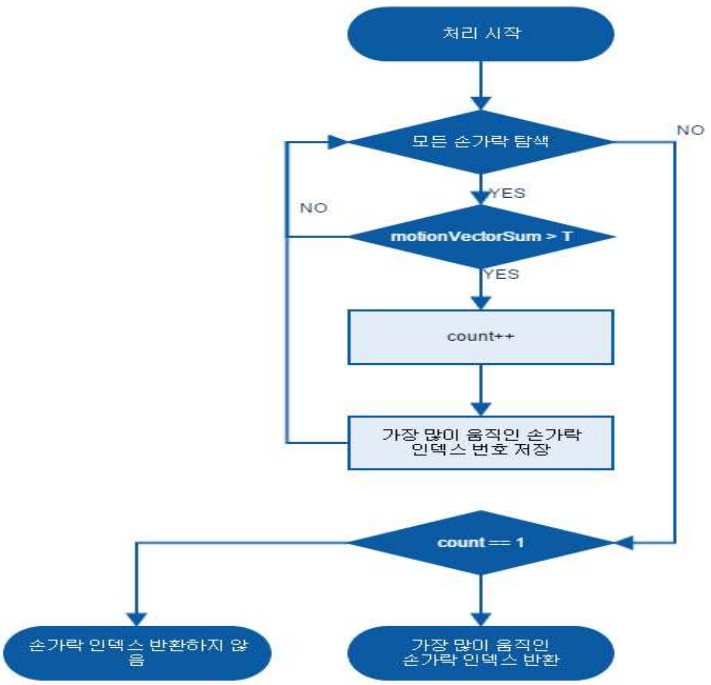
<통신 다이어그램>

5.4.2.2 누른 손가락 검출

최종적으로 사용자가 키 이벤트를 발생시킨 손가락을 검출한다. 우선 키 이벤트라고 판단되는 손가락의 개수를 계산한다. 이는 키 이벤트라고 판단되는 손가락이 하나일 경우만, 키 이벤트를 발생시켜주기 위함이다.

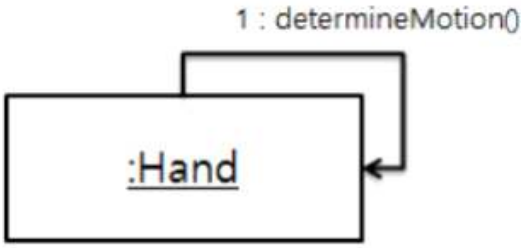
키 이벤트를 발생시키기 위해 사용자가 검지를 이용하여 자판을 눌렀다 하더라도, 중지나 약지가 더 많이 움직이는 경우가 발생하기 때문이다. 애초에 키 이벤트를 자판을 누른 것이 아닌, 손가락의 움직임 정도로 판단하기 때문에 이러한 모호성이 발생할 수 있다. 이 모호성을 제거해주기 위해서, 키 이벤트라고 판단되는 손가락이 2개 이상일 경우에는 키 이벤트를 처리해주지 않는다.

만약 키 이벤트라고 판단되는 손가락이 1개가 검출되었다면, 가장 큰 움직임을 한 손가락을 찾아서, 그 손가락 위치를 반환한다.



<누른 손가락을 검출하기 위한 Flow Chart>

Hand 클래스의 Instance인 userHand가 determineMotion() 메서드를 호출한다. 이 메서드는 키 버튼을 눌렀다고 판단되는 손가락의 인덱스 번호를 반환한다. 우선 모든 손가락을 탐색하여 키 버튼을 눌렀다고 판단되는 손가락의 개수와 가장 많이 움직인 손가락을 구한다. 만약 키 버튼을 눌렀다고 판단되는 손가락 개수가 1개라면 가장 많이 움직인 손가락 인덱스를 반환하며, 아니라면 반환하지 않는다.



