

# 3차원 객체를 2차원 디스플레이화하는 가상현실 인터페이스 구현

소속 동아리	상상 로봇연구반
연구팀명	싸컨대사건
연구팀원	2학년 5반 고은진 2학년 5반 황현조 2학년 14반 윤태웅

# 목차

## 1. 서론

### 1.1. 탐구 배경 / 탐구 목적

## 2. 본론

### 2.1. 이론적 배경

#### 2.1.1. 센서의 원리

##### 2.1.1.1. 자이로센서

##### 2.1.1.1.1. 자이로센서의 기본적인 원리

##### 2.1.1.1.2 HiTECH LEGO MINDSTORMS NXT Gyro Sensor

##### 2.1.1.2 가속도센서

#### 2.1.2. 프로그램의 소개

##### 2.1.2.1. RobotC

##### 2.1.2.2. Arduino main Program

##### 2.1.2.3. 프로세싱

##### 2.1.2.4. 시리얼통신

#### 2.1.3. 하드웨어의 소개

##### 2.1.3.1. 아두이노 우노 보드

##### 2.1.3.2. MINDSTORM NXT(LEGO No. 9797), EV3(LEGO No. 45544)

#### 2.1.4. 기본 알고리즘

##### 2.1.4.1. 수학적으로 바라본 3축 좌표계

#### 2.1.5. 현재의 가상현실

##### 2.1.5.1. 오쿨러스 리프트

##### 2.1.5.2. 마인크래프트

### 2.2. 연구내용

#### 2.2.1. 연구의 방향성

#### 2.2.2. 1세대 로봇

##### 2.2.2.1. 1축 자이로를 사용한 2차원 객체의 2차원 GUI 디스플레이화

##### 2.2.2.2.1. 직교

##### 2.2.2.2.2. 삼각형

##### 2.2.2.2.3. 사다리꼴

#### 2.2.3. 2세대 로봇

##### 2.2.3.1. 하드웨어에 대한 소개

##### 2.2.3.2. 1차 방향코사인 프로그램

##### 2.2.3.3. 중력프로그램

##### 2.2.3.4. 외재적 관점의 가상현실 - 움직이는 큐브

##### 2.2.3.5. 내재적관점의 가상현실 - 움직이는 방

#### 2.2.4. 3세대 로봇

##### 2.2.4.1. 내재적 관점과 외재적 관점의 혼합

##### 2.2.4.2. 마인크래프트의 제어

### 3. 결론

#### 3.1. 결론

#### 3.2. 제언

#### 3.3. 발전 가능성

### 4. 참고문헌

### 5. 부록

#### 5.1. 전체 소스코드(Programmed by 싸컨대사건)

#### 5.2. Driver Program(Open Source)

##### 5.2.1. HiTechnic Gyro Sensor RobotC EV3 Drive (Programmed by Xander)

##### 5.2.2. HiTechnic Accel Senosr RobotC EV3 Driver (Programmed by Xander)

##### 5.2.3. Galaxy Note 1.0 Embedded Acceleration Sensor JAVA Driver(Programmed by Antoine Vianey)

##### 5.2.4. Galaxy Note Embedded Digital Compass Sensor JAVA Driver

##### 5.2.5. -Arduino Uno 6 axis Gyro Sensor Driver.\_\_\_\_ (Programmed by

#### 5.3. 기본적인 명령어 설명

##### 5.3.1. Robot C

##### 5.3.2. Processing

드라이버 프로그램을 제외하고, 이 연구에 쓰인 모든 프로그램은 싸컨대사건이 직접 타이핑하여 알고리즘을 생각하고 100% 창작한 프로그램임을 밝힘.

# 1. 서론

## 1.1. 탐구 배경 / 탐구 목적

컴퓨터의 역사는 브라운관에서부터 시작된다. 1946년 모클리와 에커트가 발명한 브라운관 컴퓨터 에니악(ENIAC)은 그저 간단한 연산을 할 수 있는 정도에 그쳤지만, 그것은 처음으로 정보를 데이터화한 것이었으며 몇십 년 후 정보화 혁명을 불러일으키는 획기적 패러다임의 전환이었다. 이 브라운관 컴퓨터는 1981년 IBM사의 IBM PC를 탄생시켰고, 이 때부터 엔지니어링에 본격적으로 활용되기 시작한 컴퓨터는 1984년 매킨토시 컴퓨터의 등장과 1985년 Windows OS의 등장으로 PC통신의 시대를 열게 된다. 오늘날 컴퓨터가 우리 생활에 미치는 영향은 지대하다. 보급화된 퍼스널 컴퓨터에는 윈도우즈의 PC정책과 매킨토시 컴퓨터에서 처음 도입된 GUI(Graphic User Interface)의 영향이 클 것이다. 매킨토시 이전의, 검은 화면에 문자열만 출력 가능한 PC를 보급화하기는 무척 힘든 일이었을 것이다. 이 GUI를 전략적으로 강화한 결과 윈도우즈 OS는 대대적인 성공을 거두었고, 컴퓨터의 기본이라고 할 수 있는 건 역시 전원 버튼과 파란색 익스플로러 아이콘일 것이다.

GUI는 컴퓨터를 전문적으로 사용하는 프로그래머부터 컴퓨터와는 관련이 없지만 컴퓨터를 활용하면 업무 효율을 높일 수 있는 사무직이나 언뜻 보기에는 컴퓨터와 전혀 관련이 없어 보이는 디자이너, 예술가, 음악가, 그리고 컴퓨터를 처음 만져 보는 학생이나 일반인들 모두가 일상에서 계속 사용하고 있는 요소 중 하나이다. 비록 GUI라는 전문적인 용어는 모르더라도, 윈도우즈 아이콘이나 커서의 모양을 모르는 사람은 아마 없을 것이다. 이는 컴퓨터 사용의 진입 장벽을 획기적으로 낮췄고, 교육 보급화와 엔지니어링 발전 등의 긍정적 결과로 이어졌다.

뿐만 아니라 GUI는 CG라는 혁명적인 변화를 거치면서, 대표적인 컴퓨터의 출력 장치인 모니터에서 구현되었고, 그것은 로봇의 가장 기본적인 구조를 수립했다. 우리가 마우스를 움직이면 모니터의 커서가 움직인다. 워드를 치면 그 결과가 그래픽으로서 모니터에 나타난다. 전문적인 엔지니어링에서 컴퓨터가 연산한 실험 데이터의 그래프 역시 그래픽으로 나타난다. 그래픽이 없는 컴퓨터는 표현할 수 있는 범위가 몹시 줄어들 것이다.

그럼에도 이러한 그래픽은 디자이너의 손이 아닌 컴퓨터 프로그래머의 손에서 탄생한다. 우리가 평범하게 사용하고 있 마우스의 커서 역시 윈도우즈 OS에 내장되어 있는 코드로 발현 되는데, OS안에는 커서의 움직임(입력 장치로서의 마우스의 데이터를 체크하는) task가, 계속해서 무한 루프를 도는 것이다. 그리고 마우스의 데이터를 전송받아서 마우스가 현재 있을 것이라고 생각되는 장소를 연산하여 그곳에 기존에 저장되어 있는 픽셀화된 커서 이미지를 띄운다. 이 때, 그 커서 이미지는 현재 윈도우에 띄어져 있는 모든 창의 위에 존재해야 하고 끊임이 없어야 하며, 사용자가 마우스를 재조정 했을 때에도 변함이 없어야 한다. 윈도우즈 10에 따르면 이 마우스의 데이터값을 GUI로서 출력하는 것만으로도 어마어마하게 긴 프로그램이 쓰인다고 한다.

IT가 발전하고 가상현실의 시대가 도래하면서, 이러한 컴퓨터는 3D의 세계를 만나게 되고 그 중요성은 점점 더 부각되고 있다. 특히나 하드웨어의 발전과 함께 이러한 요소들은 화재대피훈련이나 고급 게이밍 기술, 현미경 디스플레이 출력 등 모든 전자적 데이터를 인간이 인지할 수 있는 시각적 요소로 바꾸는 데 이바지하고 있을 뿐만 아니라 인간과 컴퓨터가 실시간으로 데이터를 통신할 수 있는 방안으로 개발되어 오고있다.

특히나 디자인과 영화 산업, 그 중에서도 애니메이션 부분에서 그래픽은 CG라는 형태로

독특한 문화를 창조하는데, 캐릭터의 본질적 특성은 디자이너의 손에서 창조되지만 그 캐릭터 하나하나의 동작은 텍스트로 표현된 코드로 움직이며 그 이전 시대 프락시노스코프 형식의 제작에서 벗어나 훨씬 더 자연스러운 움직임이 가능하게 된 것이다. 예를 들어, 유행했던 겨울왕국(2014)에서 등장하는 모든 눈은 디자이너가 그린 것이 아니다. 모두 물리엔진 기반 하에 소스코드로 만들어진 컴퓨터 그래픽이다.

