**B. A. R.**

(Building Find Argument Reality)

|  |
| --- |
| 만약 우리가 여행을 갔을 때 주변에 대한 건물과 지리정보에 대한 지식이 없다면 답답함과 낯선 곳에 대한 두려움을 느낄 것이다. 그리고 목적지를 찾아가다 길을 잃어버렸을 경우 그 지역의 안내 표지판을 찾아다니거나 지나 가는 사람들에게 물어봐야 하는 번거로움이 있을 것이다. 위의 답답함과 어려움을 해결하기 위해 내비게이션을 사용하지만 특정 공간에서만 사용할 수 있는 단점이 있다. 네비게이션의 단점과 정보에 대한 답답함을 좀더 획기적으로 해결하기 위해서 우리는 스마트폰을 이용한 새로운 서비스를 구축하게 되었다. **BAR는 스마트폰의 GPS와 MAP DATA를 Mapping 하고 차세대 디스플레이 기술인 증강현실을 사용하여 카메라 화면을 통해 사용자가 원하는 정보를 시각적으로 보여준다.** 그리고 부가적인 정보는 실사용자의 정보로 업데이트가 되어 모든 사용자에게 공유할 수 있으며 **wiki** 방식으로 내용을 수정, 편집하는데 있어 **편리한 정보 공유 매체**가 될 수 있다. |

이규태, 최용호, 고석현, 손효일, 이진학, 최근호

**선문 비트 Windows Developer 18기 1조**

**『B. A. R.』**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
| **[ 목 차 ]** | | | | |
| **제1절 프로젝트 개요** | | |  |  |
|  | 1.1 개발배경 및 목적 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 3 |
|  | 1.2 추진체계 및 일정 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 5 |
|  | 1.3 부분별 수행업무 및 담당자 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 6 |
|  | 1.4 프로젝트 개발 범위 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 6 |
| **제2절 배경지식** | | |  |  |
|  | 2.1 기존 증강현실 내비게이션 프로그램 설명 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 7 |
|  | 2.2 A\* 알고리즘 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 7 |
|  | 2.3 윈도우 모바일 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 8 |
|  | 2.4 웹 사이트 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 9 |
| **제3절 시스템 구축내용** | | |  |  |
|  | | 3.1 시스템 구성도 | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 13 |
|  | | 3.2 시스템 흐름도 | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 14 |
|  | | 3.3 Core Techniques | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 16 |
|  | | 3.4 User Interface | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 17 |
| **제4절 개발내용** | | |  |  |
|  | 4.1 기본 클래스 및 메소드 설명 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 21 |
|  | 4.2 영상처리 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 25 |
|  | 4.3 Compass | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 32 |
|  | 4.4 건물 탐색 알고리즘 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 33 |
|  | 4.5 칼리브레이션 데이터 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 35 |
| **제5절 결론 및 개선방안** | | |  |  |
|  | 5.1 결론 및 기대효과 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 37 |
|  | 5.2 테스트 및 검증 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 37 |
|  | 5.3 문제점 및 개선 사항 | | ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ | 38 |

\*첨부 : 참고문헌 및 인터넷 사이트

제1절 프로젝트 개요

내비게이션(Navigation System)은 현재 위치로부터 목적지까지의 거리 및 교통 상황을 안내하는 도로 및 교통정보 제공 시스템이다. 이는 다양한 편리성과 기능으로 인해 수요가 높아져가며 보편화 돼가고 있다. 내비게이션의 형태는 다양하나 일반적으로 GPS수신기에서 위성으로부터 현재 자동차의 위치 정보를 얻어오고 맵 매칭 기술을 통해 2차원지도 화면으로 보여주는 방식을 사용한다.



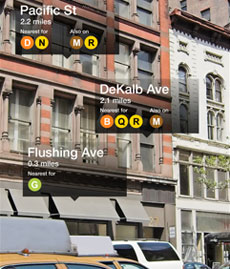
<그림 1.1 아이나비 네비게이션>

이러한 기존의 방식은 전자지도를 화면에 실시간으로 출력함에 있어서 자동차가 실제로 이동하는 방향에 따른 정확한 시야확보가 어려운 경우도 있으며, 불필요한 정보를 보여주는 경우도 있다. 또한 이동속도에 맞춰 이미지데이터가 화면에 적시에 로딩되지 않아 화면과 실제 간의 불일치가 생길 수도 있다. 또한 실제 3차원 환경과 2차원 평면지도간의 상이함으로 인해 많은 운전자들이 내비게이션을 이용하면서 불편함을 겪는 사례가 자주 발생하고 있다

이에 따라 본 프로젝트에서는 카메라에서 획득한 **실사 영상을 지도대신 보여주며 차량의 진행방향과 현재 위치와 가장 가까운 주요건물(POI, Point of Interest)의 정보를 화면에 표시**하여주는 증강현실기법을 적용한 새로운 내비게이션 시스템을 제안한다.

1.1 개발 배경 및 목적

이와 같은 증강현실을 이용한 내비게이션을 개발하게 된 동기는 처음에 일반 사람들이나 장애인들에게 도움이 될 수 있는 프로그램이 무엇이 있을까 고민하다가 청각 장애인이나 길을 찾기 어려운 사람들에게 길안내를 해줄 수 있는 시스템을 생각하게 되었다. 그리고 좀더 나아가 방문 지역에 대해 지리 정보를 알지 못하는 사람에게 스마트폰의 실 카메라를 이용하여 길안내와 실제 영상 정보에 부가적인 안내정보(문화재, 관광명소, 먹거리, 볼거리 등)를 정합하여 사용자에게 보다 나은 현실감을 제공하기 위해 개발하게 되었다.



<표1.1> 해외에서 개발된 증강현실 네비게이션 프로그램의 종류

|  |  |  |  |
| --- | --- | --- | --- |
| **장비** | **옴니아(Windows Mobile)** | **iPhone** | **안드로이드** |
| **개발OS** | Windows Only | Windows, MacOSX,  Linux | MacOSX Only  (include Hackintosh) |
| **개발Tool** | Visual Studio 2005, 2008 | Eclipse 3.3, 3.4 | Xcode 3.1,  Interface Builder |
| **프로그래밍**  **언어** | Visual C++, C#, VB.NET | Java | Objective-C |
| **프레임워크**  **or 런타임** | MFC, .NET CompactFramework | Android Application  Framework, Dalvik | Cocoa Touch |
| **바이너리** | .exe(CLR) | .dex  (Dalvik executable) | .app |
| **패키징** | .cab | .apk | .zip |

<표1.2> 플랫폼들의 특징

안드로이드는 오픈 소스 개발도구인 이클립스를 개발도구로 Windows, Mac, Linux 어디에서나 개발이 가능하다는 점이 장점이다. 개발 도구인 이클립스도 누구나가 무료로 사용할 수 있는 오픈 소스 프로그램이면서 Visual Studio 만큼이나 편리한 개발 환경을 가지고 있다. 또한 안드로이드 자체가 오픈 소스 이므로, 그만큼 유연한 개발이 가능하다.

 iPhone 어플리케이션 개발은 위의 두 플랫폼 보다 상당히 Close된 환경이다. iPhone의 앱스토어는 지금 많이 이슈가 되고 있고 활성화 되어있지만 개발을 위해 Mac이 필요하다는 제한점이 있다.

각 장비들의 플랫폼을 비교해본 결과 현재 비트 교육과정에서 다뤄졌던 Windows 프로그래밍을 할 수 있고 C#을 사용할 수 있는 Windows Mobile을 통해 개발하는 것이 최선이라고 생각을 하였고 장비로는 Windows Mobile 6.1 SDK 기반의 옴니아를 선택하게 되었다.

우리가 찾아본 바로 현재까지는 증강현실 네비게이션이 Window Mobile 환경에서는 개발되지 않은 상태였기 때문에 프로젝트를 계획하게 되었다.

1.2 추진체계 및 일정

팀장 : 이규태

팀원 : 고석현, 손효일, 이진학, 최근호, 최용호

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 11월 | | | | | 12월 | | | | | 1월 | | | | | 2월 | | |
| 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 |
| 설  계 | 주제선정 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 자료수집 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 관련 기술 스터디 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 학  습 | 알고리즘및  관련 논문분석 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 영상처리 학습 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 프  로  그  램  구  현 | 증강현실 구현 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 영상 처리 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 웹사이트 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 나침반, 속도계 구현 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MapData |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Navigation 구현 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 소켓, A\* 알고리즘 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| UI 및 기타 기능구현 | ㅡ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 테  스  트 | 실외 테스트 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 검증 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 문서 정리 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

<표1.3> 프로젝트 일정표

1.3 부분 별 수행 업무 수행 및 담당자

|  |  |  |  |
| --- | --- | --- | --- |
| **분 야** | **담당 업무** | **사용 툴/ 언어/ 라이브러리** | **담 당 자** |
| 증강현실 구현 | 삼성 카메라 API를 마이그레이션 해서 영상을 화면에 출력 | Visual C++, Camera API | 최용호 |
| 영상처리 | OpenCV, CxImage, DirectShow를 이용한 영상처리 | Visual C#, C++, OpenCV, CxImage, DirectShow | 고석현, 손효일, 최근호 |
| 웹사이트 구현 | 네이버 지도API를 이용하여 건물 정보를 해당 좌표에 표시 | Visual C#, Asp.Net | 최용호 |
| 나침반,  속도계 구현 | GPS좌표를 이용하여 구현 | Visual C#, GPS | 이규태 |
| Map Data, Navigation, A\* 알고리즘 구현 | 학교 내의 맵 데이터 제작,  길찾기 기능 구현,  경로 표시, 안내 기능 구현 | Visual C# | 고석현, 이진학 |
| 칼리브레이션 | 영상에 비춰지는 도로의  실거리를 계산 | 실외 측정 | 손효일 |
| UI 및 기타 기능구현 | UI설계, 디자인,  프로그램의 기타 기능 구현 | Visual C#, Asp.Net | 손효일, 최근호 |

<표1.4> 프로젝트 업무수행 및 담당자

1.4 프로젝트 개발 범위

본 프로젝트는 안드로이드폰이나 아이폰에서는 구현이 된 기능이지만 Windows Mobile에는 구현되지 않은 프로그램이라서 도전해보게 되었다. 구현 과정 중에 이러한 증강현실 기법이 Windows Mobile에서 왜 아직까지 구현되지 않았는지가 이해가 되었다. 지원되는 함수들도 제한적이었고, 속도, 성능 면에서도 어려운 점이 많았었다.