<통신 다이어그램>

5.5 키 버튼

KeyButton 클래스의 instance인 keyButton은 자신의 고유 아이디와 상태를 갖는다. keyAction() 메서드에는 자신의 고유 아이디에 따른 키 이벤트가 정의되어 있으며, 키의 상태에 따라 다르게 동작한다. 일반적으로 Single Key에 대한 처리는 키를 누른 경우에만 발생하며, Modifier Key에 대한 처리는 키를 눌렀을 경우, 그리고 키를 뗐을 경우에 발생시킨다.

Target Machine이 Windows와 BlueTooth 통신을 하는 기기이기 때문에, Windows와 Raspbian 환경에서 다르게 작성되어야 한다. Windows 환경에서의 프로그램은 windows 라이브러리에서 제공하는 keybd_event를 이용하여 키 이벤트를 처리한다. Raspbian 환경에서의 프로그램은 pipe 통신을 통해 눌린 키에 대한 키 값을 BlueTooth 프로그램에게 전송하여 키를 처리한다.

5.5.1 키 버튼 상태

상태	설명
KEY_STATE_NONE	키 버튼의 초기 상태이다.
KEY_STATE_PRESS	키 버튼이 눌린 상태이다.
KEY_STATE_PRESS_HOLD	키 버튼이 눌려지고 있는 상태이다.
KEY_STATE_RELEASE	키 버튼이 떼진 상태이다.

5.6 키 이벤트 처리

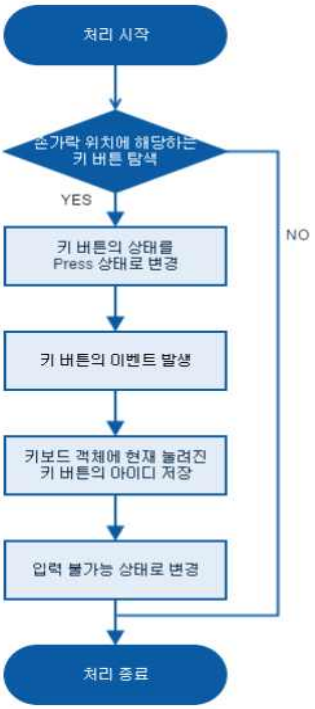
키 이벤트를 발생시키기 위해서는 손가락의 위치, 입력 가능 시간, 키 버튼의 상태가 고려된다. 이 요소들을 고려하여, 키를 눌렀을 때, 키를 누르고 있을 때, 키를 뗐을 때에 대한 처리를 수행한다.

5.6.1 입력 가능 시간

키 이벤트가 발생했다면, 일정 시간동안 키 이벤트를 발생시키지 않는다. 키 이벤트는 입력 가능 시간에만 발생하며, 사용자가 키를 누르고 있는 행동을 판단한다.

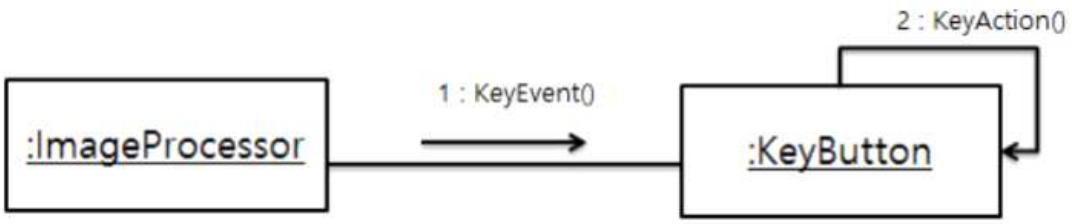
5.6.2 키를 눌렀을 경우

사용자가 키를 눌렀을 때에 대한 키 이벤트 처리이다. 손 끝점 위치를 이용하여 그 위치에 있는 키 버튼의 이벤트를 발생시킨다.