	
<p>&lt;그림 1-1-1&gt; 프락시노스코프 클레이 형식의 애니메이션 펭구(1986)</p>	<p>&lt;그림 1-1-2&gt; CG로 더 자연스러운 움직임을 보이는 애니메이션 뽀로로(2003)</p>

프로그래밍을 공부하는 컴퓨터공학도로서, 이러한 컴퓨터를 코딩의 형태로 출력하는 것은 컴퓨터의 가장 기초적인 부분을 공부하는 것의 첫 발걸음이라고 생각한다. 그래서 우리는 EV3를 이용하여서 2차원 객체를 3차원으로 표현하는 그 기본 원리에 대해 탐구 해보고 그 원리를 적용한 아두이노 프로세싱 프로그램을 사용하여 더욱 발전된 형태의 가상현실에 대해 탐구해보았다. 마지막으로 이 아두이노를 이용한 가상현실의 확장성을 이용하여 다른 게임 프로그램에 우리의 알고리즘을 적용해 보아 총체적인 가상현실에 대한 R&D를 진행하였다. 이는 기존의 로봇 모터 제어보다도 훨씬 어렵고 수학적인 부분이 많이 차지하는 부분이지만, 로봇 청소기보다는 스마트폰이 우리 생활에 더 밀접한 관련이 있듯이 우리에게 큰 영향을 끼칠 수 있고 중요한 부분이라고 생각한다. 특히나, 요즘 각광받고 있는 가상현실과 물리엔진 인터페이스 상과, 현재 대부분의 시각적 출력장치의 역할을 맡고 있는 2차원 디스플레이 상에서 이러한 점을 살펴봄으로서, 다가오는 가상현실 시대에 필요한 원천 기술들을 좀 더 효과적으로 구현해보기로 한다.

## 2. 본론

### 2.1. 이론적 배경

#### 2.1.1. 센서의 원리

##### 2.1.1.1. 자이로센서

##### 2.1.1.1.1. 자이로센서의 기본적인 원리

	
<p>&lt;그림 1-1-1-1-1-1&gt; 직접 자이로스코프를 돌리는 레퍼런스 스타일의 자이로스코프 (JG7005 Auto Pilot Internal copy_)</p>	<p>&lt;그림 1-1-1-1-1-2&gt; LEGO MINDSTORM EV3에서 지원하는 자이로센서</p>

자이로센서는 공통적으로 뭔가의 스피네에 의해 발생하는 자이로 효과를 사용하여 원점위치를 역추정, 현재의 방향이 추정하고자 하는 축에서 얼마나 뒤틀렸는지를 역산출하는 기계다. 그래서 방향성을 측정할 때 쓰게 된다.

여기서 말하는 자이로 효과는, 물체가 고속으로 회전하여 다량의 키네틱을 가지게 될 때 스피네 축 방향으로의 정렬을 유지한다는 것으로, 자이로스코프의 회전축은 변하지 않는다는 것을 의미한다. 이는 일종의 관성력인 회전관성모멘트때문인데, 자이로스코프는 이것을 이용하여 고속으로 물체를 회전시킨 다음 그것을 기준축 삼아 얼마나 돌아갔는지를 측정하는 것이다. 그렇기 때문에 자이로스코프에 가해지는 토크는 90도 뒤틀려서 출력되게 된다.

이제 이 연구에서 사용된 센서에 대해 알아보자.

##### 2.1.1.1.2. HiTECH LEGO MINDSTORMS NXT Gyro Sensor

이 보고서에서 이 부분은 그다지 중요하지 않지만, 뒤에 후술할 프로그램 알고리즘의 원리에 서 자이로센서의 원리를 알지 못하면 이해하기 힘든 부분이 많기 때문에 보고서의 요지를 흐리지 않는 정도까지만 서술하기로 한다.

EV3에는 기본 센서로 자이로센서가 내장되어 있지만 RCX - NXT 호환 케이블로 케이블 연결을 할 수 없는 단점이 있다. 우리가 앞으로 구현할 그네 로봇이나 바퀴벌레 로봇 등은 잘 구부러지지 않고 짧은 NXT - EV3케이블로는 주변의 영향을 너무나도 많이 받기 때문에 부득이하게 NXT의 추가 센서인 자이로센서를 사용할 수 밖에 없었다.

자이로센서는 기본적으로, 센서의 윗부분과 수평한 평면에서 센서가 움직이는 각속도를 측정한다. 다른 말로는 각속도센서라고도 한다. 안에는 자이로스코프라는 구조물이 들어 있어서

각속도를 측정하게 해 주지만, 자이로스코프는 주변의 충격에 영향을 많이 받도록 일부러 설계되어 있기 때문에 각속도를 측정하는 부분이 모터와 밀접하게 연결되어 있으면 값에 오류가 난다. 또, 각속도를 측정하기 전에 흔들리는 자이로스코프가 0점을 잘 맞출 수 있도록 약 1초간 기다려줘야 한다. 즉, 로봇 프로그램에 기다리는 프로그램(딜레이)을 넣어줘야 한다.

	
<p>〈그림 2-1-1-1-2-1〉 HiTECH LEGO MINDSTORMS NXT Gyro Sensor</p>	<p>〈그림 2-1-1-1-2-2〉 자이로스코프 구조물</p>

그러나 자이로센서를 이용하는 많은 사람들은 각속도(각이 변한 속도)를 측정하는 데에 자이로센서를 사용하지는 않는다. (그러나 센서가 기본적으로 각을 측정하는 원리는 각속도 측정이다.) 이 각속도 센서값을 잘 가공하면, 한마디로 적분하면 각이 변한 정도를 측정할 수 있다. 그렇게 해 주는 프로그램을 우리는 Gyrodriver라고 한다. 이 GyroDriver에 관한 것은 부록을 참고하길 바란다.

#### 2.1.1.2. 가속도센서

가속도 센서는 거리를 두 번 미분한 가속도를 측정하는 센서이다. 가속도 센서는  $x$ ,  $y$ ,  $z$ 축으로 부터의 가속도를 측정하게 되는데 이 세 개의 축은 센서를 기준으로 한 축으로 센서가 기울어 지면서 바뀌게 된다. 우리는 지구상에서 가만히 있어도 항상 중력 가속도를 받기 때문에 이론적으로는 가속도 센서를 정 방향으로 가만히 둔다면 항상  $z$ 축으로의 9.8의 가속도를 받게 된다. 마찬가지로 외부에서 힘을 주지 않은 상태에서 이 센서를 이리저리 기울이게 된다면 가속도 센서의  $X, Y, Z$ 축의 가속도를 합성한 가속도는 항상 9.8이 나오게 된다. 이러한 성질 때문에 우리는 가속도 센서를 Tilt Sensor이라고 부르기도 한다.  $x$ ,  $y$ ,  $z$ 축의 가속도를 통해 얼마나 기울어졌는지를 계산할 수 있기 때문이다.

하지만 이 센서도 오차가 존재하기 때문에 오차가 누적되어 드리프트 현상이 나타나기도 한다. 이 때문에 가속도 센서는 주로 자이로 센서와 같이 사용되어 서로의 오차를 보완해준다.

##### 2.1.1.2.1. Arduino 6-axis Acceleration Sensor MPU-6050 GY-251



<그림 2-1-1-2-1-1>  
아두이노용 6축 가속도센서

아두이노용 6축 자이로센서는 6가지의 센서값을 가지며 이는 모두 아두이노 프로그램에서 SensorRawValue를 변환해 배열 데이터로 프로세싱에 전송할 수 있다. 6가지의 데이터값은 다음과 같다.

배열이름	내용
teapotPacket	아두이노에서 프로세싱으로 보내는 원시값
q	원시값 중 필요한 값만 다시 고르는 배열
gravity	가속도값(중력)
ypr	yaw, pitch, roll값
euler	오일러 방식으로 나타낸 값

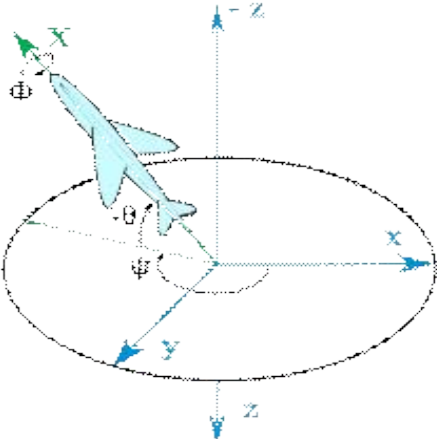
<표 2-1-1-2-1-2>  
아두이노용 6축 자이로센서의 데이터값

우리는 이 값들 중에 주로 ypr과 gravity값을 사용할 것이다.