본 프로젝트는 안드로이드 폰의 Wikitude를 모델로 삼고 제작하였고 최대한 근접하려고 노력하였다. 사용한 장비인 옴니아는 지자기센서가 탑재되어있지 않아서 제자리에 서서 방향정보를 얻기는 불가능 하였고 GPS좌표를 이용하여 나침반 기능을 개발해서 사용하였다.

Navigation 기능을 위해 A\* 알고리즘을 사용해서 경로탐색을 가능하게 만들었고, Navigation은 팀에서 직접 제작한 맵데이터(선문대학교 내)를 이용해서 구현되었다.

제2절 배경지식

2.1. 기존 증강현실 네비게이션 프로그램 설명

2.1.1. 증강현실 프로그램의 흐름

영어로 ‘AR(Agumented Reality)’는 ‘증강현실’ 또는 ‘확장현실’로 번역한다. 증강현실은 현재 사용자가 보고 있는 사물이나 장소를 사용하여 기계 안에 만든 공간이다. 가상 현실이 100% 가짜로 만든 공간인 것과 달리 증강현실은 눈에 보이는 실제 공간에 각종 정보와 가상공간이 추가된 공간이라는 차이가 있다.

최근 카메라모듈과 GPS·나침반 기능 등이 탑재된 스마트폰이 확산되면서 내장 카메라에 비친 현실정보에 다양한 부가 정보가 결합되는 형태로 개발돼 모바일 서비스의 차세대 키워드로 급 부상 중이다

증강현실 서비스는 지식의 입출력에 대한 새로운 방향을 제시한다. 과거처럼 책을 사서 외워야 하는 시대에서 다시 한번 변화하는 것이다. 과거에는 여행 책자를 들고 여행을 다녀야 했지만 이제는 휴대폰 하나만 들고 다니면 된다. 여행을 하다가 본 멋진 유적에 대한 설명이 궁금하다면 휴대폰 카메라를 건물 쪽으로 향하는 것으로 충분하다. 휴대폰 화면에 표시되는 건물 영상 위에 건물에 대한 정보와 건물 안의 유물에 대한 정보가 글과 그림으로 표시된다. Mobilizy사에서 만든 ‘Wikitude AR’ 프로그램이 바로 이런 일을 하는 프로그램이다. 휴대폰을 대면 유적이나 유명 건물에 대한 설명이 표시되므로 관광가이드가 필요 없다. 길을 잃어도 휴대폰만 켜면 근처 지하철역까지 가능 방향이 표시되고, 처음 가본 나라의 밤하늘 별자리가 궁금할 때도 별자리로 휴대폰을 향하기만 하면 별자리 소개가 나타난다. 먼 미래의 이야기가 아니라 이미 해외에서 사용되고 있는 증강현실 프로그램 이야기이다.

2.1.2. WikiTude AR

이와 같은 AR은 60년대 후반부터 본격적인 연구가 시작돼 지난 92년 처음으로 사용되기 시작한 용어로 ‘WikiTude AR’이 상용 애플리케이션의 효시격으로 꼽힌다.

안드로이드폰에서 위키튜드를 실행한 뒤 휴대전화 카메라로 주변 건물이나 산을 비추면 이름이나 유래 등의 정보가 팝업창처럼 떠오른다. 이렇게 보이는 정보는 위키피디아와 연계된 것으로 굳이 가이드북을 가지고 다니지 않아도 비교적 정확한 정보를 손쉽게 얻을 수 있다.

최근에는 위키튜드 드라이브라는 내비게이션 어플리케이션도 출시됐다. GPS가 장착된 휴대전화 카메라를 구동시키면 동영상 화면 위로 방향을 지시하고, 음성안내까지 제공된다. 어떻게든 실제와 비슷하게 보여주려던 기존 2D, 3D 그림 기반의 내비게이션은 카메라 속에서 구현되는 `확장된 현실' 앞에서 시시하게 느껴질 뿐이다.

2.2. A\* 알고리즘

2.2.1. A\* 알고리즘 개요

길찾기 알고리즘 중 대표적인 알고리즘은 다익스트라(Dijkstra) 알고리즘과 A\* 알고리즘이 있다. 두 알고리즘은 차이점은 다익스트라(Dijkstra) 알고리즘은 지금까지 탐색한 비용만 고려하고, A\* 알고리즘은 앞으로 탐색할 예상비용(휴리스틱값)까지 고려하여 좀더 빠른 길찾기가 가능하다.

A\*는 상태 공간 안의 특정 상태에 이웃한, 즉 인접한 상태들을 조사해 나가면서 출발지로부터 목적지까지 이르는 가장 싼 비용의 경로를 찾는 알고리즘이다. 현재 네비게이션에서도 많이 사용된다.

2.3 윈도우 모바일

2.3.1 윈도우 모바일 어플리케이션 개발

- .NET 컴팩트 프레임워크의 특징

C# 과 VB.NET 언어는 사실 언어만으로써 장점을 가지지 않는다. 바로 .NET CLR 이라는 메인 런타임을 가진 .NET 프레임워크가 있어야 함께 동작한다. 다시 말해 .NET CLR이 C# 과 VB.NET 언어로 생성한 응용 프로그램을 실행시켜 준다.

한편 윈도우 모바일 어플리케이션도 이와 같이 .NET CLR에서 동작되는 데, 이러한 것을 담당하는 것이 바로 .NET 컴팩트 프레임워크 이다. .NET 컴팩트 프레임워크(이하 .NET CF로 표기)는 데스크탑/서버의 .NET 프레임워크의 모바일 버전으로써 윈도우 모바일에 필요한 기능만 탑재되어 있다.

그러나 닷넷CF의 가장 큰 장점은 기존의 비관리형 코드에서 CPU에 종속되어 컴파일 하던 것이 ‘추상 플랫폼 레이어(APL)’을 통하여 .NET CF만 설치되어 있으면 CPU에 관계 없이 모바일 응용 프로그램을 개발 및 배포를 할 수 있다. 따라서 이러한 것은 플랫폼 탈피를 해서 포켓PC용으로 만든 코드를 간단히 UI만 변경하여 스마트 폰 이나 커넥트 카 시스템과 같은 다른 윈도우 모바일 응용 프로그램으로 발전시킬 수 있는 큰 장점을 가진다.

결론적으로 이제 윈도우 모바일 응용 프로그램은 하나의 플랫폼만 설정하는 것이 아니라 .NET CF가 지원하는 모든 플랫폼을 지원하기 위해 화면 UI 와 비즈니스 데이터 처리 등을 별도로 분리해서 개발하는 것이 당연시 된다. 결국은 개체 지향 기법(OOP)으로써 컴포넌트의 재활용성을 십분 발휘하게 되는 모습이다.

- .NET 컴팩트 프레임워크의 UI

지난 2004년 1월에 필자는 마이크로소프트 .NET Advisor 컬럼에서 ‘.NET CF를 기업에서 사용해야 하는 10가지 이유’에 대해 역설했다. IT 개발 현장에서 속칭 ‘삽질’이라 불리우는 처음부터 끝까지 응용 프레임워크 도움 없이 비즈니스 로직을 파악하여 작성하는 것과 응용 프레임워크에서 솔루션을 가지고 개발하는 것은 차이가 크다고 설명했다.

이것은 2002년도 하반기에 필자가 L사의 영업/물류/생산 PDA 시스템을 개발하면서 체험했던 결과이다. 그 당시 .NET CF 바탕으로 시작된 프로젝트는 국내 최초로써 다른 회사들은 아직 EVC++의 MFC로 UI를 개발하며 CDB이나 ISAM 파일을 모바일 데이터 베이스로 사용하고 있을 시점에서, 필자와 필자 회사에서는 과감히 VS 2003의 .NET CF와 SQLCE 를 도입해서 윈도우 모바일 응용 프로그램을 개발했던 것이 오늘날 모바일 개발 업체 들에서 간에서 기술적 경쟁 우위를 점유하지 않았나 싶다.

그림에서는 .NET CF에서 대표적으로 사용할 수 있는 사용자 인터페이스(UI) 프로그램이다. 실행 파일과 소스는 부록으로 제공하겠으니 VS 2003을 가지고 있는 독자라면 실행 해보기 바란다. 또한 .NET CF에서 지원하는 사용자 인터페이스 컨트롤은 기본적으로 윈도우 공통 컨트롤이 지원하는 것과 비슷하다.

그러나 윈도우 모바일에서는 데스크탑의 메모리와 기능을 모두 사용할 수 없으므로 몇 가지 메서드들이 누락된 것이 있다. 이러한 것들은 반드시 MSDN 문서에서 찾아서 확인 해보기 바란다. 처음 .NET CF 으로 프로그래밍 하는 독자들이 가끔 그 기능성이 너무 제약이 많아 쓰기가 불편하다고 호소하는 사람들이 있다.

.NET CF 가 버전 1.0 밖에 안되니 기본적인 기능 외 에는 모두가 WinCE API의 힘을 빌려올 수 밖에 없다. 혹은 .NET CF 가 윈도우 모바일 응용 프로그램을 개발하는 데 있어서 만병 통치 약(?)쯤 생각하는 사람들도 있다.

.NET CF의 주 목적은 공통적인 UI로 인한 빠른 생산성과 더불어 CPU나 플랫폼에 종속하지 않고 배포하여 사용할 수 있는 장점을 가진다고 했다. 따라서 바코드 스캐너와 프린터, MSR과 같은 장치에 의존되는 SDK들은 WinCE API에 기반할 수 밖에 없다.

이런 한계점을 관리형 코드로써 프레임워크로 미국 Embedded MVP들 주 측으로 OpenNETCF 프로젝트를 2003년에 시작하기 되었다. .NET CF 가 부족한 Open 소스 코드들을 확장적으로 잘 연결시켜 놓았다. OpenNETCF 사이트는 http://www.opennetcf.org 로 접속해서 Smart Device Framework 1.2 버전을 다운로드 받기 바란다.

OpenNETCF 프로젝트의 영향을 받아 대거 소스 코드들이 관리화된 형태로 된 것을 볼 수 있다. 이것은 .NET CF 2.0이 들어있는 VS2005(Windows Mobile 5.0 SDK 포함)에서도 마찬가지이다. 예를 들어 VS2003 에서는 하나의 프로그램을 실행시키는 것은 WinCE API의 CreateProcess 함수를 DllImport 시키는 과정을 거쳐야 하지만 VS2005은 EnableApplicationLauncher 함수 한 번만 호출하면 쉽게 해결 할 수 있으니 얼마나 개발자의 생산성을 도와주는가를 알 수 있다.