<키를 눌렀을 경우를 위한 Flow Chart>

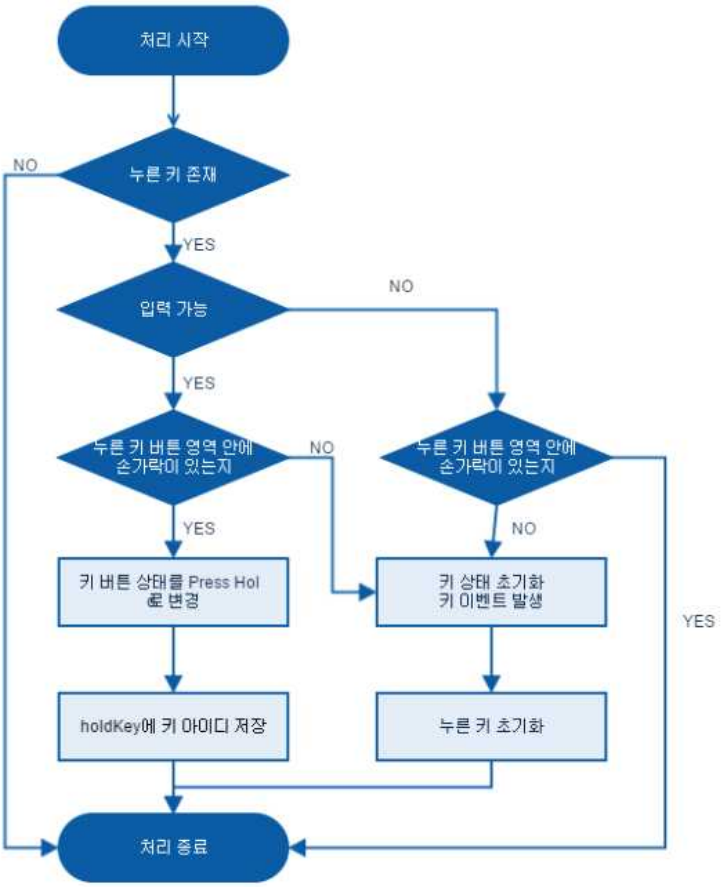
ImageProcessor 클래스의 객체는 사용자가 키를 눌렀다고 판단되면 KeyEvent() 메서드를 호출하여 키 이벤트를 발생시킨다. 사용자가 누른 위치에 있는 KeyButton 클래스의 객체는 자신의 KeyAction() 메서드를 호출하여 자신의 아이디와 상태를 보고 키 이벤트를 발생시킨다.



<객체 간의 통신>

5.6.3 키를 누르고 있을 때에 대한 처리

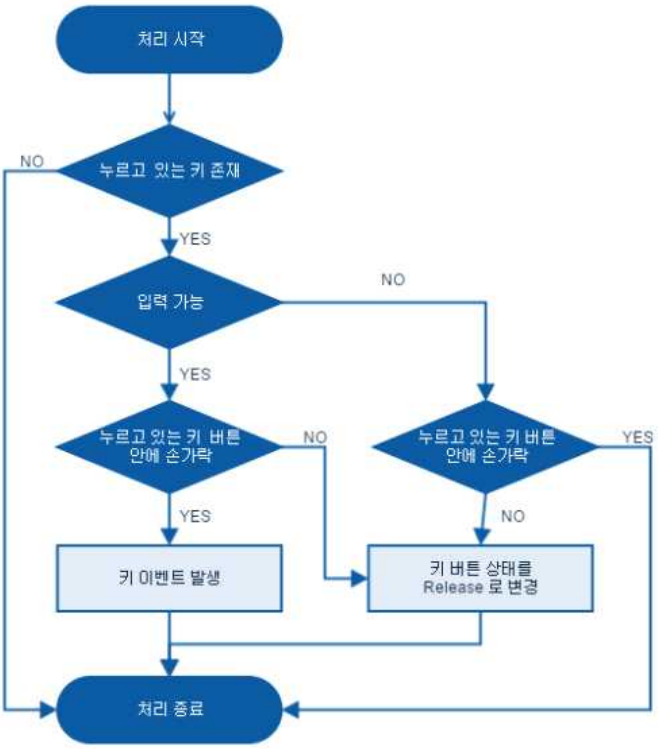
사용자가 키를 누르고 있는 상태에서 키 이벤트에 대한 처리이다. 이전에 눌렸던 키에 현재에도 손가락이 위치해 있다면 사용자가 키를 누르고 있다고 판단한다.