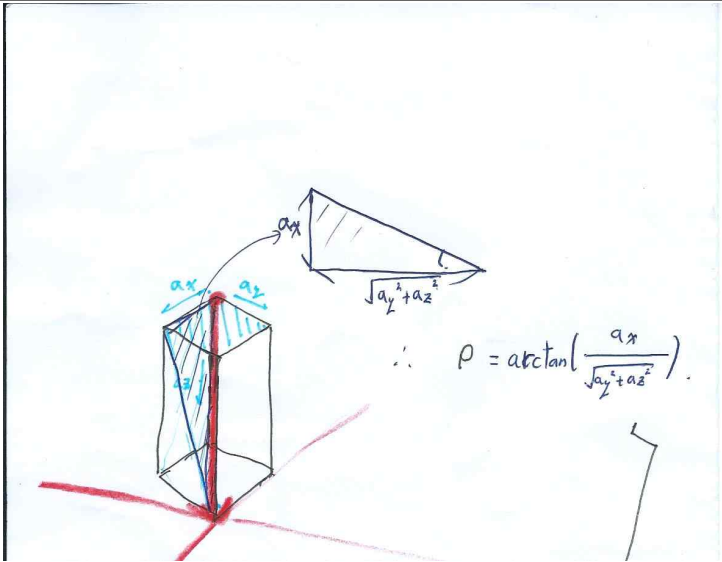
## 2.1.2. 기본 알고리즘

### 2.1.2.1. 수학적으로 바라본 3축 좌표계



	$\rho = \arctan\left(\frac{Ax}{\sqrt{Ay^2 + Az^2}}\right)$ $\phi = \arctan\left(\frac{Ay}{\sqrt{Ax^2 + Az^2}}\right)$ $\theta = \arctan\left(\frac{\sqrt{Ay^2 + Az^2}}{Ax}\right)$
<p>&lt;그림 2-1-2-1-1&gt; yaw, pitch, roll</p>	<p>&lt;그림 2-1-2-1-2&gt; yaw, pitch, roll 유도공식</p>

위 <그림 2-3-22>의 공식의 유도과정이다. 센서의 중심을 기준으로 ax, ay, az 벡터를 모두 합성하면 정지 상태에서는 9.8의 값을 가져야 한다. 그렇기 때문에 우리는 ax, ay, az의 크기를 가지고 대각선이 9.8인 직육면체를 상상할 수 있다.

	<p>&lt;그림 2-1-2-1-3&gt; 유도공식의 유도</p>
---	--

## 2.1.3. 현재의 가상현실

### 2.1.3.1. 오쿨러스 리프트



<사진 2-1-3-1-1>

픽 스타터에서 출시한 옵시물러스(2012.08)

오culus 리프트(영어: Oculus Rift)는 가상현실 게임을 위한 장비이다. 헤드셋을 쓰면 헤드셋이 머리의 움직임을 실시간으로 감지하여 머리가 어느 방향으로 움직이든지 그 방향으로의 시각을 제공한다. 또한 각각의 오른쪽, 왼쪽 렌즈는 오목하게 굽어진 파노라마 디스플레이 영상을 제공한다. 이는 넓은 시야각을 제공하여 눈동자를 움직여도 가상 현실의 디스플레이를 볼 수 있다. 헤드를 트래킹하는 기술과 양 쪽 눈에 제공되는 각각의 디스플레이는 마치 사용자가 가상현실에 들어와있다는 착각을 하게 만드는 역할을 하게 된다. [1] 하지만 시야각은 110°에 한정되어있고, 헤드 트래킹의 속도 문제, 시각과 청각 정보의 불일치 문제, 모션을 인식 못하는 문제 등으로 완벽한 가상현실 구현에 한계가 있고, 사용 중 멀미 현상을 보이는 사용자가 많다는 단점이 있다. 이러한 단점은 기술의 발전으로 해결되어야 할 부분이다.

#### 2.1.3.2. 마인크래프트



<사진 2-1-3-2-1>

마인크래프트 1.8.8 실행화면

마인크래프트는 2009년부터 Team Mojang에서 내놓은, 1단위 블록들로 이루어진 물리엔진을 적용한 가상현실 그래픽이다. 학생으로서 그래픽 가상현실을 제작하는 것은 무리가 있기에 이미 있는 2차원 디스플레이상에 표현된 가상현실 플랫폼을 이용하여 3차원 객체의 움직임을

표현하는 장으로 활용하기로 했다.

## 2.2. 연구내용

### 2.2.1. 연구의 방향성

가상현실을 구현할 때, 우리는 세 가지로 가상현실 디스플레이를 구분지을 수 있다. 첫째는 외재적 관점의 가상현실이다. 외재적 관점의 가상현실에서는 우리가 디스플레이 상의 상자를 관찰하거나 어떠한, 특히나 3D 프린팅 기술에서 프린팅 될 객체를 2차원 디스플레이에서 3차원적으로 먼저 감상할 수 있는 기술에서 많이 응용된다. 사용자를 가상현실을 우려볼 수 있는 신적인 존재로 가정한다.

둘째는 내재적 관점의 가상현실이다. 대부분 게임에서 많이 쓰이며, 가상현실 디스플레이는 현재 사용자가 방 안이나 어떤 세계에 안에 들어와 있다고 가정하고 시선이 변하면 사용자를 둘러싼 환경도 변한다. 사용자를 가상현실 안에서 활동하는 인간적인 존재로 가정한다.

그러나 대부분의 가상현실은 이 두 가지를 혼합한 경우가 많은데, 그것이 세 번째이다. 세 번째의 가상현실은 굉장히 복잡한 매커니즘을 띄는데, 이유는 위에 서술한 두 가지의 가상현실을 동시에 연산하여 구현해 내야 하기 때문이다. 가장 간단한 가상현실은 방 안에 놓여 있는 상자를 관찰하는 가상현실이 될 것이다.

		
<그림 2-2-1-1> 외재적 관점의 가상현실	<그림 2-2-1-2> 내재적 관점의 가상현실	<그림 2-2-1-3> 혼합형 가상현실의 예로서의 마인크래프트

우리의 로봇은 크게 1세대 로봇, 2세대 로봇, 3세대 로봇으로 나뉘는데, 그 차이점은 다음과 같다.

	사용 하드웨어	사용 센서	사용 소프트웨어
1세대 로봇	MINDSTORM EV3	2-axis EV3 Gyro	RobotC 4.32
2세대 로봇	Galaxy Note 1.0	Galaxy Note 1.0 Embedded Sensor	Android APDE Processing 2.0
3세대 로봇	Arduino UNO	MPU 6050 6-axis Accelerometer GY-251	Arduino Program Processing 3.1
<그림 2-2-1-4> 1세대, 2세대, 3세대 로봇의 차이			

1세대 로봇은 EV3 자이로센서 두 개에서 두 개의 축을 가진 각도 값을 받아와 2차원 디스플레이 상에 표시한다. 사용자가 기울이는 정도에 따라 모양이 달라지는 직교선, 삼각형, 사다

리꼴을 표시했으나 구조상 3개의 축을 사용할 수 없었기에 새로운 소프트웨어를 사용해야 했다.

2세대 로봇은 3축 가속도센서와, 3축 자이로센서로는 z축  $\psi$ 값을 측정할 수 없었기에 compass 센서를 사용하고, 3축 좌표계를 사용하기 위해 안드로이드 상에서 프로세싱을 이용해 사용자가 기울이는 정도에 따라 모양이 달라지는 큐브(외재적 관점의 3차원 가상현실), 사용자의 시선에 따라 모양이 달라지는 방 안(내재적 관점의 3차원 가상현실), 오일러 법칙으로 구현한 벡터, 가상중력에 따라 떨어지는 공을 만들었다. 이론상으로 이러한 방식으로 모든 가상현실을 구현할 수는 있지만, 우리는 아주 시간을 많이 들여야만 구현할 수 있는 그래픽을 구현할 수가 없었다. 그래서 기존에 있는 그래픽 플랫폼을 사용하여 제어를 만드는 방법을 택했다. 그러기 위해서는 시리얼 통신을 통해 프로세싱을 사용해야만 했다.

더 많은 기능을 사용하기 위해 6축 자이로센서를 매개하는 아두이노 우노와 프로세싱 간의 시리얼 통신을 사용, 내재적 관점과 외재적 관점의 혼합, 그리고 웨어러블 하드웨어를 활용한 마인크래프트 제어(방향전환, 이동)를 성공시켰다.

## 2.2.2. 1세대 로봇

### 2.2.2.1. 1축 자이로를 사용한 2차원 객체의 2차원 GUI 디스플레이화

우선 첫 번째 가상현실인 외재적 관점의 가상현실을 보도록 하자.

다음 소프트웨어에 사용된 하드웨어는 모두 다음과 같다.