참고로 VS2005 에는 더 추가된 내역은 UI적인 컨트롤이야 말로 WinCE API로 제공했던 여러 가지 컨트롤을 모두 다 지원해주고 있다. 그 동안 오픈 소스 또는 써드 파티 컨트롤로 지원하던 LinkLabel, DateTimePicker, MonthCaledar, WebBrowser 와 Notification 컨트롤 등을 VS2005 에서는 기본적으로 제공해 준다. 특히 HTMLView 컨트롤을 관리형 코드로 지원함으로써 PDA에서 웹과 같은 인터페이스나 컬러를 원하는 분들에게는 더욱더 유연한 코드를 생성하지 않을까 한다.

그리고 내부 프로토콜적으로 웹 서비스가 WSE 2.0을 지원함으로써 명실 상부하게 웹 서비스의 모든 기능을 지원한다. 또한 XML 직렬화(Serialization) 기능으로써 그 동안 XML 문서를 읽고 DOM 방식으로 운영하는 데 있어서 불편하고 속도가 느린 점을 한층 더 개선했다. 또한 MQ와 같은 비동기화적인 프로토콜을 지원하므로 메시지 상호 방식에 좀더 성능을 향상 시키는 기능이 더 추가 되었다.

2.3.2 모바일 플랫폼 간의 물리적 차이점

 모바일 플랫폼은 크게 세 개의 범주로 나눌 수 있다. 윈도우 모바일이 실행되는 포켓PC, 스마트폰에 특화된 버전이긴 하나 역시 윈도우 모바일이 실행되는 스마트폰, 윈도우 CE가 실행되는 임베디드 하드웨어나 맞춤형 하드웨어이다.

|  |  |  |  |
| --- | --- | --- | --- |
| 플랫폼 | 포켓 PC/윈도우 모바일6 클래식과  프로페셔널 | 스마트폰/윈도우  모바일6 스탠다드 | 윈도우 CE |
| 터치  스크린 | 있음 | 없음 | 디바이스 제조사(OEM)에서 결정 |
| 키패드 | 없거나 QWERTY 키보드 | 휴대폰 숫자  키패드나  QWERTY 키보드 | 제조사에서 결정 |
| 전화기능 | 포켓 PC폰 에디션이나 윈도우 모바일 6 프로페셔널일 경우에만 있으며 다른 경우에는 없음 | 있음 | 기기 별로 다름 |
| 운영체제 | 포켓 PC용 윈도우 모바일  2003 프리미엄 에디션  / 프로페셔널 에디션  / 폰 에디션  포켓 PC용 윈도우 모바일 5  윈도우 모바일 6  클래식 / 프로페셔널 | 스마트폰용 윈도우  모바일 2003  스마트폰용 윈도우  모바일 5  윈도우 모바일 6  스탠다드 | 윈도우 CE 4.2  윈도우 CE 5.0  윈도우 임베디드 CE 6.0 |

<표2.1> 모바일 플랫폼 간의 중요한 물리적 차이점

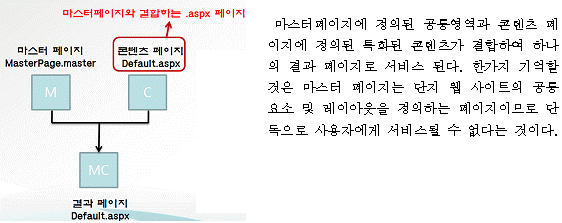
2.4 웹사이트

2.4.1 Master Page

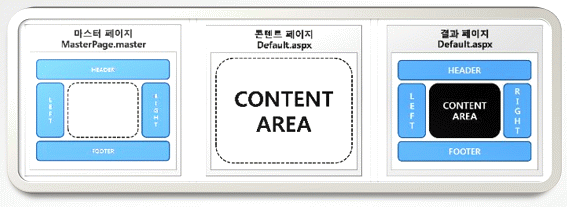
 마스터 페이지는 .master라는 확장자를 가지면서 전체 웹사이트의 공통적인 구조를 정의하는 새로운 개념의 페이지이다. 일반적으로 웹사이트는 상단영역, 좌측영역, 탐색경로 영역, 하단영역과 같이 웹 사이트를 이루는 공통 영역들의 내용은 변동이 없고 각 페이지의 내용들만 바뀌게 된다.



<그림2.1 마스터페이지 영역>

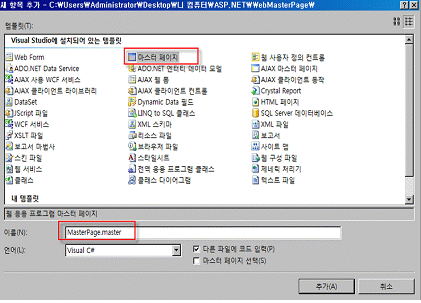


<그림2.2 마스터페이지 구동원리>



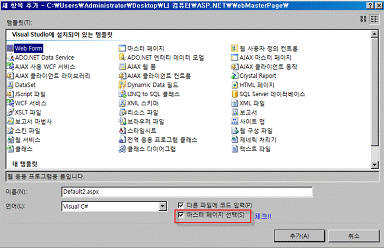
<그림2.3 MasterPage의 Content Area>

 웹 사이트의 공통 영역과 전체적인 레이아웃이 마스터 페이지에 정의되어 있고, 이 마스터 페이지와 특정 콘텐츠를 가지고 있는 콘텐츠 페이지가 결합하여 하나의 결과 페이지로 서비스 됨



<그림2.4 마스터페이지 파일 만들기 - MasterPage.master>

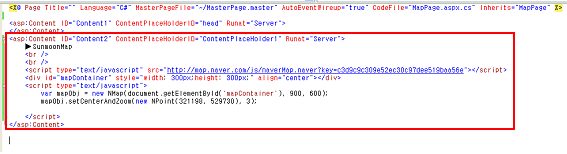
 <그림2.4>와 같이 프로젝트에서 새 항목 추가 >마스터 페이지를 생성하면 확장자 .master로 파일이 생성된다.



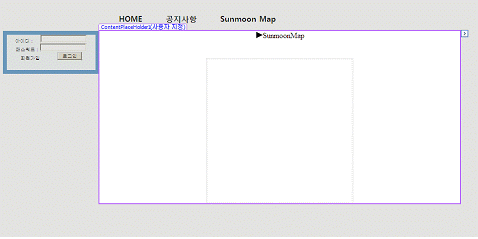
<그림2.5 MasterPage.master 파일>

마스터 페이지를 만들고 난 후 웹 폼 생성 시에 마스터 페이지를 선택해 주면

<ContentPlaceHolder></ContentPlaceHolder> 컨트롤 안의 부분만 수정할 수 있다.



<그림2.6> Default.aspx 소스



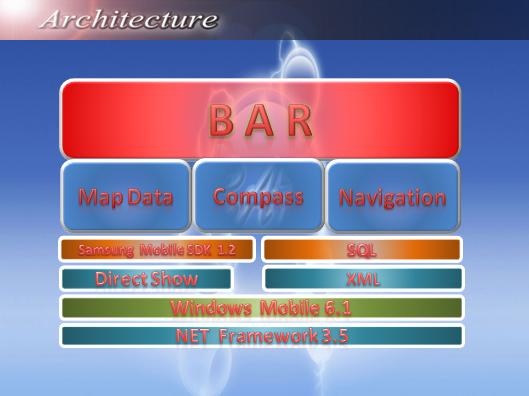
<그림2.7> 페이지 출력 화면

<그림2.7>에서 ContentPlaceHolder1 부분을 제외한 나머지 부분은 마스터페이지에서 구현된 공통 영역이다.

제3절 시스템 구축 내용

3.1. 시스템 구성도

3.1.1. 전체 시스템 구조



<그림3.1 시스템 구성도>

닷넷 컴팩트 프레임워크 3.5기반에 윈도우 모바일 6.1 SDK를 사용하였다.  또한 다이렉트 쇼를 사용하는 카메라 API를 이용하여 영상을 띄웠고,  XML을 사용해서 클라이언트와 서버간 데이터 통신을 하였다.  본 프로젝트에서는 GPS좌표를 이용하여 맵데이터를 구성하였고 나침반 기능을 구현하여 방향 정보를 나타냈다.

3.2. 시스템 흐름도

<그림3.2 시스템 흐름도>

**GPS 수신**

**DB 탐색**

**방향 설정**

**거리 계산**

**건물 표시**

**Navigation**

Insert Container

경로탐색

화살표 표시

[1] GPS를 수신 받는다.

[2] DB를 탐색하여 주변 건물의 정보를 얻어온다.

[3] 방향 설정과 거리계산을 한다.

[4] DB에서 탐색한 정보를 화면에 표시해준다.

[5] 출발지부터 목적지까지의 경로 탐색 기능이 수행 중이라 길안내 기능을 제공한다.