<키를 누르고 있을 때에 대한 Flow Chart>

5.6.4 키를 뗐을 때에 대한 처리

키 버튼의 자신이 어떤 키인지, 어떤 상태인지에 따라 키 이벤트를 다르게 발생시킨다. 예로, Shift Key, Control Key와 같은 Modifier Key는 누르고 있을 때는 적용하고 있다가, 뗐을 때는 다시 키 이벤트를 발생시켜 뗀 작업을 수행시켜주어야 한다. 다음은 사용자가 키를 뗐을 때에 대한 처리 과정을 보여주는 Flow Chart이다.

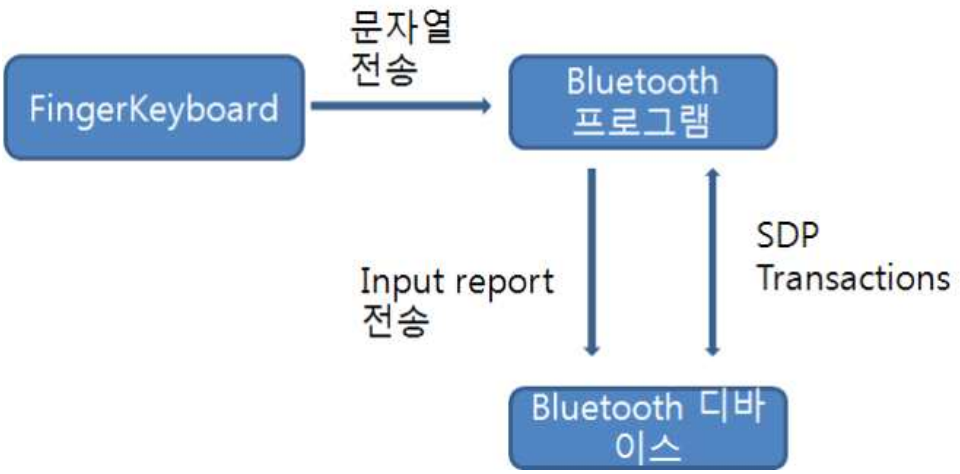


<키를 뗐을 때에 대한 Flow Chart>

5.7. BlueTooth 통신

5.7.1 Bluetooth 통신 구조

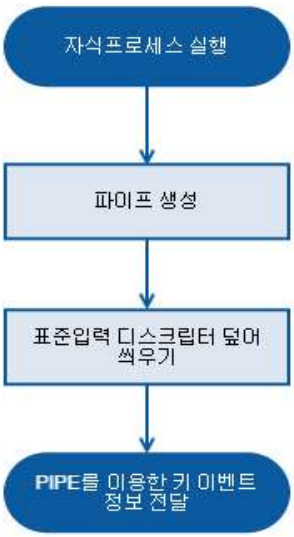
FingerKeyboard로부터 입력된 키에 대한 정보를 pipe를 통해 문자열 형태로 전송받고, 블루투스 통신 프로그램은 전송받은 문자열을 16진수 문자열의 Input report형태로 바꾸어서 Interrupt socket을 통해 키 이벤트를 전송시킴으로써 타깃 블루투스 장치에 키 이벤트가 발생되게 한다.