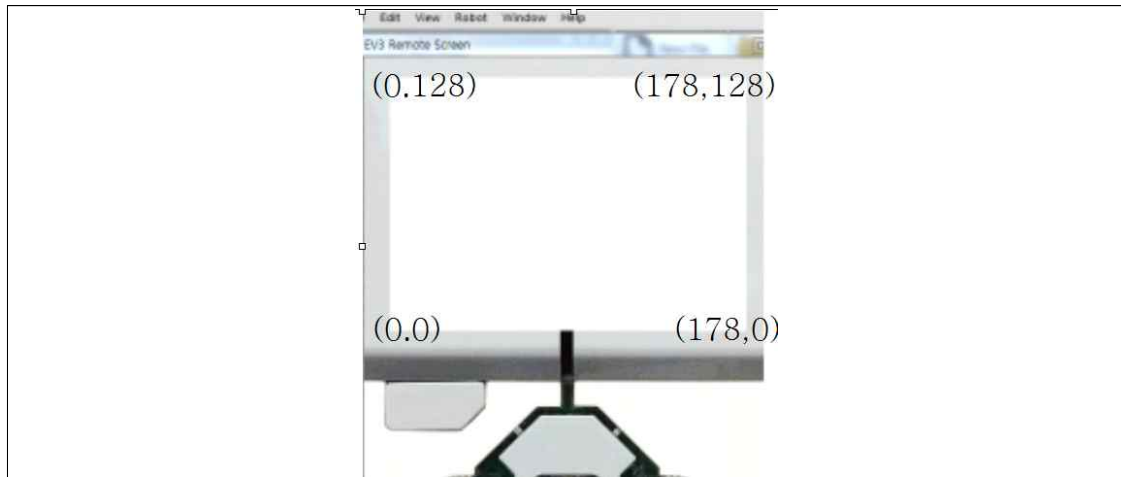
<그림 2-2-2-1-1>

하드웨어

S1에 연결된 EV3 Gyro Sensor는 x축의 자이로센서값을 측정한다면, S2에 연결된 EV3 Gyro Sensor는 z축의 자이로센서값을 측정한다고 볼 수 있다.

3차원 객체를 2차원 디스플레이화하는 하드웨어에서는 소프트웨어가 얼마나 많은 축을 표현할 수 있는가와 디스플레이의 성능이 몹시 중요하다. EV3 디스플레이는 가로로 178픽셀, 세로로 128픽셀로 이루어져 있으며, 좌측 하단을 (0,0)원점으로 설정하고 원점을 직접적으로 바

꾸는 translate()명령어는 지원되지 않는다. 모든 픽셀은 0이나 자연수로 표현된다.



<그림 2-2-2-1-2> EV3 디스플레이의 구조

```

1  #pragma config(Sensor, S1, , sensorEV3_Gyro)
2  #pragma config(Sensor, S2, , sensorEV3_Gyro)
3  /**!!Code automatically generated by 'ROBOTC' configuration wizard      !!*/
4
5  //O(80,20)
6  //a=20, b=40
7  int theta, theta2;
8  float thetain, thetain2;
9  task main()
10 {
11     wait1Msec(10);
12     resetGyro(S1);
13     wait1Msec(10);
14     resetGyro(S2);
15     wait1Msec(10);
16     displayTextLine(2, "chcking the gyro");
17     wait1Msec(5000);
18     eraseDisplay();
19     if(abs(sensorValue[S1])>5 || abs(sensorValue[S2])>5)
20     {
21         displayTextLine(2, "ERROR!!");
22         wait1Msec(5000);
23         stopAllTasks();
24     }
25     wait1Msec(100);
26     resetGyro(S1);
27     wait1Msec(5);
28     resetGyro(S2);
29     wait1Msec(5);
30     while(1)
31     {
32         theta=SensorValue[S1];
33         thetain = theta*PI/180;
34         theta2 = SensorValue[S2];
35         if(sensorValue[S2]>90)
36             theta2=180-theta2;
37         thetain2 = theta2*PI/180;\
38         drawLine(80-20*cos(theta2), 40+20*sin(theta2), 80+20*cos(theta2), 40-20*sin(theta2));
39         //drawCircle(80, 20, ceta2); //drawline(80+40*sin(ceta2)-20*cos(ceta2),20+40*cos(ceta2)-20
40         drawLine(80,40,80+0.5*theta2*sin(theta2),40+0.5*theta2*cos(theta2));
41         wait1Msec(3);
42         eraseDisplay();
43     }
44 }

```

<그림 2-2-2-1-3> 1축 자이로를 사용한 2차원 객체의 2차원 GUI

1~2. 센서 중 Ev3 자이로센서를 정의해준다. #pragma config(sensor, sensorPort, Sensorname, sensortype)은 센서나 모터를 정의하는 명령어이다.

5~6. 원점과 만들어질 직선의 길이를 정의한다. 이 프로그램에서 원점은 80, 20이고 직선의 길이는 40이다. 원점이 80,20인 이유는 EV3 디스플레이 상 왼쪽 하단부가 0,0 위치값을 가지고 픽셀 하나당 1의 이산적 값을 가지며, 모두 양수의 값을 가지기 때문에 디스플레이를 우리가 흔히 사용하는 종이로 볼 때 사용자가 가장 보기 쉬운 중앙부 하단의 원점이 약 80,20 정도이기 때문이다.

7~8. 필요한 변수를 선언한다. S1이 theta, S2가 theta2로 선언되고, sin이나 cos안에 연산을 시킬 것이기 때문에 라디안 값으로 변화할 필요가 있다. 라디안 값은 정수값 변수인 int나 long을 사용하는 것과 달리 float변수를 사용해야 한다. 여기에는 다음과 같은 수식이 사용된다.

$$\theta^{\circ} = \left( \theta \times \frac{\pi}{180} \right) rad$$

<그림 2-2-2-1-4> 라디안값 도출공식 (고등학교 미적분 II)

위 그림에서 theta는 센서값에 해당된다.

11~15. 자이로값을 초기화시키는데, 자이로값은 초기화시킬 때 딜레이를 많이 주면 안정화되기 때문에 10ms만큼씩 딜레이를 줬다. resetGyro(SensorPort);는 센서종류중에서도 자이로센서를 초기화시키는 센서이고, wait1Msec(time);은 time변수에 들어가는 숫자에 1000분의 1초를 곱해서 아무것도 하지 않고 쉬는 명령어이다. RobotC 3에서는 딜레이 명령어가 이 명령어밖에 없었으나, RobotC 4로 업그레이드되면서 Sleep(time);과 구분된다. 이 Sleep(time);명령어는 Xander의 드라이버 소스코드에서 많이 찾아볼 수 있는데, wait1Msec은 주로 현재상태를 유지하는데 쓰인다. 예를 들어, motor[motorA]=100; wait1Msec(1000);이면, 모터 A 포트에 전력 100을 주면서 1초간 쉬는 코드이지만, motor[motorA]=100; sleep(1000);이면 아무것도 하지 않고 1초간 쉬는 코드라고 볼 수 있다.

16~24. 11~15줄에서 자이로센서값을 초기화시켰기 때문에 정상적으로 코드가 실행되고 센서가 제대로 구동되고 있다면 이 단계에서 자이로센서값은 0이어야 한다. 그리고 16줄에서 checking the gyro라는 메시지를 화면에 출력함에 따라 사용자는 EV3브릭을 자이로센서를 체크할때까지 건드리지 않게 된다. 이 상황에서 Ev3센서값이 처음과 변함없이, 혹은 사소한 오차로 5 이내로 변한다면 이 센서는 제대로 구동되고 있는 것이다. 그러나 워낙 저가형 센서를 쓰기 때문에 가끔씩 Ev3센서가 표류하는 현상이 있다. 이 내용은 부록을 참고하길 바란다.

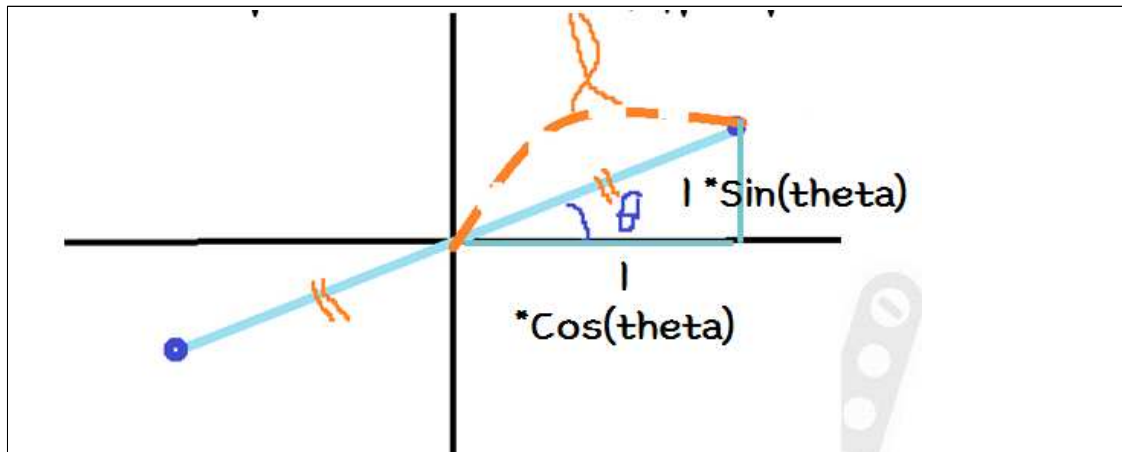
DisplayTextLine(TextLine,"Text");는 디스플레이에 텍스트를 띄우는 함수이고, abs(변수)는 포인터 안의 변수값이나 상수값을 절대값으로 변환시켜주는 함수이다.