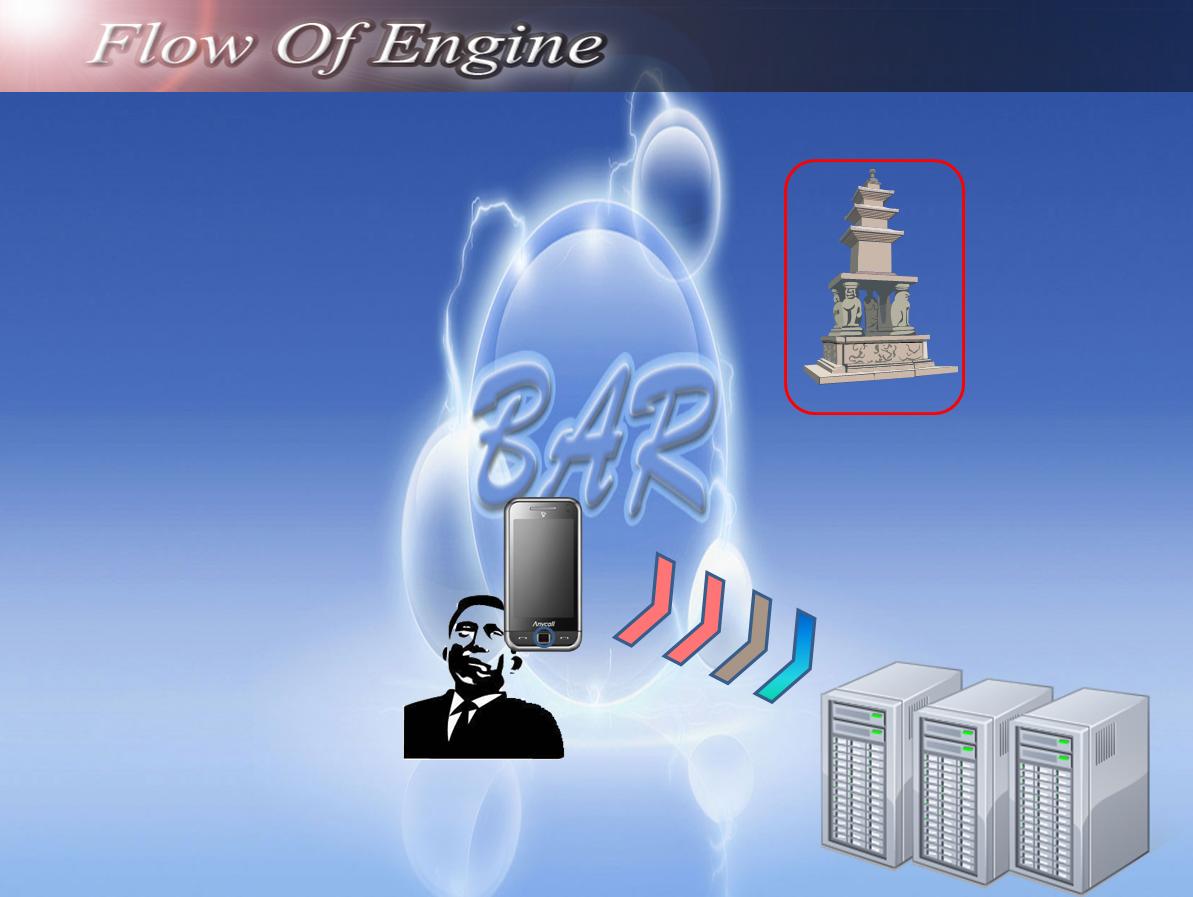
3.2.1. GPS 수신



<그림3.3 GPS 수신>

주변 건물의 정보를 얻으려면 먼저 GPS좌표를 수신 받아 현재 자신의 위치를 파악해야 한다. GPS 좌표를 수신 받으면 위도, 경도 값을 이용하여 데이터베이스를 탐색과 동시에 현재 진행중인 방향이 어떤 방향인지를 연산하게 된다. 방향에 따라 현재 정면에 보이는 건물들에 대한 정보만을 화면이 표시해주게 된다.

3.2.2. DB 탐색



<그림3.4 DB 탐색>

현재 GPS 좌표와 데이터베이스에 기록되어있는 건물들의 좌표를 비교해서 현재 진행 방향에 있는 주변 건물들을 탐색하게 되고 탐색된 건물정보들은 ArrayList에 담겨있다가 실제 카메라 영상에 표시해준다. 탐색은 2초마다 주기적으로 이루어지며 화면에 갱신된다.

3.2.3. 건물정보 출력



<그림3.5 건물 정보>

화면에 표시되어 있는 건물 정보를 클릭하게 되면 건물 정보를 출력해주는 새로운 폼이 열리게 되고 건물에 대한 정보를 인자로 전달 받아 해당 폼에 건물에 대한 상세 정보가 출력된다.

3.3. Core Techniques

3.3.1. A\* 알고리즘



<그림3.6 A\*알고리즘>

F = G(출발지부터 현재까지 비용) + H(현재부터 도착지까지 드는 예상비용)

열린 목록(OpenList) : 아직 탐색하지 못한 노드들을 보관하는 목록

닫힌 목록(CloseList) : 이미 탐색한 노드들을 보관하는 목록

출발지부터 8방향 인접노드들을 확인 후 노드를 열린 목록에 넣는다. 열린목록 중 가장 비용(F)이 적은 노드를 선택한다. 선택된 노드는 닫힌 목록에 넣는다. 선택된 노드의 인접 노드들을 확인후 열린 목록에 넣고, 반복적인 연산을 통해 도착지에 도착하게 된다.

A\* 알고리즘으로 최단경로를 찾으면 네비게이션 길 안내, 경로를 벗어났을 시 재 탐색 기능, 현재위치부터 도착지까지 남은 거리가 보이도록 하였다.

3.3.2 증강현실 기법



<그림3.7 증강현실>

화면에 표시되는 건물 정보는 UserControl로 만들어져 있다. UserControl안에는 건물명, 건물정보가 담겨 있고 클릭하면 상세정보를 보여주는 창을 띄워주도록 이벤트가 정의되어 있다. 건물 정보를 화면에 표시해 주기 위해서 먼저 데이터베이스에서 주변 건물정보들을 탐색하여 ArrayList안에 담게 된다. 그 후에 현재 진행 중인 방향이 동, 서, 남, 북 중에 어떤 방향인지를 계산하여 실제 카메라 영상에 보여지는 건물들에 대한 정보들만 띄워지도록 하였다. 또한 여러 건물들이 영상에 보여 질 경우 현재 위치에서 건물들까지가 어느 정도 인지를 계산하여 거리에 따라 컨트롤이 보여질 수 있도록 구현하였다. 정확한 위치를 파악하기 위해서 영상처리를 하여 도로의 직선을 검출해낸 뒤에 거리 계산을 하고 건물들도 인식할 수 있도록 해서 정확한 건물 위치에 정보를 표시해주려고 했지만 옴니아의 성능 문제와 지원되는 라이브러리가 제한 되어서 구현에 어려움이 있었다.

3.4. User Interface

3.4.1 BAR 프로젝트 실행 화면



<그림 3.8 초기화면>

BAR 프로젝트의 초기화면이 나오고 Click 버튼을 클릭하게 되면 프로젝트가 실행 된다.

카메라 화면이 보여지며 왼쪽 사이드에 버튼 MENU에는 경로탐색, 업데이트/업로드, 사용자의 건물 정보 입력, GPS ON/OFF 버튼이 표시가 된다.

3.4.2 BAR 프로젝트의 실행 화면



<그림 3.9 실행화면>

GPS를 실행(ON)으로서 내 위치에 대한 정보를 통해서 주변에 대한 정보를 받아올 수 있는 준비가 된다. 경로탐색 버튼은 사용자가 목표로 하는 지점까지의 최단거리를 나타내는 지도로 볼 수 있으며 목표 지점까지의 각 특정 위치에 대한 거리를 알 수 있다. 업데이트 버튼은 사용자가 특정 건물에 대한 정보를 수정 / 추가 한 내용에 대한 정보를 업데이트 / 업로드 할 수 있다. 사용자의 건물 정보 입력 버튼은 사용자가 특정 건물에 대한 정보를 수정 / 추가 하가 할 수 있도록 하는 버튼이다. GPS ON / OFF 버튼은 사용자가 GPS 사용 유 / 무를 바꿀 수 있는 버튼이다.

3.4.3 경로탐색 화면



<그림 3.10 경로탐색 지도 화면>

경로탐색 화면이 실행되면 지도가 나오며 지도에는 확대/축소를 할 수 있는 버튼이 있고 오른쪽 상단의 ◀버튼을 클릭하게 되면 사용자가의 원하는 목표지점을 검색하는 창을 볼 수 있습니다.

3.4.4 경로탐색 입력



<그림 3.11 경로탐색 입력화면>

현재 위치와 목표지점을 입력하게 되면 최단거리에 대한 정보가 표시되며 최단거리에 대한 각 특정 위치의 거리를 알 수 있는 정보를 볼 수 있다.

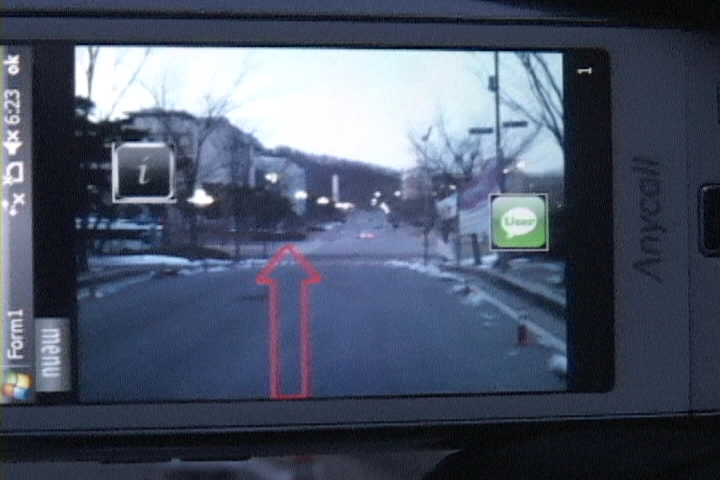
3.4.5 경로탐색 완료



<그림 3.12 경로탐색 완료>

목표 지점까지에 대한 최단 경로탐색이 완료된 지도를 볼 수 있다.

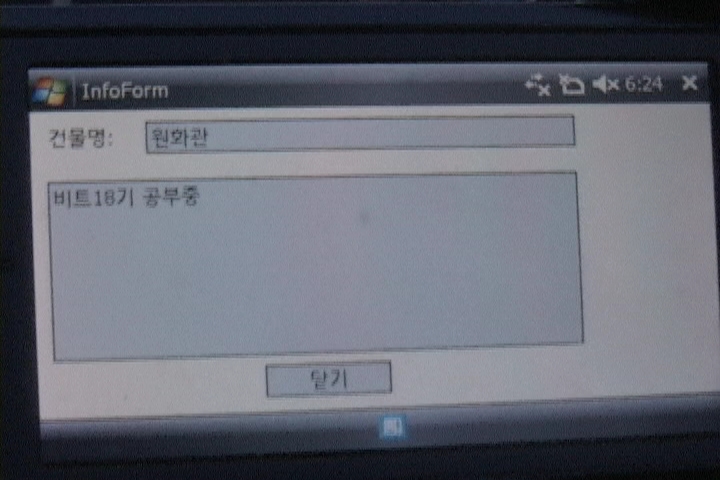
3.4.6 네비게이션 화면



<그림 3.13 네비게이션 화면>

사용자가 입력한 목표지점까지의 최단거리를 안내해주는 네비게이션 화면을 볼 수 있다. 빨간색 화살표는 목표지점까지의 방향을 가리켜 준다. Information 버튼은 건물에 대한 정보를 볼 수 있는 버튼이다. User 버튼은 건물에 대한 사용자가 올린 정보를 볼 수 있는 버튼이다.