<Bluetooth 통신 구조>

5.7.1.1 FingerKeyboard로부터 키 값 정보 받아오기

부모프로세스인 FingerKeyboard가 자식프로세스인 블루투스 통신 프로그램을 실행시키고 자식프로세스의 표준입력 디스크립터에다가 파이프 출력 디스크립터를 덮어씌운다. 그리고 파이프 통신을 통해 발생된 키에 대한 정보를 문자열로 전달한다. 그러면 블루투스 통신 프로그램에서 무한 루프를 돌면서 계속 호출 되는 stdin.readline() 메서드를 통해 해당 문자열을 읽어올 수 있다.



<FingerKeyboard와 Bluetooth 프로그램의 Pipe 통신>

5.7.1.2 Raspberry Pi를 Bluetooth 키보드로 동작시키기

os.system() 메서드를 이용하여 hciconfig를 실행시킴으로써 라즈베리파이와 연결할 타깃 디바이스 사용자가 라즈베리파이를 블루투스 키보드처럼 보도록 할 수 있다. Device class와 Device name을 설정하고 page scan, inquiry scan을 수행하도록 한다.

sdp_record라는 xml파일을 읽은 다음, Dbus를 이용하여 Bluez의 객체를 얻어오고, 이것을 이용하여 읽어온 sdp record를 추가시킨다. 이제, sdp protocol을 통해 라즈베리파이가 실제 블루투스 키보드처럼 동작할 수 있다.