25~29. 자이로센서를 다시 한 번 초기화시킨다.

30~34. 이 while문 안에서 영원히 돌게 된다. 즉, 계속해서 GUI를 화면에 띄우게 된다. 아까 변수선언했던 대로 라디안 값으로 변환시켜서 넣는다. 라디안값으로 변환시키는 명령어는 무한 루프 안에 반드시 있어야 한다. 계속해서 센서값 피드백이 일어나야 하기 때문이다.

38. Drawline(x1, y1, x2, y2):함수로, 이 프로그램에서 가장 핵심적인 부분이다.



<그림 2-2-2-1-5> 1축 자이로 프로그램의 원리 설명

그림의 세타값을 첫 번째 자이로센서의 라디안값이라고 하면, 우리가 임의로 정한 l 값 (line=20)에 따라 좌표평면에 다음과 같은 점을 정의 할 수있다. 즉, 원점을 (80,20)으로 두었으므로 l의 끝값은  $(80+l*\cos(\theta), 20+l*\sin(\theta))$ 이 되고, 나머지 한 점은 (80,20)에 대해 대칭시키면 된다.

42. 이 모든 과정은 Erasedisplay();라는 명령어로 지워져야 한다. 디스플레이에 선을 띄우면 프로그램은 자신이 띄웠다는 사실을 프로그래머가 지정해주지 않는 한 잊어버리고, 어떤 메모리에도 저장되지 않는다. 그러므로 새로운 라인을 띄워도 기존의 라인과 겹쳐서 보일 것이다. 이러한 것을 방지하기 위해 우리 프로그램의 알고리즘은 그리고, 지우고, 새로운 센서값을 받아서 그리고, 지우고, 또 새로운 센서값으로 그리고, 지우는 것을 무한히 반복하게 된다.



<그림 2-2-2-1-6> 실행화면



<QR코드 2-2-2-1-7>

실행화면

## 2.2.2.2. 2축 자이로를 사용한 3차원 객체의 2차원 GUI 디스플레이화

### 2.2.2.2.1. 직교

지금까지는 1축 자이로를 사용했기 때문에 센서가 1개만 씌웠다. 그러나 이번에는 2축 자이로를 사용해서 2개의 센서를 사용하게 된다.

```
1  #pragma config(Sensor, S1, , sensorEV3_Gyro)
2  #pragma config(Sensor, S2, , sensorEV3_Gyro)
3  /**!!Code automatically generated by 'ROBOTC' configuration wizard !!*/
4
5  //o(80,20)
6  //a=20, b=40
7  int theta, theta2;
8  float thetain, thetain2;
9  task main()
10 {
11     wait1Msec(10);
12     resetGyro(S1);
13     wait1Msec(10);
14     resetGyro(S2);
15     wait1Msec(10);
16     displayTextLine(2, "chcking the gyro");
17     wait1Msec(5000);
18     eraseDisplay();
19     if(abs(sensorValue[S1])>5||abs(sensorValue[S2])>5)
20     {
21         displayTextLine(2, "ERROR!!");
```



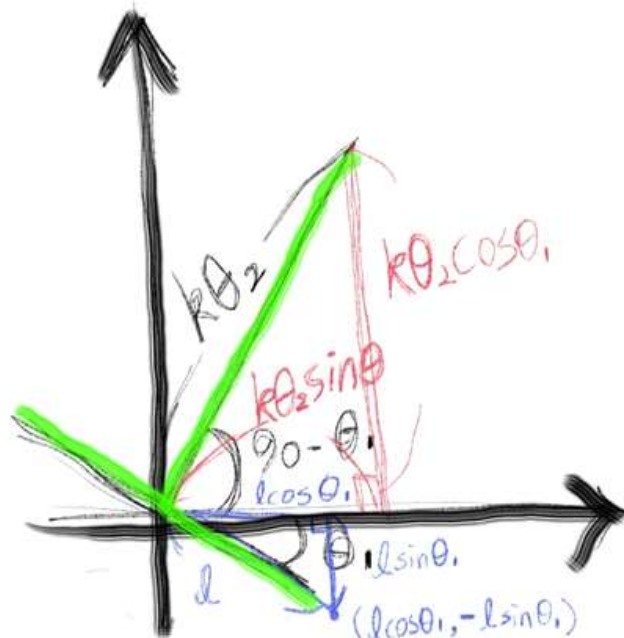
```

22     wait1Msec(5000);
23     | stopAllTasks();
24     }
25     wait1Msec(100);
26     resetGyro(S1);
27     wait1Msec(5);
28     resetGyro(S2);
29     wait1Msec(5);
30     while(1)
31     {
32         theta=SensorValue[S1];
33         thetain = theta*PI/180;
34         theta2 = SensorValue[S2];
35         if(sensorValue[S2]>90)
36             theta2=180-theta2;
37         thetain2 = theta2*PI/180;\
38         drawLine(80-20*cos(thetain), 40+20*sin(thetain), 80+20*cos(thetain), 40-20*sin(thetain));
39         //drawCircle(80, 20, ceta2);//drawline(80+40*sin(cetain)-20*cos(cetain),20+40*cos(cetain)-20
40         drawLine(80,40,80+0.5*theta2*sin(thetain),40+0.5*theta2*cos(thetain));
41         wait1Msec(3);
42         eraseDisplay();
43     }
44 }

```

<소스코드 2-2-2-2-1-1> 직교로 띄워지는 소스코드

위의 소스코드와 달라진 점은 40줄의 추가이다.



<그림 2-2-2-2-1-2> 직교 소소코드의 원리

그림의 연두색 부분이 디스플레이에 띄워질 부분이다.  $l$ 을 표현하는 부분은 1차 소프트웨어와 같지만, 두 번째 자이로센서값을 띄우기 위해선 많은 고민이 필요했다. 우선 두 번째 자이로센서값은 처음에 실행했을때 z축이 하늘을 향해 있는 방향이라고 생각했다. 그러므로 위에서 바라봤을 때 제트축은 길이가 0일 것이다. 그러나 그 z축이 점점 더 기울어지면서 두 번째 자이로센서값이 커질수록 두 번째 축의 길이는 점점 길어진다. 원래대로라면 z축을 생각해서 각 축마다의 방향코사인을 생각해 xy평면에 정사영을 내리려고 했지만 그것은 너무 복잡해서 포기했었다. 그러나 나중에 삼각형과 사다리꼴을 만들 때에 성공했다. 그러므로 두 번째 축의 길이는 두 번째 자이로센서값에 비례한다고 생각해,  $k$ 값을 상수(적당히 0.5)로 두기로 했다. 첫 번째 축과 두 번째 축은 항상 수직이어야 하므로 두 번째 축의 끝값을 설정하는 각은 (90

도-첫번째 자이로센서값)이 된다. 여기서 RobotC와 EV3 CPU의 연산을 빠르게 하기 위해 다음과 같은 공식을 이용한다.

$$\sin(90^\circ - \theta) = \cos\theta$$

$$\cos(90^\circ - \theta) = \sin\theta$$

<수식 2-2-2-2-1-3> sin cos 변환공식 (미적분 II)



<그림 2-2-2-2-1-4>

실행화면



<QR코드 2-2-2-2-1-5>

실행화면

이 소프트웨어에서는 다음과 같은 문제가 있었다.

방향코사인 등 정확한 방법을 측정한 게 아니라 우리가 임의로 설정한 원근법 내에서 작동하는 소프트웨어이기 때문에 -90도~90도 사이에서는 잘 움직이지만 그 범위를 벗어나면 값을 수정하는 과정 역시 번거로울뿐더러 범용적이지 않다. 또한 실제 자연에서 움직이는 3차원 객체의 모습과 다르기 때문에 우리는 처음에 포기했던 정사영을 내리기로 했다.