3.4.7 Information 화면



<그림 3.14 Information 정보 클릭 화면>

화면에 보이는 건물의 Information 버튼을 클릭 했을 때의 화면으로 건물명과 사용자가 올린 정보 / 건물에 대한 정보를 볼 수 있다.

제4절 개발내용

4.1 기본 클래스 및 메소드 설명

4.1.1 Class 소개

4.1.1.1 MapData Class

출발지와 도착지가 선택되면 자동적으로 탐색(Find함수 호출)된다. 최단경로가 리턴된다.

|  |
| --- |
| //최단경로 검색(FIND)  private void Find()  {  Camera.Set\_Direction(1);  BackTrace.Clear();  BackTrace = mapdll.Find(Spoint, Epoint, 0);  OnDraw();  SetInformation();  }  //진행 방향의 반대방향의 탐색을 막기위한 FIND  private void Find(Point np)  {  BackTrace.Clear();  //Check\_Point\_num은 좌는 1, 상은 2, 우는 3, 하는 4 로 입력됨.  BackTrace = mapdll.Find(Spoint, Epoint, Check\_Point\_num);  OnDraw();  SetInformation();  } |

4.1.1.2 AR\_Navi\_Dll Class

Find 매개변수로 num의 값은 이동 방항성을 나타낸다. Num의 값에 따라서 탐색을 실시한다. 현재위치에서 도착위치까지 A\*알고리즘을 사용하여 최단경로를 탐색한다.

|  |
| --- |
| //최단경로 검색 -- Find  public ArrayList Find(Point sp, Point ep, int num)  {  Init();  //이동방향성 반대로 검색하지 않기 위해서...  if (num != 0)  {  switch (num)  {  case 1: Non\_Back\_Road(sp, -1, 0); break;  case 2: Non\_Back\_Road(sp, 0, 1); break;  case 3: Non\_Back\_Road(sp, 1, 0); break;  case 4: Non\_Back\_Road(sp, 0, -1); break;  }  }  ArrayList con = new ArrayList();  SPoint = sp;  EPoint = ep;  Near(0);  Near(1);  myNode snode = new myNode(SPoint, Start);  CloseList.Add(snode);  Point re = Start;  do  {  int cnt2 = 0;  int L = 0, R = 0, T = 0, D = 0;  //왼쪽 확인  if (Road\_Check(re, 0, -1))  {  L++;  cnt2++;  }  //오른쪽확인  if (Road\_Check(re, -1, 0))  {  T++;  cnt2++;  }  //위에 확인  if (Road\_Check(re, 0, 1))  {  R++;  cnt2++;  }  //아래확인  if (Road\_Check(re, 1, 0))  {  D++;  cnt2++;  }  //좌상확인  if (L + T == 0 && Road\_Check(re, -1, -1))  {  cnt2++;  }  //우상확인  if (T + R == 0 && Road\_Check(re, -1, 1))  {  cnt2++;  }  //좌하확인  if (L + D == 0 && Road\_Check(re, 1, -1))  {  cnt2++;  }  //우하확인  if (D + R == 0 && Road\_Check(re, 1, 1))  {  cnt2++;  }  int cnt3 = -1;  //선택된 노드를 OpenList에서 제거함.  foreach (myNode mn in OpenList)  {  cnt3++;  if (mn.GetNow == re && OpenList.Count != 1)  {  OpenList.RemoveAt(cnt3);  cnt3 = 1000;  break;  }  }  //OpenList중 최소비용인 값을 찾기 위해서 오름정렬  IComparer myFComparer = new GetFComparer();  OpenList.Sort(myFComparer);  myNode sn = (myNode)OpenList[0];  int cnum = 0;  for (int i = 1; i < OpenList.Count; i++)  {  myNode temp = (myNode)OpenList[i];  if (sn.GetF == temp.GetF)  {  cnum++;  }  else  {  break;  }  }  //OpenList중 최소비용인 값을 찾기 위해서 오름정렬  if (cnum != 0)  {  IComparer myHComparer = new GetHCompare();  OpenList.Sort(0, cnum + 1, myHComparer);  }  //도착지 확인  if (Goal\_Check((myNode)OpenList[0]))  {  break;  }  else  {  myNode node1 = (myNode)OpenList[0];  re = node1.GetNow;  //선택된 노드 CloseList에 추가  CloseList.Add(node1);  if (cnt2 != 0 && cnt3 != 1000 && count != 0)  {  OpenList.RemoveAt(0);  }  }  count++;  } while (OpenList.Count != 0);  //최단경로 con1에 대입  ArrayList con1 = BackTrace();  return con1;  } |

4.1.1.3 CMDCtrl Class

삼성에서 제공되는 카메라 API를 사용할 수 있도록 Library의 함수들을 호출할 수 있는 함수 원형들이 선언되어 있다. 모바일에 영상을 출력해주는 역할하고, 원하는 크기로 해상도 설정을 할 수 있고, 화면 캡쳐도 가능하다.

|  |
| --- |
| GRABBERCALLBACK m\_PreviewCallback;  Preview의 Callback 함수포인터  GRABBERCALLBACK m\_StillCallback;  Still의 Callback 함수포인터  CRITICAL\_SECTION m\_Crit;  크리티컬 섹션  static CMDCtrl \* GetInstance();  Singleton 객체  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\* 카메라컨트롤  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  HRESULT InitCamera(WCHAR\* pszCamDriver, HWND hEventWnd);  카메라 필터 그래프 및 필터들을 만든다.  HRESULT OpenCamera();  카메라 필터를 연결한다.  HRESULT StartCamera();  카메라 프리뷰를 RUNNING 상태로 만든다.  HRESULT PauseCamera();  카메라 프리뷰를 PAUSE 상태로 만든다.  HRESULT StopCamera();  카메라 프리뷰를 STOP상태로 만든다  HRESULT SetResolution(TCHAR\* pszPinName, int iWidth, int iHeight, GUID SubType);  카메라의 해상도를 설정한다.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\* 콜백함수  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Void SetStillCallback(GRABBERCALLBACK Callback);  StillCallback 함수를 설정한다. TriggerShutter를 부르면 콜백함수가 실행된다.  Void SetPreviewCallback(GRABBERCALLBACK Callback);  PreviewCallback 함수를 설정한다. StartCamera가 불리면 주기적으로 콜백함수가 실행된다. |

4.1.1.4 MainForm Class

MainForm Class는 프로그램의 중심이 되는 클래스이다. 이 클래스에서는 프로그램의 메뉴들을 클릭하였을 때 폼을 만들어 주고, 카메라의 실제 영상을 출력해주는 기능을 가진 클래스이다. 타이머를 이용해서 주기적으로 GPS 좌표를 받아오고 주변 건물을 탐색하는 역할을 수행한다.

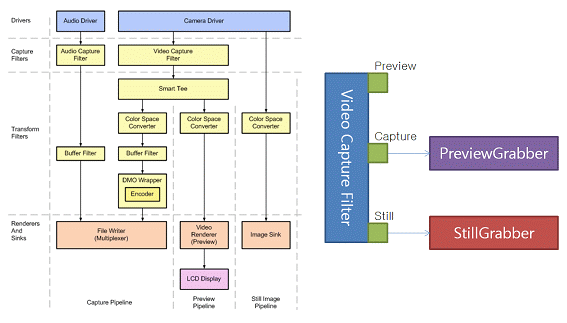
4.1.1.5 GPS\_Compass class

GPS 와 지자기 센서가 없는 옴니아에서 동서남북 방위각을 알아내기 위해 GPS의 좌표를 이용하여 나침반기능을 구현한 Class이다.

|  |
| --- |
| private double Previous\_Lati = 0.0; //전 위치의 위도  //움직인 거리를 구하기 위해 움직이기 이전의 위치를 담기위한 변수  private double Previous\_Long = 0.0; //전 위치의 경도  //움직인 거리를 구하기 위해 움직이기 이전의 위치를 담기위한 변수  private double Change\_lati = 0.0; //변환된 위도  //움직인 만큼의 좌표의 차를 담기위한 변수  private double Change\_long = 0.0; //변환된 경도  //움직인 만큼의 좌표의 차를 담기위한 변수  private double Current\_Lati = 0.0; //현재 위치의 위도  private double Current\_Long = 0.0; //현재 위치의 경도  private double My\_radian = 0.0; //각  private ArrayList myarray;  // 클래스를 통해 제공되는 정보를 return 하는 ArrayList배열  // Index값  // [0] Index MapDataBase 와 비교를 위해 MapDataBase의 좌표와 같은 형식으로 변환된  GPS 위도좌표값  // [1] Index MapDataBase 와 비교를 위해 MapDataBase의 좌표와 같은 형식으로 변환된 GPS 경도좌표값  // [2] Index 360도로 변환한 값  // [3] Index 방향정보  // [4] Index //도,분,초 가 아닌 Double형으로 제공되는 GPS좌표  // [5] Index //도,분,초 가 아닌 Double형으로 제공되는 GPS좌표 |

4.2 영상처리

4.2.1 T-Omnia 카메라 API



<그림 4.1> 카메라 API 구조

카메라를 초기화하기 위해 필요한 함수들로 해상도를 설정하고 콜백 함수들을 지정한 후에 카메라를 구동 시킨다.

|  |
| --- |
| CMDCtrl \* pCtrl = CMDCtrl::GetInstance();  pCtrl->InitCamera(L"CAM1:", hWnd);  pCtrl->SetPreviewCallback(PreviewCallback);  pCtrl->SetStillCallback(StillCallback);  pCtrl->OpenCamera();  pCtrl->SetResolution(L"Capture", REAR\_PREVIEW\_WIDTH  , REAR\_PREVIEW\_HEIGHT, MEDIASUBTYPE\_YV12);  pCtrl->SetResolution(L"Still", REAR\_STILL\_WIDTH, -REAR\_STILL\_HEIGHT,  MEDIASUBTYPE\_IJPG);  pCtrl->StartCamera(); |

HRESULT InitCamera(WCHAR \*pszCamDriver, HWND hEventWnd)

 : 카메라 필터 그래프 및 필터들을 만든다.

HRESULT OpenCamera()

 : 카메라 필터를 연결한다.

HRESULT StartCamera()

 : 카메라 프리뷰를 Running 상태로 만든다.

HRESULT DeInitCamera()

 : 카메라 필터그래프 및 필터들을 해제한다.

HRESULT  SetResolution (TCHAR \*pszPinName, int iWidth, int iHeight,  GUID SubType)

 : 해상도를 설정하는 함수

void  SetStillCallback (GRABBERCALLBACK Callback)

 : StillCallback 함수를 설정한다. TriggerShutter를 부르면 콜백 함 수가 실행된다.

void  SetPreviewCallback (GRABBERCALLBACK Callback)

 : PreviewCallback 함수를 설정한다. StartCamera가 불리면 주기적으로 콜백 함수가 실행된다.