<Raspberry Pi를 Bluetooth 키보드로 동작시키기 위한 Flow Chart>

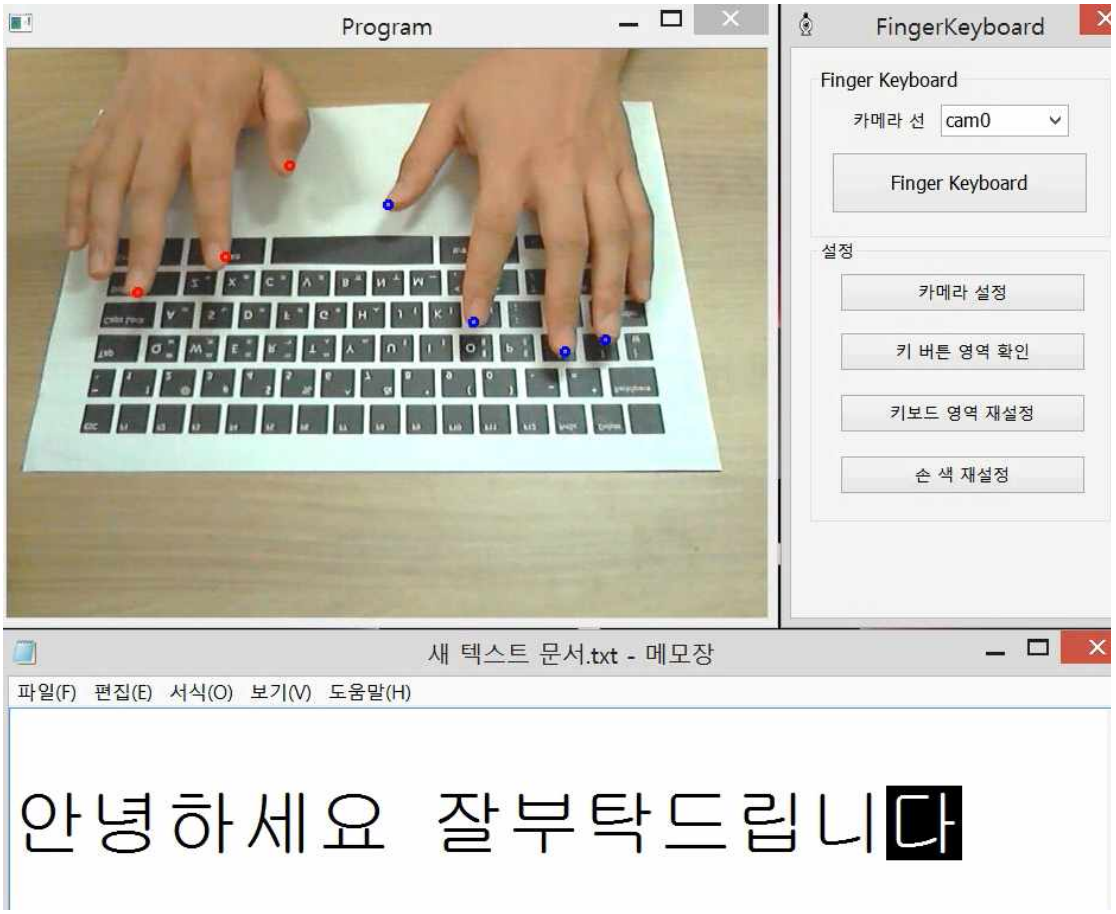
5.7.1.3 Bluetooth Device로 키 이벤트 전송

조건문을 이용하여 pipe로 읽어온 문자열에 대응하는 16진수 문자열형태의 Input report를 Interrupt socket을 통해 전송한다. (프로토콜은 L2CAP프로토콜을 사용) 이것으로 연결된 블루투스 장치에 키 이벤트를 발생시킬 수 있다.