#### 2.2.2.2.2. 삼각형

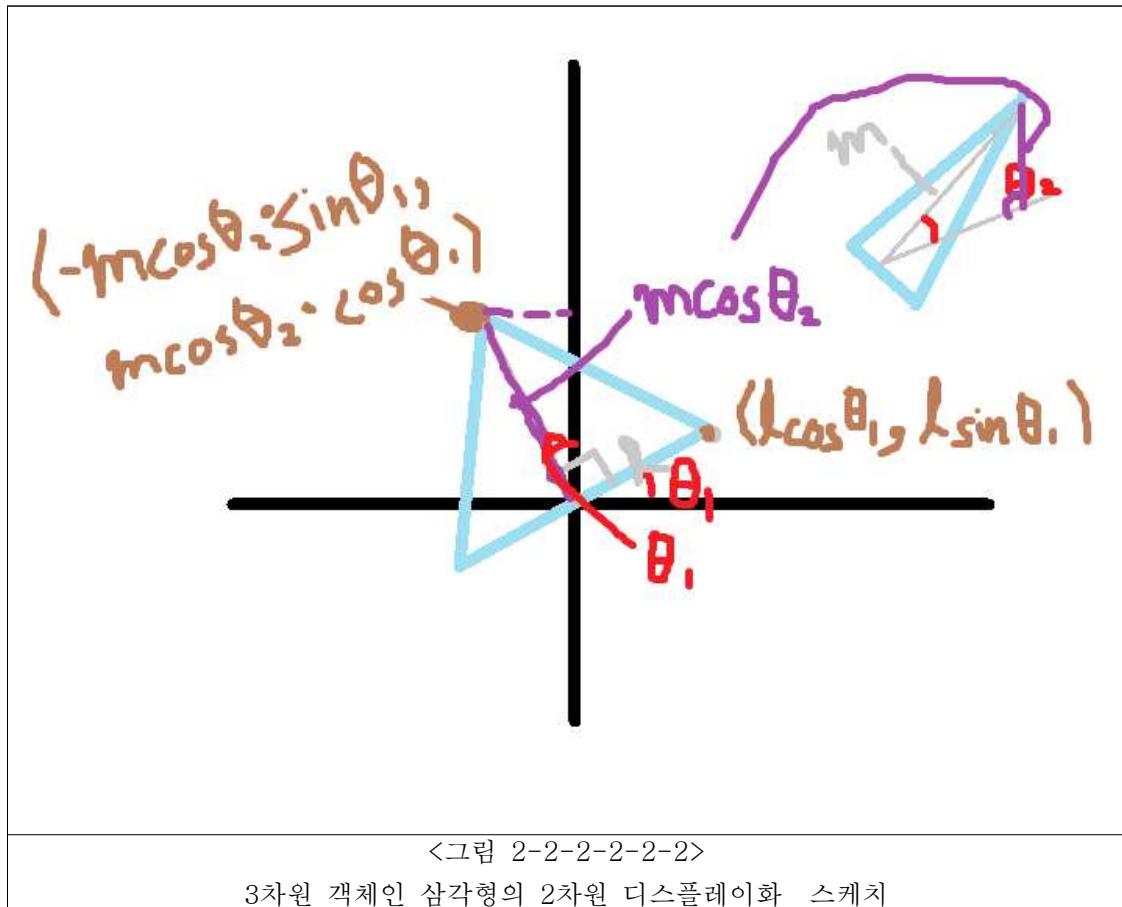
```

3  task main()
4  {
5      resetGyro(S1);
6      resetGyro(S2);
7      int a, b; //,c,d;
8      float ga1, ga2;
9      while(1)
10     {
11
12         ga1 = SensorValue[S1]*PI/180;
13         ga2 = - SensorValue[S2]*PI/180;
14
15         a = 80-25*sin(ga1)*sin(ga2); //+20*(1-cos(ga2))*cos(ga1);
16         b = 40+25*sin(ga2)*cos(ga1); //+20*(1-cos(ga2))*sin(ga1);
17         //c = 80-25*sin(ga1)*sin(ga2)-20*(1-cos(ga2))*cos(ga1);
18         //d = 40+25*sin(ga2)*cos(ga1)-20*(1-cos(ga2))*sin(ga1);
19
20         drawLine(80-30*cos(ga1),40-30*sin(ga1),80+30*cos(ga1),40+30*sin(ga1));
21         drawLine(80-30*cos(ga1),40-30*sin(ga1),a,b);
22         //drawLine(c,d,a,b);
23         drawLine(a,b,80+30*cos(ga1),40+30*sin(ga1));
24         wait1Msec(80);
25         eraseDisplay();
26
27     }
28 }

```

<그림 2-2-2-2-2-1>

3차원 객체인 삼각형의 2차원 디스플레이화



<그림 2-3-4>에서 볼 수 있는 것같이 이 소프트웨어는 xy평면에 내린 정사영과 같다. 두 번째 자이로센서값을 현재 실제 물체가 있는 평면 a와 디스플레이에 비춰지는 평면 b 사이의 이면각으로 정의하였다.

4,5. 자이로 값을 초기화 하였다. 이때 포트 1번에 있는 자이로는 1차 소프트웨어의 축이고, 2번 포트에 있는 자이로는 z축에 관한 자이로이다.

12,13. 두 자이로 값이 도로 나와서 라디안 값으로 변환했다.

15,16. 그림 삼각형의 끝점을 (a, b) 로 놓았다. <그림 2-3-4> 참고.

20. 삼각형의 밑변을 그리는 명령이다.

21. 삼각형의 왼쪽 부분을 그리는 명령이다.

22. 삼각형의 오른쪽 부분을 그리는 명령이다.

25. 디스플레이를 초기화하는 명령이다.



삼각형은 2차원에서 효율적으로 3차원을 표현할 수 있는 도형이지만 입체감이 잘 살지 않는다는 단점이 있다. 그래서 우리는 흔히 3차원표현용 2차원객체로 쓰이는 사다리꼴을 이용하기로 했다. 원근법에 따라, 객체가 뒤로 뉘어져서 시선과 비슷해질 수록 사다리꼴 윗변의 길이는 짧아졌다가 길어진다. 이 소프트웨어는 발전형이기 때문에 <그림 2-3-9> 스케치와 함께 참고해서 보면 편하다.

#### 2.2.2.2.3. 사다리꼴

```

3  task main()
4  {
5      resetGyro(S1);
6      resetGyro(S2);
7      int a, b, c, d;
8      float ga1, ga2;
9      while(1)
10     {
11
12         ga1 = SensorValue[S1]*PI/180;
13         ga2 = - SensorValue[S2]*PI/180;
14
15         a = 80-25*sin(ga1)*sin(ga2)+20*(1-cos(ga2))*cos(ga1);
16         b = 40+25*sin(ga2)*cos(ga1)+20*(1-cos(ga2))*sin(ga1);
17         c = 80-25*sin(ga1)*sin(ga2)-20*(1-cos(ga2))*cos(ga1);
18         d = 40+25*sin(ga2)*cos(ga1)-20*(1-cos(ga2))*sin(ga1);
19
20         drawLine(80-30*cos(ga1),40-30*sin(ga1),80+30*cos(ga1),40+30*sin(ga1));
21         drawLine(80-30*cos(ga1),40-30*sin(ga1), c, d);
22         drawLine(c, d, a, b);
23         drawLine(a, b, 80+30*cos(ga1),40+30*sin(ga1));
24         wait1Msec(80);
25         eraseDisplay();
26
27     }
28 }

```

<그림 2-2-2-2-3-1>  
 4차 소프트웨어 소스코드


12,13 각도를 라디안 값으로 바꿨다.

15,16,17,18 좌표값을 계산했다.

20~24 사다리꼴로 표현하였다.





	
<p style="text-align: center;">&lt;QR코드 2-2-2-2-3-3&gt; 실행화면</p>	

지금까지 EV3와 RobotC 4.32를 이용한 소프트웨어에서는 2차원 좌표로 센서값을 나타냈어야 했기 때문에 정사영을 내렸다. 그러나 본질적인 의미로의 3차원 객체는 사실 각 축 (x축, y축, z축과 이루는) 방향코사인이다. 이러한 방법을 사용하기 위해서는 3차원 좌표축이 필요한데, 현재 우리가 다룰 수 있는 EV3 프로그램들은 3차원 좌표축을 지원하지 않았다. 또한 EV3 자이로센서 자체의 문제점도 심했고 가속도센서를 적분하기엔 센서의 오차가 너무 크다고 판단하여 우리는 다른 플랫폼을 사용하여 방향벡터로 외재적 관점의 가상현실을 구현하기로 했다.