다음 소스는 영상 캡쳐하여 그림 파일로 저장하는 StillCallback 콜백 함수이다.

|  |
| --- |
| HRESULT StillCallback(PBYTE pBuffer, int iBufferSize, int nWidth, int nHeight)  {          RETAILMSG(1,(L"StillCallback(pBuffer:0x%X,  BufferSize:%d, Width:%d, Height:%d)\r\n",  pBuffer, iBufferSize, nWidth, nHeight));            HANDLE hFile = CreateFile(L"\\Rear.jpg", GENERIC\_WRITE,  0, NULL,CREATE\_ALWAYS,  FILE\_ATTRIBUTE\_NORMAL, (HANDLE)NULL);          DWORD dwWritten = 0;          WriteFile(hFile, pBuffer, iBufferSize, &dwWritten, NULL);          CloseHandle(hFile);          return S\_OK;  } |

StillCallback 함수는 TriggerShutter()를 통해 호출 된다.

|  |
| --- |
| CMDCtrl \* pCtrl = CMDCtrl::GetInstance();  pCtrl->SetAMCameraControl(CameraControl\_Zoom, SENSOR\_AUTOFOCUS\_DO);  pCtrl->SetAMCameraControl(CameraControl\_Zoom, SENSOR\_AUTOFOCUS\_RELEASE);  pCtrl->TriggerShutter(); |

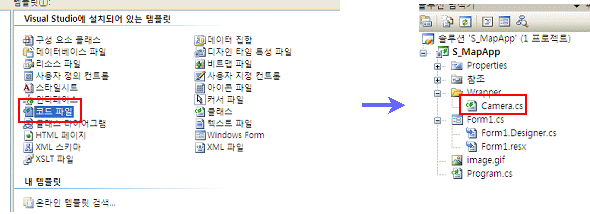
4.2.2 Dll파일 Wrapping하여 사용하기

extern "C" \_\_declspec(dllexport)

HRESULT StartCamera(HWND hWnd)

export한 dll 함수를 사용하기 위해서 코드파일을 생성하여 import부분을 클래스 화 한다.

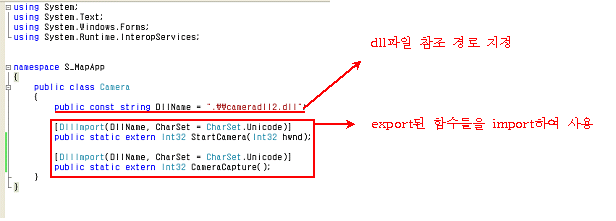
1. 프로젝트에서 추가 → 새 참조에서 코드파일을 선택한다.



2. System.Runtime.InteropServices를 참조 한다.



InteropServices는 플랫폼 호출 서비스를 지원하는 많은 멤버들을 제공한다. 닷넷 컴팩트 프레임워크가 풍부한 프로그래밍 모델을 제공하더라도 가끔 라이브러리에서 제공하지 않는 기능이 필요하고, 윈도우 CE 운영체제의 기능 일부분이나 레거시 네이티브 코드 같은 추가적인 기능이 필요하다. 외부 코드에서 사용할 수 있게 하는 강력한 플랫폼 호출 하부 시스템을 사용할 수 있다.



 Win32 Api로 제작된 dll을 C#에서 사용하기 위해선 네이티브 타입을 매니지드 타입에 적용시켜야 한다. 닷넷 컴팩트 프레임워크 버전 1.0 이상에서는 이러한 수동 변환 작업을 상당부분 대신 해준다.

|  |  |  |
| --- | --- | --- |
| 닷넷 타입 | 크기(바이트) | C++타입 |
| System.Boolean | 2 | short |
| System.Byte | 1 | uchar |
| System.SByte | 1 | char |
| System.Int16 | 2 | short |
| System.UInt16 | 2 | ushort |
| System.Int32 | 4 | LONG |
| System.UInt32 | 4 | DWORD, UINT, ULONG |
| System.Int64 | 8 | LONGLONG |
| System.UInt64 | 8 | ULONGLONG |
| System.IntPtr | 4 | HANDLE, HWND, PBYTE 등 |
| System.UIntPtr | 4 | - |
| System.Single | 4 | Float |
| System.Double | 8 | - |
| System.Guid | 16 | GUID |
| System.String | 4(Pointer) | LPWSTR |
| System.Array | - | PInvoke 메소드에서는 LPVOID,  COM연동 시에서는 SAFEARRAY |
| System.Enum | - | 열거형 타입은 원래의 기본타입을 바탕으로 마샬링된다.  구체적으로 지정하지 않으면 모든 열거형은 Int32타입으로 생성된다. |

<표 4.1> 닷넷 타입과 기본 마샬링 동작

특정 마샬링 동작을 지정하고 싶다면 MarshalAsAttribute를 적용한다.

ex)

|  |
| --- |
| [DllImport("coredll.dll", SetLastError = true)]  [return: MarshalAs(UnmanagedType.Bool)]  internal static extern bool CloseHandle(IntPtr hObject); |

CloseHandle은 이벤트, 뮤텍스, 데이터베이스 테이블 같은 리소스에 연결된 네이티브 시스템 핸들을 종료시킨다. 그 후 런타임은 함수 호출 결과가 네이티브 BOOL타입임을 확인하고 System.Boolean타입으로 자동 변환한다.

4.2.3 영상처리 구현 부분

CxImage 라이브러리에서 라인검출 알고리즘이 지원되지 않아 라인검출 알고리즘을 직접 만들어서 사용하려고 했다. 카메라 영상의 이미지를 CxImage 만들었다. 영상처리의 속도를 높이기 위해서 CxImage 라이브러리 함수 중 Resample 함수를 사용하여 이미지를 1/3로 축소하여 속도를 높이려고 하였다. 이미지를 임의의 경계 값을 기준으로 이진화하였다. 기울기를 기준으로 직선을 검출하려고 하였다.

|  |
| --- |
| DWORD WINAPI MainThread(LPVOID temp)  {  HDC dc = GetDC(g\_hWnd);  HDC memdc = CreateCompatibleDC(dc);  Info \*info = (Info \*)temp;  CxImage \*img = new CxImage(0);  //이미지를CxImage로생성  img->CreateFromArray(info->pBuf, info->width, info->height,  24, info->width/3\*4, true);  //이미지를1/3로축소  img->Resample(info->width,info->height/3, 1, NULL);    int i, j, k;  int Lnumcnt=0;  int Rnumcnt=0;  // 초기화  for(i=0; i<20; i++)  {  Lnum[i][0] = 0;  Lnum[i][1] = 0;  Rnum[i][0] = 0;  Rnum[i][1] = 0;  num[0][i] = 0;  num[1][i] = 0;  m\_num[0][i]=0;  m\_num[1][i]=0;  }  int x1=0, x2=0;  // 임의의경계값으로이진화  for(i = 202; i>=102; i=i-10)  {  for(j=10; j<202 ; j++)  {  RGBQUAD rgb = img->GetPixelColor(j,i,true);  TCHAR str[100];  if(rgb.rgbRed > 150)  {  img->SetPixelColor  (j,i,new RGBQUAD(255,255,255),false);  }  else  {  img->SetPixelColor(j,i,new RGBQUAD(0,0,0),false);  }  // 이미지를반으로나눠서왼쪽,  오른쪽배열에픽셀의좌표값을넣는다.  if(j>106)  {  Rnum[(202-i)/10][0] = i;  Rnum[(202-i)/10 + 1][0] = j;  }  else  {  Lnum[(202-i)/10][1] = i;  Lnum[(202-i)/10 + 1][1] = j;  }  }  }  i=0;  int numcnt =0;  int m\_tcnt = 0;  int m\_cnt = 0;  int maxnum = Rnum[i][0];  // 점들의기울기를확인후num에대입  for(i = 0; i<20 ; i++)  {  if(maxnum > Rnum[i+1][0])  {  maxnum = Rnum[i+1][0];  if(Rnum[i+1][0] != 0 && Rnum[i+1][0] != Rnum[i][0])  {  x1 = ((0 - Rnum[i][0])\*(Rnum[i+1][1]-Rnum[i][1]))  /(Rnum[i+1][0] - Rnum[i][0]) + Rnum[i][1];  x2 = ((160 - Rnum[i][0])\*(Rnum[i+1][1]-Rnum[i][1]  ))/(Rnum[i+1][0] - Rnum[i][0]) + Rnum[i][1];  num[0][numcnt++] = x1;  num[0][numcnt++] = x2;  num[0][numcnt++] = (Rnum[i+1][1]-Rnum[i][1])  /(Rnum[i+1][0] - Rnum[i][0]);  }  for(j=0; j<m\_tcnt; j++)  {  if(num[0][numcnt++] != 0 &&m\_num[0][j] ==  (Rnum[i+1][1]-Rnum[i][1])/(Rnum[i+1][0] - Rnum[i][0]) )  {  m\_num[1][j]++;  m\_cnt++;  m\_tcnt++;  }  }  if(m\_cnt==0)  {  m\_num[0][m\_tcnt++] = (Rnum[i+1][1]-Rnum[i][1]  )/(Rnum[i+1][0] - Rnum[i][0]);  m\_num[0][m\_tcnt++]++;  }  m\_cnt=0;  }  }    numcnt=0;  // 점들의기울기를확인후num에대입  for(i = 0; i<20 ; i++)  {  if(maxnum > Lnum[i+1][0])  {  maxnum = Lnum[i+1][0];  if(Rnum[i+1][0] != 0 && Lnum[i+1][0] != Lnum[i][0])  {  x1 = ((0 - Lnum[i][0])\*(Lnum[i+1][1]-  Lnum[i][1]))/(Lnum[i+1][0] - Lnum[i][0]) + Lnum[i][1];  x2 = ((160 - Lnum[i][0])\*(Lnum[i+1][1]-  Lnum[i][1]))/(Lnum[i+1][0] - Lnum[i][0]) + Lnum[i][1];  num[0][numcnt++] = x1;  num[0][numcnt++] = x2;  num[0][numcnt++] = (Lnum[i+1][1]-  Lnum[i][1])/(Lnum[i+1][0] - Lnum[i][0]);  }  for(j=0; j<m\_tcnt; j++)  {  if(num[0][numcnt++] != 0 &&m\_num[0][j] ==  (Lnum[i+1][1]-Rnum[i][1])/(Lnum[i+1][0] - Lnum[i][0]) )  {  m\_num[1][j]++;  m\_cnt++;  m\_tcnt++;  }  }  if(m\_cnt==0)  {  m\_num[0][m\_tcnt++] = (Lnum[i+1][1]-  Rnum[i][1])/(Lnum[i+1][0] - Lnum[i][0]);  m\_num[0][m\_tcnt++]++;  }  m\_cnt=0;  }  }    DeleteDC(memdc);  ReleaseDC(g\_hWnd, dc);  threadcheck = TRUE;  delete info;  return 0;  } |