<Bluetooth Device로 키 이벤트 전송>

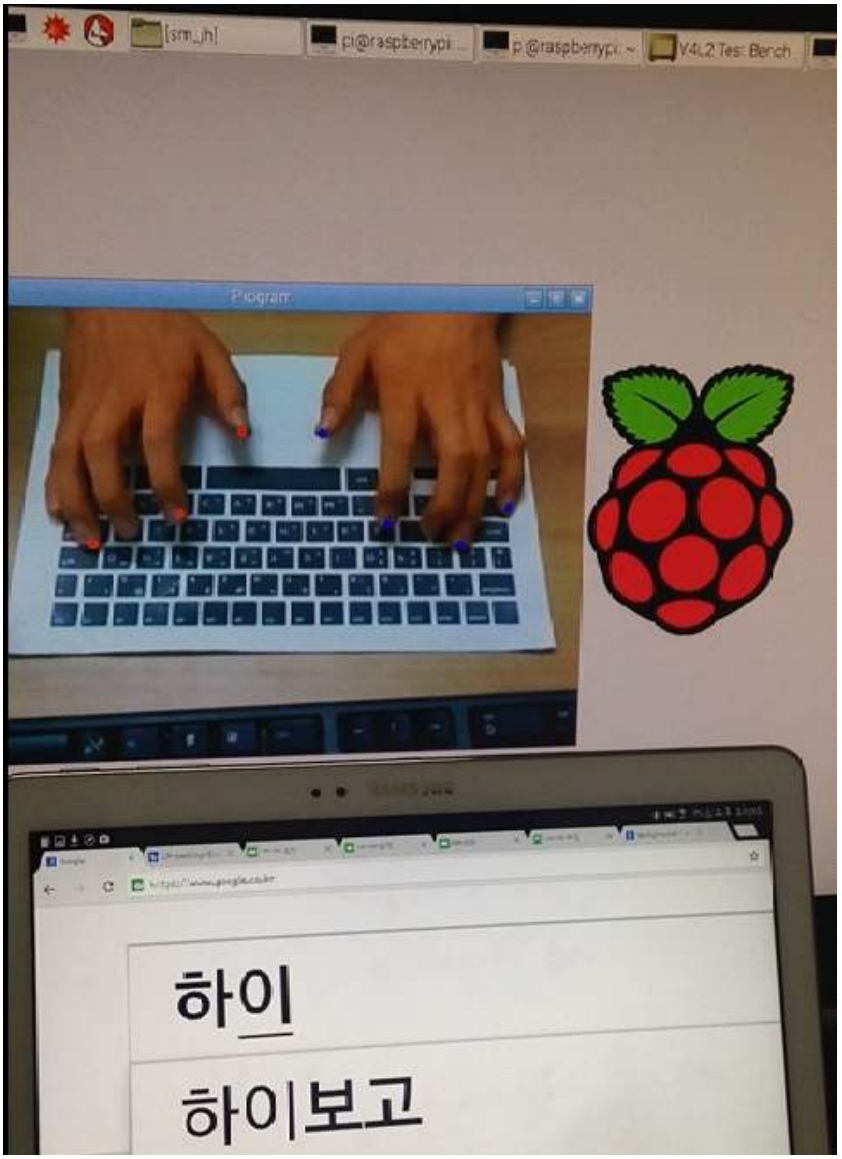
6. 구현 결과

6.1 Desktop 환경



<Windows 환경의 프로그램>

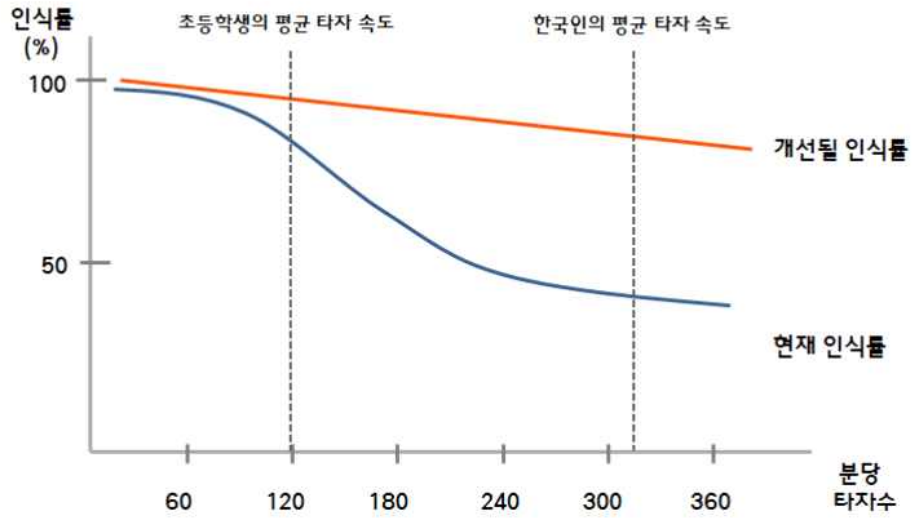
6.2 Embedded 환경



<Raspberry Pi와 Bluetooth 통신을 하는 기기>

7. 인식률

인식률 = (1− 인식오류 개수 / 타이핑한 문자 개수) × 100



<키 입력 인식률>

8. 개발 환경 및 도구

8.1 Windows

윈도우즈 운영체제는 가정용, 업무용 컴퓨터, 노트북 컴퓨터와 같은 일반 목적의 컴퓨터 시스템에서 사용할 수 있도록 마이크로소프트사가 개발한 운영체제이다. 현재 대중화 되어있기 때문에 개발환경을 구축하기 위한 운영체제로 Windows를 선택하였다.

8.2 파이 전용 Linux인 Raspbian

타겟 Embedded 환경인 Raspberry Pi 2에서 구동되는 운영체제이다. 리눅스 계열의 운영체제인 Debian을 라즈베리에 최적화한 운영체제이며 Raspberry Pi 2에 가장 최적화된 운영체제이다. 기본적으로 가장 많은 패키지가 설치되어 있으며 가장 기초적인 개발환경으로 널리 사용되기 때문에 Raspbian을 선택하였다.

8.3 Linux Ubuntu

Ubuntu는 Debian GNU/Linux에 기초한 컴퓨터 운영체제이다. Raspbian에서 구동되는 FingerKeyboard 프로그램의 Cross-compiler환경을 구축하기 위해 사용한다.