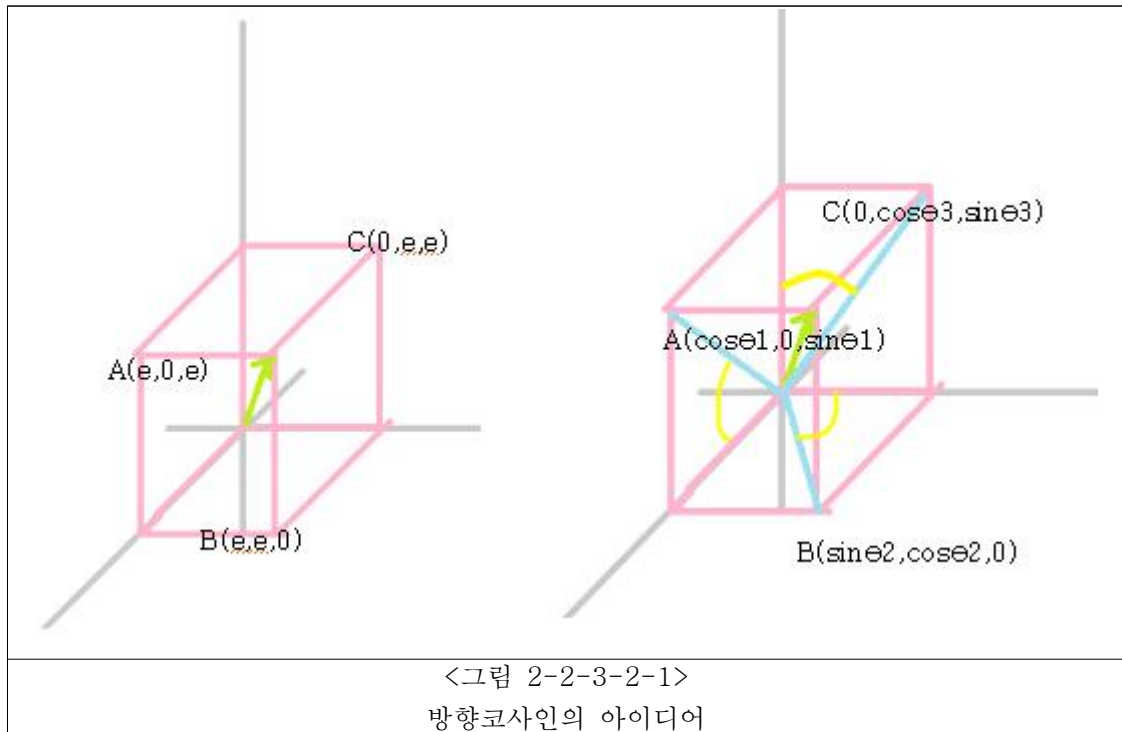
### 2.2.3. 2세대 로봇

#### 2.2.3.1. 하드웨어에 대한 소개

		
<p style="text-align: center;">&lt;그림 2-2-3-1-1&gt; 레고로 만든 헤드기어</p>	<p style="text-align: center;">&lt;그림 2-2-3-1-2&gt; 레고로 만든 헤드기어</p>	<p style="text-align: center;">&lt;그림 2-2-3-1-3&gt; 레고로 만든 헤드기어</p>

HMD(Head mounted Display)란 안경처럼 머리에 쓰고 대형 영상을 즐길 수 있는 영상표시 장치다. 휴대하면서 영상물을 대형화면으로 즐기거나 수술이나 진단에 사용하는 의료기기에 적용할 수 있는 차세대 영상표시 장치다.

#### 2.2.3.2. 1차 방향코사인 프로그램



다음 3차원 그림에서 나타내고 싶은 벡터값이 초록색 벡터라고 하자. 이 벡터를 평면에 나타내려면 우리가 RobotC에서 취한 방법은 xy평면에 정사영을 내리는 것이었다. 그러나 이제 센서값이 3개가 되었고 정사영을 내리는 것이 더 복잡해진 만큼 우리는 다른 방법을 생각해야만 했다. 나타내고자 하는 벡터값이 초록색이라면, e를 길이가 1인 단위벡터를 이루는 각각의 벡터값이라고 설정하고 세 개의 벡터 A벡터, C벡터, B벡터를 합성한 값은 우리가 나타내고자 하는 초록색 X벡터의 두 배라고 볼 수 있다. 한 벡터의 값을 정하고 나면 물체에 따라 그 벡터값의 실수배를 하면 되므로 어떠한 값이라도 표현할 수 있다.

<그림 2-3-25>의 노란색 값들을 센서값이라고 하자. A벡터가 x축과 이루는 센서값을  $\text{axis.x.value} = \theta_1$ 값, B벡터가 y축과 이루는 센서값을  $\text{axis.y.value} = \theta_2$ 값, C벡터가 z축과 이루는 센서값을  $\text{axis.z.value} = \theta_3$ 값으로 두면 각 벡터의 성분은  $C(0, \cos\theta_3, \sin\theta_3)$ ,  $A(\cos\theta_1, 0, \sin\theta_1)$ ,  $B(\sin\theta_2, \cos\theta_2, 0)$ 와 같이 된다. 그리고 나타내고자 하는 초록색 X벡터의 성분값은  $X((\cos\theta_1 + \sin\theta_2)/2, (\cos\theta_3 + \cos\theta_2)/2, (\sin\theta_3 + \sin\theta_1)/2)$ 과 같이 된다. 그러나 실수배는 어차피 나중에 가공할 것이므로 생략해도 좋다.

$$\vec{X} = (\cos\theta_1 + \sin\theta_2, \cos\theta_3 + \cos\theta_2, \sin\theta_3 + \sin\theta_1)$$

<그림 2-2-3-2-2>  
오일러 방식에 따른 방향코사인의 최종 공식

그리고 그 원리에 따른 프로그램은 다음과 같다.



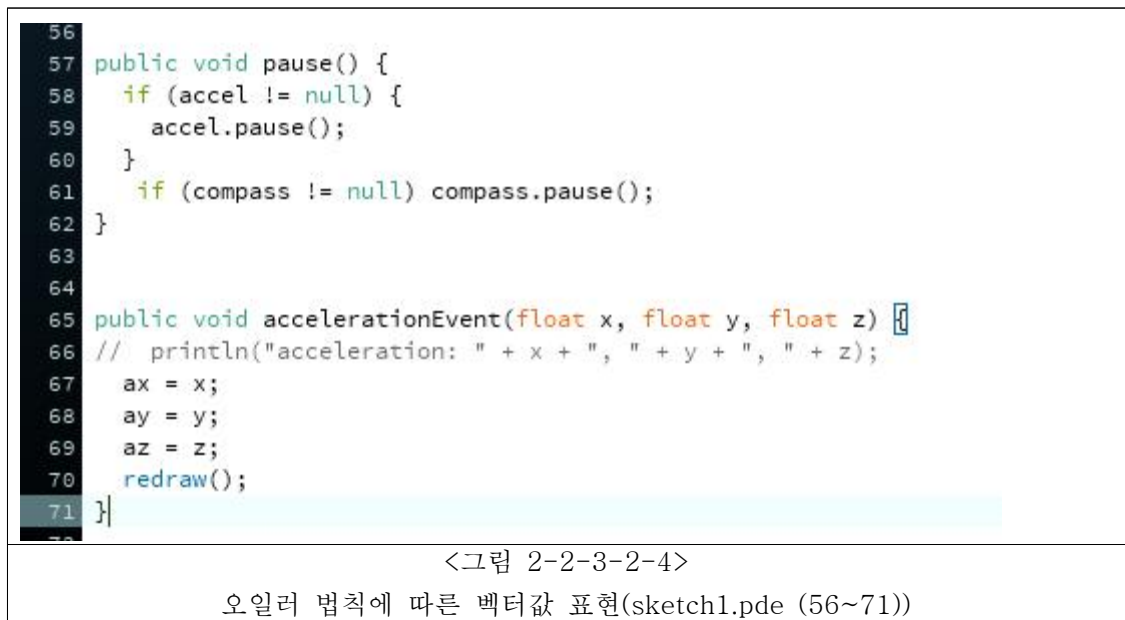
```

3 AccelerometerManager accel;
4 CompassManager compass;
5
6 float direction;
7 float ax, ay, az;
8 float gx, gy, gz;
9
10
11
12 |
13 void setup() {
14
15     accel = new AccelerometerManager(this);
16     orientation(PORTRAIT);
17     compass = new CompassManager(this);
18     size(400, 400, P3D);
19
20 }
21
22 void draw()
23 {
24     background(255);
25     translate(width/2,height/2);
26     line(0,0,0,200,0,0);
27     line(0,0,0,0,200,0);
28     line(0,0,0,0,0,200);
29     // line(0,0,0,-200*gx, 200*gy , 200*direction );
30     line(0,0,0,(sin(gy)+cos(direction))*100,
31         (sin(gx)+cos(gy))*100,
32         (cos(direction)+sin(direction))*100);
33
34     //camera(100,100,100,0,0,0,0,0,0);
35     /* x=radians(random(90));
36     y=radians(random(90));
37     z=radians(random(90));*/
38
39     gx = atan (ax/(sqrt(ay *ay+az*az)));
40     gy = atan (ay/(sqrt(ax*ax+az*az)));
41
42 }
43
44 void directionEvent(float newDirection) {
45     direction = newDirection;
46 }
47
48
49 public void resume() {
50     if (accel != null) {
51         accel.resume();
52     }
53
54     if (compass != null) compass.resume();
55 }

```

<그림 2-2-3-2-3>

오일러 법칙에 따른 벡터값 표현(sketch1.pde (1~55))



3~4. 뒤에서 후술할 AccelerationManager.java와 CompassManager.java를 불러온다. import는 불러온다는 명령어이다.

6~8. 변수들을 선언하는데, 모든 변수는 오차를 최소화하기 위해 실수값 float로 선언한다.

변수명	변수형	용도
direction	float(실수형)	$\psi$ 값 측정
ax(accelration.x)	float(실수형)	x축 가속도 측정
ay(accelration.y)	float(실수형)	y축 가속도 측정
az(axxelration.z)	float(실수형)	z축 가속도 측정
gx(gyro.x)	float(실수형)	$\phi$ 값 측정
gy(gyro.y)	float(실수형)	$\theta$ 값 측정
gz(gyro.z)	float(실수형)	측정 불가 <sup>1)</sup>

<그림 2-2-3-2-5>  
<그림 2-2-3-2-3>, <그림 2-2-3-2-4>의 변수값

15~17. AccelerationManager.java와 CompassManager.java를 사용할 준비를 한다.