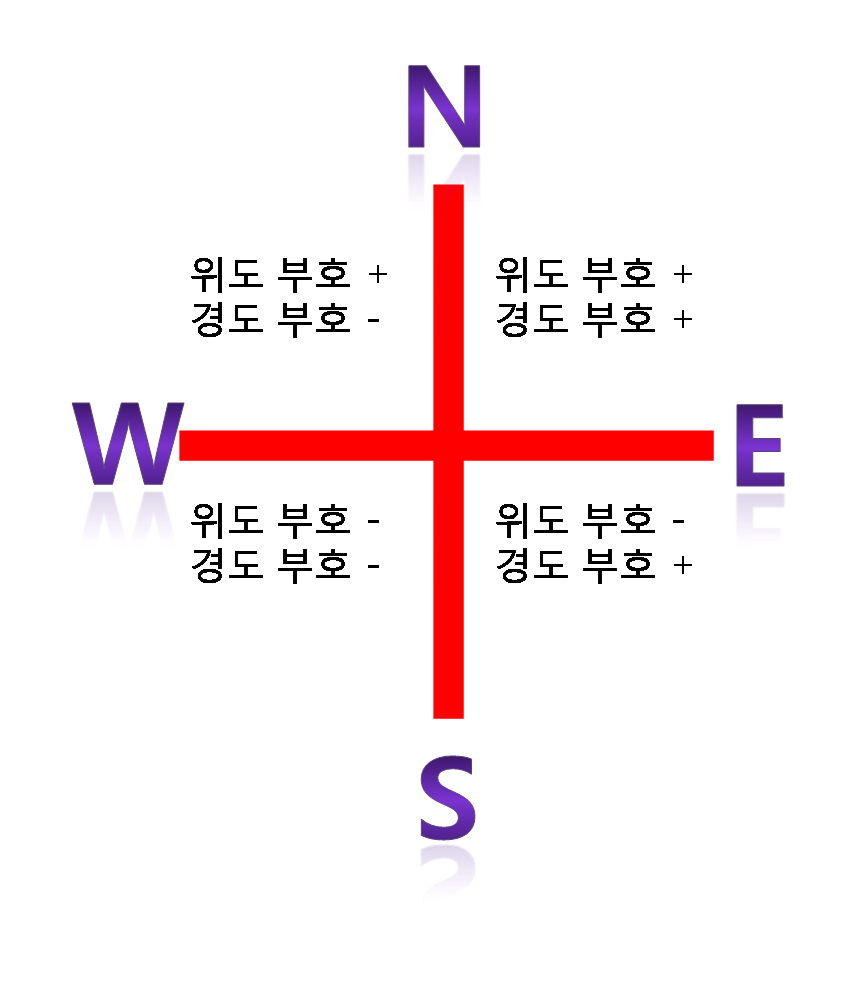
4.3 Compass구현

GPS와 지자기 센서가 없는 옴니아에서 동서남북 방위각을 알아내기 위해 GPS의 좌표를 이용하여 나침반을 구현하였다 .

GPS를 구동시켜 GPS좌표 값을 실시간으로 비교하여 현재의 위치와 그 전 위치의 좌표를 비교하여 현재 자신이 향하고 있는 방향의 방위각을 계산한다.

위도의 값이 상승하면 북을 향하게 되고 위도 값이 감소하게 되면 남, 경도 값이 상승하면 동

경도 값이 감소하게 되면 서를 향하게 된다.



<그림 4.9 나침반>

좀더 자세한 방위각을 알아내기 위하여 위도가 변한만큼의 값을 직각삼각형의 세로, 경도가 변한만큼의 값을 직각삼각형의 가로로 이용하여 의 공식으로 자신이 이동한 위치의 각을 구하게 되고 이 각을 360으로 변환하기 위해 <그림1>에서 각 분면마다 알맞은 값을 더해준다. 나온값의 각에 따라서 자신이 향하고 있는 방향의 정확한 방위각을 구하게 된다.

1/4분면 : +0, 2/4분면, 3/4분면 : +180, 4/4분면: +360

|  |
| --- |
| Change\_lati = Math.Round((Current\_Lati - Previous\_Lati) / 2, 5);  Change\_long = Math.Round((Current\_Long - Previous\_Long) / 2, 5);  My\_radian = Math.Atan(Change\_lati / Change\_long) \* (180 / Math.PI); |

4.4 건물 탐색 알고리즘

화면에 증강현실의 건물을 띄우기 위해서는 자신이 향하고 있는 방향의 건물을 탐색하여야 한다. 건물을 탐색하기 위하여 데이타베이스에 저장되어 있는 건물의 좌표를 현재 자신의 위치와 비교하게 되고, 건물과의 거리가 200M 이하의 건물을 인식하게 된다. 여기에 카메라 화면에 보이는 건물들의 정보를 표시해주기 위해서 현재 진행중인 방향의 일정 범위(바라보는 방향에서의 좌우 60도씩 120도내)를 계산하게 된다.

건물과의 거리를 구하는 방법은 피타고라스의 법칙을 이용하여 현재위치의 좌표와 건물의 좌표를 경도 가로, 위도 세로로 계산하여 빗변의 길이를 구하여 건물과의 거리를 측정하게 된다.

위도 경도의 Second기준으로 1Second당 37M가 된다. 이를 이용하여 빗변의 길이에 37M를 곱해주어 건물과의 거리를 구하게 된다.

|  |
| --- |
| // Lati\_Position 현재위도, Long\_Position  // 현재경도 B\_lati건물위치 위도값,B\_long건물위치경도값  double X = build[num].B\_lati - Lati\_Position;  double Y = build[num].B\_long - Long\_Position;  double length = Math.Sqrt((Math.Pow(X \* 15, 2) + Math.Pow(Y \* 15, 2))); |

건물의 위치에 있는 방향을 구하는 방법으로는 Compass를 구현할 때에 자신이 향하는 방향을 구하는 방법과 같다.

건물과 현재위치의 위도 경도의 차를 구하여 위도의 차는 직각삼각형의 세로 경도의 차는 직각삼각형의 가로로 이용하여 이용하여 의 공식으로 자신이 이동한 위치의 각을 구하게 되고 이 각을 360도에 맞게 변환하기 위해 각 분면마다 알맞은 값을 더해준다.

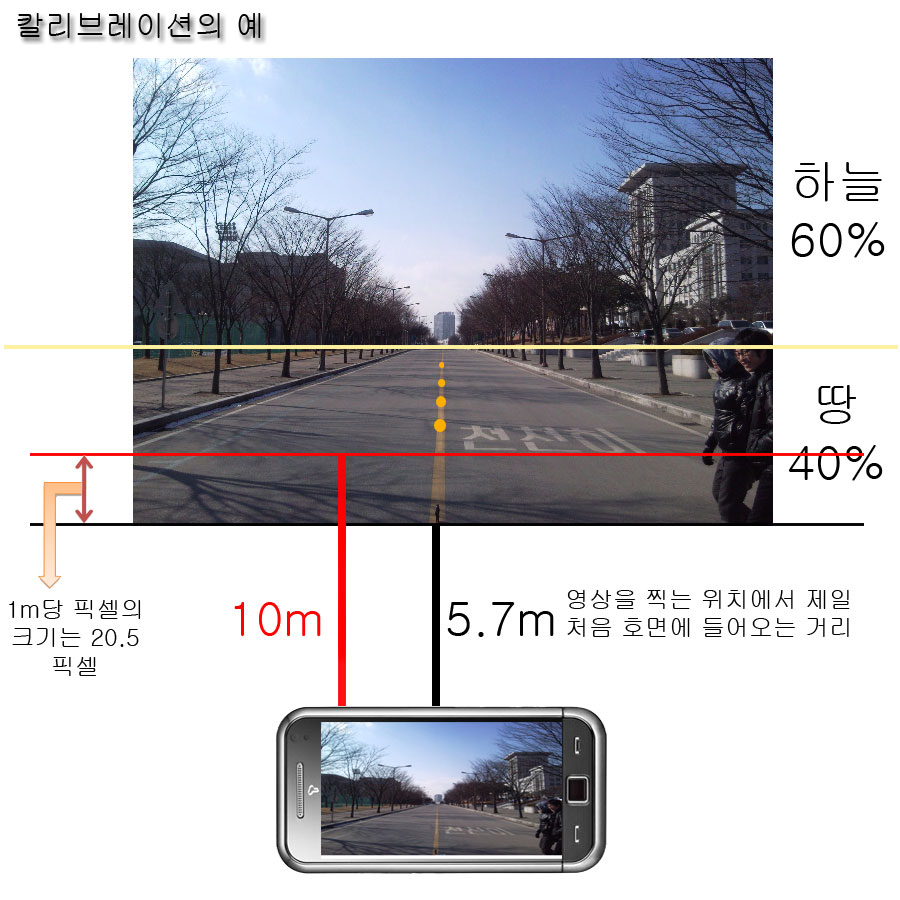
1/4분면 : +0, 2/4분면, 3/4분면 : +180, 4/4분면: +360이 각과 현재 자신이 향하고 있는 방위각을 서로 비교하여 ±60도 내에 위치하고 있는 건물을 찾게 된다..

|  |
| --- |
| // Lati\_Position 현재위도, Long\_Position  // 현재경도 B\_lati건물위치 위도값,B\_long건물위치경도값  double X = build.B\_lati - Lati\_Position;  double Y = build.B\_long - Long\_Position;  double Build\_radian = Math.Atan(X / Y) \* (180 / Math.PI);  // 1/4  if (Build\_radian < 0.0 && (X < 0.0 || Y > 0.0))  {  BuildFullRadi = ABS\_radian; //ABS\_radian 절대값  }  // 2/4  else if (Build\_radian > 0.0 && (X < 0.0 || Y < 0.0))  {  BuildFullRadi = 180 - ABS\_radian;  }  // 3/4  else if (Build\_radian < 0.0 && (X > 0.0 || Y < 0.0))  {  BuildFullRadi = 180 + ABS\_radian;  }  // 4/4  else if (Build\_radian > 0.0 && (X > 0.0 || Y > 0.0))  {  BuildFullRadi = 360 - ABS\_radian;  } |