8.4 Microsoft Visual Studio 2012

마이크로소프트 윈도에서 작동하며, 다양한 언어로 프로그래밍 할 수 있는 마이크로소프트의 통합 개발 환경이다. 프로그램, 웹 사이트, 웹 프로그램 등을 개발할 수 있다. 마이크로소프트에서는 비주얼 베이직, 비주얼 C#, 비주얼 J# 등 특정한 언어로만 프로그래밍 할 수 있는 언어별 버전도 제공하고 있다.

9. 용어 설명

9.1 OpenCV

Open CV란 C와 C++로 작성된 컴퓨터 비전 라이브러리이다. 이 라이브러리는 동영상 파일이나 컴퓨터에 연결된 Web Cam으로부터 영상을 쉽게 가져올 수 있도록 하는 함수와 기본적인 GUI 관련 함수들, 그리고 영상처리에 있어서 자주 사용되는 morphology 연산, convolution 연산 등등 다양한 영상에 변화를 주는 기능의 함수들도 제공하고, 뿐만 아니라 기계학습에 관한 함수들도 제공한다. 즉, 컴퓨터 비전을 위한 다양한 기능들을 제공하고 있다. Open CV는 계속적으로 업데이트 되고 있고 학술 연구, 상업적 용도로 쓰이는 경우 모두 무료로 제공된다.

9.2 Cross-Compiler

원시프로그램의 번역이 이루어지는 컴퓨터와 번역된 기계어에 이용되는 컴퓨터가 서로 다른 기종의 컴퓨터일 때 사용하는 컴파일러의 한 가지이다. 어떤 컴퓨터에서 동작하는 프로그램을 만들기 위해 다른 컴퓨터의 개발 환경을 사용해서 프로그램을 작성하는 경우에 사용된다. 동작 속도가 느린 컴퓨터, 완성되어 있지 않은 컴퓨터, 개발 환경 구축이 불가능한 컴퓨터용의 실행프로그램을 만드는 경우 등에 사용한다. 우리의 경우는 동작속도가 느리고, 개발 환경 구축이 불가능하진 않으나 어려운 환경인 Raspberry Pi 2이므로 Cross-Compiler를 사용한다. Raspberry Pi 2에서 구동될 FingerKeyboard를 X86 CPU를 사용하는 Ubuntu에서 ARM A7 CPU를 사용하는 Raspberry Pi 2 용 프로그램으로 컴파일하기 위해서는 gnueabihf를 통해서 Cross-Compile을 한다.

9.3 DBUS

D-Bus는 프로세스간 통신 (IPC)을 위한 시스템이다. 오픈 소스 소프트웨어로서 리눅스 데스크톱의 프로세스 통신을 위한 수단으로 널리 사용되고 있다. 특히 KDE와 Gnome같은 데스크탑 환경에서 중요하게 사용된다.

9.4 SDP

SDP는 블루투스 디바이스가 제공하는 서비스를 찾고 응답하기 위한 프로토콜이다.

9.5 Input report

HID 디바이스가 데이터를 주고받을 때 데이터의 형식은 Report인데, Input Report는 interrupt 채널을 통해 HID 디바이스로부터 전송되어진다. 예를 들어, 키 이벤트 전송에 쓰인다.

9.6 Interrupt

HID 디바이스는 Control pipe와 Interrupt pipe를 통해 통신을 하는데, Interrupt의 경우에는 장치로부터 비동기(요청되지 않은) 데이터를 받을 때와 낮은 지연의 데이터를 전송할 때 쓴다.

9.7 page scan

블루투스 슬레이브 디바이스가 page scan을 통해 마스터 디바이스로부터의 page를 감지하고 그것에 응답해줌으로써 블루투스 연결이 이루어질 수 있게 한다.

9.8 inquiry scan

블루투스 슬레이브 디바이스가 inquiry scan을 통해 마스터 디바이스로부터의 inquiry를 감지하고 그것에 응답해줌으로써 자신을 마스터 디바이스의 주변기기 목록에 노출시킨다.

9.9 hciconfig

hciconfig는 블루투스 디바이스들을 구성하기위한 프로그램이다.