건물의 거리와 방향 두 조건이 모두 다 맞게 되면 카메라 화면에 건물의 컨트롤을 표시해준다.

|  |
| --- |
| // Fullradian = 현재자신의 진행 방위각  // 방위각에서 좌우 60도반경에 위치하고 있는 건물을 표시  // returnVal,ComValue GPS값이 들어오기전 null일 때 예외처리  // X\_Pos,Y\_Pos = 현재자신의 위치  if (returnVal != null && ComValue != null &&  ((Fullradian + 60 >= B\_Fullradi && Fullradian - 60 <= B\_Fullradi) ||  (X\_Pos == 0 && Y\_Pos == 0))) //X\_Pos,Y\_Pos == 0일 때 예외처리  {  // UserControl 위치  // 카메라 폼의 Location1 == 1M  int left = 205 + ((int)((Math.Cos(Math.PI \* B\_radian / 180.0)) \* B\_length)) \* 2;  int top = (int)Math.Sqrt(Math.Pow(B\_length, 2) - Math.Pow(left, 2));    int x1 = 200, y1 = 30, x2 = 200, y2 = 590;  y1 = (Y\_Pos \* 15) + 348;  x1 = (int)Math.Sqrt(Math.Pow(B\_length, 2) - Math.Pow(Y\_Pos, 2));  y2 = 348 - (Y\_Pos \* 15);  int x = 200, y = 30;  if (Fullradian > B\_Fullradi)  {  x = x1;  y = y1;  }  else if (Fullradian < B\_Fullradi)  {  x = x1;  y = y2;  }  if (build.state == 0)  {  UserControl.BInfo b = new S\_MapApp.UserControl.BInfo(build.BuildName, build.BuildInfo, this);  b.Location = new Point(x, y);  b.Parent = this;  b.Show();  BinfoCon.Add(b);  }  else  {  UserControl.UInfo u = new S\_MapApp.UserControl.UInfo(build.BuildName, build.BuildInfo, this);  u.Location = new Point(x, y);  u.Parent = this;  u.Show();  UinfoCon.Add(u);  }  Find\_buildNum++;  }  cnt++;  }  } |

4.5 칼리브레이션 데이터



<그림 4.10> 칼리브레이션의 예

칼리브레이션은 하드웨어가 최적의 상태로 구동 할 수 있도록 조작하는 방법을 말한다.

여기서는 영상처리가 되었을 때 카메라 화면으로 보이는 영상과 현실세계의 오차를 최대한 줄일 수 있게 데이터를 정리 하였다.

실제 카메라를 통해 보여지는 화면은 480픽셀 안에서 보여진다. 480픽셀 중에서 하늘이 차지하는 부분과 땅이 차지 하는 부분을 10퍼센트씩 비율화 하여서 기준 픽셀을 만든다.

그리고 각 퍼센트마다 카메라와 실제 화상이 처음 보이는 부분까지의 거리를 측정을 한다.

그 후에는 5m 마다 보이는 부분을 체크하고 30m 이후에는 10m씩 체크를 하다가 50m 이후에는 90m 한번 150m까지 측정을 한다.

각 지점과 매칭되는 픽셀 값들을 구하고 각 거리당 1픽셀당 실제거리를 구해서 데이터화 한다.

이렇게 모아진 데이터들의 수치를 가지고 영상 처리 후 카메라 위에 버튼 등을 만들 때 위치를 좀더 세밀하게 지정 할 수 있다.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 하늘 | 288 | 60% | 480픽셀 | | 392픽셀 | | 351픽셀 | | 333픽셀 | 320픽셀 | 313픽셀 |
| 땅 | 192 | 40% | 0픽셀 | | 88픽셀 | | 129픽셀 | | 147픽셀 | 160픽셀 | 167픽셀 |
|  |  |  | 5.7m | | 10m | | 15m | | 20m | 25m | 30m |
|  |  |  |  | | 1m=20.5픽셀 | | 1m=8.2픽셀 | | 1m=3.6픽셀 | 1m=2.6픽셀 | 1m=1.4픽셀 |
| 307픽셀 | | 301픽셀 | | 293픽셀 | | 288픽셀 | |
| 173픽셀 | | 179픽셀 | | 187픽셀 | | 192픽셀 | |
| 40m | | 50m | | 90m | | 150m | |
| 1m=0.6픽셀 | | 1m=0.6픽셀 | | 1m=02픽셀 | | 1m=0.083 | |

<표4.2> 칼리브레이션 데이터의 예

제5절 결론 및 개선방향

5.1 결론 및 기대효과

본 프로젝트는 내비게이션의 화면 인터페이스에 증강현실기법을 적용하여 현존하는 내비게이션들이 갖는 문제점을 보완하였다. 기존 내비게이션들은 2D 맵 또는 3D 맵을 화면에 보여줌으로써 주행 방향과 교통상황 등을 알려주었다. 그러나 이러한 방법들은, 첫째 맵 상에서의 현재 진행 차선 표시의 불명확 , 둘 째 맵 상의 환경에서의 척도 및 지형지물 데이터가 실제 도로 환경에 맞도록 적시에 업데이트 되지 않아서 운전자로 하여금 인식의 차이를 가져오게 하거나, 셋째 고차의 복잡한 교차로에서 좌∙우 방향 안내 멘트 및 표시가 명확하지 않는 등 상이한 점을 갖고 있으며, 맵 이미지 데이터제작 및 수정에 고비용과 어려움이 있다.

본 프로젝트에서는 실사이미지를 내비게이션의 화면으로 보여줌으로써 현재 주행하는 거리와 같은 화면을 바로 볼 수 있어 실세계와 맵의 상이함으로 발생되는 문제가 없으며 맵의 이미지 데이터 등을 따로 제작, 관리하지 않아도 되는 경제적 이점이 있다. 또한 주행방향뿐만 아니라 근처에 중요한 건물의 정보를 실시간으로 제공함으로써 사용자가 운전하는 지역 정보를 쉽게 알 수 있다. 기간 내에 영상처리 기술을 완성하지 못했지만 영상처리가 완벽하게 이루어지고, 지자기센서가 탑재된 장비를 사용한다면 실제 건물의 위치에 건물 정보들을 표시해줌으로써 사용자가 원하는 건물 정보를 정확하게 얻을 수 있게 될 것이다..

5.2 테스트 및 검증

5.2.1 국내외 프로그램 비교

|  |  |  |
| --- | --- | --- |
| **국내외 프로그램 비교** | | |
| **프로그램명(회사명)**  **<국가>** | 증강현실 길찾기 프로그램(B.A.R)  <한국> | Wikitude AR  <미국> | 플레이맵  <한국> |
| **실사 영상 이용** | O | O | X |
| **네비게이션** | O | O | X |
| **건물정보 표시** | O | O | O |
| **Wiki방식** | O | O | O |
| **사용자 편의성 고려기능** | 경로 탐색시 화살표 표시 | 경로 탐색시 화살표 표시 | 2D맵으로 다른 사용자들이 등록한 정보들을 보기쉽게 표시 |
| **사용 용도** | 주변 건물에 대한 정보 검색, 내비게이션 | 주변 건물에 대한 정보 검색, 내비게이션 | 주변 건물에 대한 정보 검색 |
| **속도계, 나침반 구현** | O  (지자기센서에 의한 나침반 기능 구현 안됨) | O  (지자기센서를 이용한 나침반기능 구현) | X |
| **개발시기** | 09년 | 09년 | 09년 |
| **특  징** | 증강현실, Wiki방식 | 증강현실, Wiki방식 | Wiki방식 |

<표5.1> 국내외 프로그램 비교

5.3. 문제점 및 개선사항

본 프로젝트인 BAR를 구현 하면서 발생한 문제점 중에 가장 크게 발생된 문제점은 .NetCF의 사용 기능이 상당부분 제한적이었다는 것이다. 기능적으로 제한이 많이 따랐기 때문에 구현하는 데에 많은 어려움이 있었다. 또한 작은 모바일 기기에 비교적 규모가 큰 프로젝트를 진행하다보니 성능 면에서 많이 떨어지는 문제점이 발생하였다. .NetCF의 제한적인 기능이 개선되어 좀더 다양한 기능 사용이 가능해진다면,

좀더 좋은 프로젝트로 발전 될 수 있을 것으로 생각된다. 또한 모바일 기기에서도 좋은 성능을 유지할 수 있도록 개선해 나아 갈 방침이다.

* 부록 및 참고자료 \*

|  |  |
| --- | --- |
| 자료목록 (인터넷 사이트 주소 및 참고 도서 자료) | |
| 주소 및 도서, 논문 이름 | 비고 |
| 증강현실 기술을 적용한 실감 내비게이션 시스템 개발 | 영상처리 관련 논문 |
| A Study on Architecture of Real Time Image Processing System | 영상처리 관련 논문 |
| The Study of Aerial Triangulation Using GPS | GPS 관련 논문 |
| Analysis of Adaptive Digital Signal Processing for Anti-Jamming GPS System | GPS 관련 논문 |
| 선 형 대 수 학 ( 황인홍 ) | 건물 탐색 관련 |
| C# Digital Image Processing : 디지털 영상처리 ( 정민영 ) | 영상처리 기술 관련 |
| Learning OpenCV : Computer Vision with the Opencv Library ( Gray Bradski ) | 영상처리 기술 관련 |
| Visual C++을 이용한 디지털 영상처리 ( 강동중 ) | 영상처리 기술 관련 |
| GPS 이론과 응용 ( B. Hofmann-Wellenhof ) | GPS 기술 관련 |
| 익스프레션 Web2 ( 김영일 ) | Web 관련 |
| Network Programing Windows ( Anthony Jones ) | Socket 관련 |
| 윈도우 모바일 애플리케이션 개발 가이드 ( Anthony Jones ) | 모바일 기술 관련 |
| CLR via C# 2nd Edition ( Jeffrey Richter ) | C# 관련 |
| about ADO.NET Programming ( 손성필 ) | DB 관련 |
| www.opencv.co.kr | OpenCV 관련 |
| cafe.naver.com/openapi | OpenAPI 관련 |
| www.aistudy.com/heuristic/A\_star.htm | Astar 알고리즘 관련 |