18. 픽워지는 결과물은 가로로 400픽셀, 세로로 400픽셀이며, 3개의 축을 사용한다.

24. 배경색은 RGB코드로 255인 하얀색이다.

25~28. 원점을 화면 중앙으로 바꾼다. 바꾸지 않으면 왼쪽 상단부가 원점이 되는데, 굳이 바꿀 필요는 없지만 프로그래머의 편의를 위해 주로 화면 중앙으로 바꾸게 된다. 이 때, 표시되는 디스플레이는 2차원이기 때문에 z축까지 원점을 지정해주지는 않는다. 26번째 줄이 x축, 27번째 줄이 y축, 28번째 줄이 z축이다. line명령어는 line(x1,y1,z1,x2,y2,z2);로 (x1,y1,z2)이라는 점과 (x2,y2,z2)라는 점을 잇는 직선을 만든다.

1) 컴패스 센서 direction 값에서 보정 받음

30~32. 위에서 설명한 오일러 법칙에 따른 벡터값 추론에 따라 공식을 작성한다.

39~40. 가속도 센서로  $\psi, \theta, \phi$  값을 계산한다.

그 외의 함수들은 센서계산값에 오차가 누적되는 것을 막기 위해 만들어진 필터들이다.

<그림 2-2-3-2-6> 실행화면	<그림 2-2-3-2-7> 실행화면 QR코드
------------------------	-----------------------------

보이는 것과 같이, 사용자가 기울이는 ypr값에 따라 벡터를 자동으로 계산할 수는 없으므로 다음과 같은 공식들을 세워서 움직인다. 현재 사용자는 z축 방향을 시선으로 하며 xy평면을 바라보고 있음을 가정한다.

### 2.2.3.3. 중력 프로그램

<pre>1 2 float ypos; 3 4 void setup() 5 { 6   size(400,400,P2D); 7   noStroke(); 8   smooth(); 9   ypos=height/2+10; 10 } 11 12 void draw() 13 { 14   background(0); 15   ypos+=(-height/2+ypos)*0.13; 16   ellipse(200,ypos,100,100); 17 }</pre>	<그림 2-2-3-3-1> 중력 소프트웨어
---	----------------------------

2. 중력은 y축 방향으로 작용한다. 그러므로 y축만 변수로 잡아두고 x축은 생각하지 않아도 좋다.

6. 창은 가로 400픽셀, 세로 400픽셀, 2차원 평면으로 구성되어 있다.

9. 밑으로 떨어질수록 가속도가 일정하므로 속도는 증가하여만 한다. 그러나 프로세싱에서는 속도라는 개념이 없다. 한 프레임 (30헤르츠)에 얼마나 많이 위치가 변하는가가 속도가 된다. 그러므로 속도적인 개념은 결국 좌표값인 ypos가 된다.

15. 이 프로그램에서 가장 중요한 부분이다. 소프트웨어적 코드가 쓰였기 때문에 알아보기 힘들 수 있지만, +=연산자는 실제 있던 값에 누적해서 더한다는 뜻이다. 저 코드를 풀어 전개해서 보면, 저 연산식의 중간에 ypos가 없다면 결국 공은 창의 높이와 상수(0.13)의 곱의 비율

에 따라 등속운동하게 된다. 그러나 현재 위치인 ypos가 들어가기 때문에 등가속도 운동을 할 수 있다. 즉, 현재 떨어진 정도가 크다면 ypos의 값이 커지게 되는데 그러면 그 다음 프레임에서 증가하는 속도에 매개된다. 즉 가속운동하게 되는 것이다.

	
<p>&lt;그림 2-2-3-3-2&gt; 실행화면</p>	<p>&lt;그림 2-2-3-3-3&gt; QR코드</p>

#### 2.2.3.4. 외재적 관점의 가상현실-움직이는 큐브

외재적 관점에서는 우리가 눈앞에 있는 큐브를 우리의 시야에 따라 돌려보는 프로그램이다. 가웃각과 끄덕각은 가속도 센서에서 받아온 x, y, z축의 가속도 값을 가공하여 썼고 도리각은 Compass 센서의 초기 값과 비교한 변화 값을 받아왔다. 헤드기어 착용자의 위치와 시선 방향을 제어하는 camera명령어를 이용하여 시선의 시작점을 큐브 내부로 옮겼다.

	
<p>&lt;그림 2-2-3-2-6&gt; 실행화면</p>	<p>&lt;그림 2-2-3-2-7&gt; 실행화면 QR코드</p>

### 2.2.3.5. 내재적 관점의 가상현실-움직이는 방

```
1 AccelerometerManager accel;  
2 CompassManager compass;  
3 float direction;  
4 float ax, ay, az;  
5 float gx, gy, gz;  
6  
7 void setup()  
8 {  
9     accel = new AccelerometerManager(this);  
10    compass = new CompassManager(this);  
11    orientation(PORTRAIT);  
12    size(640, 360, P3D);  
13 }  
14  
15 void draw() {  
16     translate(width / 2.0, height / 2.0, -100);  
17     stroke (255,100);  
18     /*rotateX(ax );  
19     rotateY(ay);  
20     rotateZ(az );*/  
21     //frameRate (5);  
22     //textSize(32);  
23     gx = atan (-ax/(sqrt(ay *ay+az*az)));  
24     gy = atan (-ay/(sqrt(ax*ax+az*az)));
```

<그림 2-2-3-5-1>

내재적 관점의 가상현실-움직이는 방(1~24)

```

30 camera(0,0,0,0,1,0,0,0,1);
31 rotateX (-gy);
32 rotateZ(-direction);
33 //rotateY(gx*0.5);
34
35
36 box(200);
37
38 pointLight (0,0,0,0,0,0);
39 fill(35);
98 void directionEvent(float newDirection) {
99     direction = newDirection;
100 }
101 public void resume() {
102     if (accel != null) {
103         accel.resume();
104     }
105     if(compass!=null){compass.resume();}
106 }
107
108 public void pause() {
109     if (accel != null) {
110         accel.pause();
111     }
112     if (compass!=null){compass.pause();}
113 }
114
115
116 public void accelerationEvent(float x, float y, float z) {
117     // println("acceleration: " + x + ", " + y + ", " + z);
118     ax = x;
119     ay = y;
120     az = z;
121     redraw();
122 }

```

(생략된 부분은 주석임을 밝힘)

<그림 2-2-3-5-2>

내재적 관점의 가상현실-움직이는 방(30~122)

1~5. 위의 프로그램과 같다. 안드로이드의 센서를 이용함을 밝힌다.

24번째 줄 까지 필요한 값들을 모두 계산하여 변수에 적용한다. 사용된 변수 역시 위의 프로그램과 같다.

30. 시선을 고정한다. 이 부분부터 중요한 부분이다. 시선과 위치를 고정시키고 방을 돌려도 사람들은 자신이 돌고 있다고 생각한다. 모두 상대적인 움직임이기 때문이다.

31. 라디안값을 가지고 있는 gy방향으로 rotateX한다. 이게 시선을 돌려 위를 바라보거나 밑을 바라보는 프로그램이다. 마이너스 값을 붙인 이유는 양의 값을 가질 때가 외재적 관점이고, 내재적 관점은 그 반대가 되어야 한다.

32. 라디안값을 가지고 있는 direction방향으로 rotateZ한다. 이게 회전하면서 방을 둘러보는 프로그램이다. 마이너스 값을 붙인 이유는 양의 값을 가질 때가 외재적 관점인데 내재적 관점은 그 반대가 되어야 하기 때문이다.

	
<p>&lt;그림 2-2-3-5-3&gt; 실행화면</p>	<p>&lt;그림 2-2-3-5-4&gt; QR코드</p>

## 2.2.4. 3세대 로봇

### 2.2.4.1. 내재적 관점과 외재적 관점의 혼합

앞의 내재적 관점과 외재적 관점을 하나의 프로그램에 담은 프로그램이다. 즉, 정육면체의 방안에 정육면체를 바라보는 상황이다. 이 프로그램은 위의 두 프로그램을 결합했을 뿐 아니라 한 가지 보완점이 있다. 바로 이동이 가능하다는 것이다.



```

8 void setup()
9 {
10   accel = new AccelerometerManager(this);
11   compass = new CompassManager(this);
12   orientation(PORTRAIT);
13   size(640, 360, P3D);
14   d0 = direction;
15   camera(800,800,0,0,0,0,0,0,1);
16
17 }
18
19 void draw() {
20   translate(width / 2.0, height / 2.0, -100);
21   stroke (255,100);
22   background(0);
23   /*rotateX(ax );
24   rotateY(ay);
25   rotateZ(az );*/
26   //frameRate (5);
27   //textSize(32);
28   dn = direction - d0 ;
29
30   gx = atan (-ax/(sqrt(ay *ay+az*az)));
31   gy = atan (-ay/(sqrt(ax*ax+az*az)));
32   sum =sqrt(ax*ax+ay*ay+az*az);
33   if(sum <6.9|sum>12.6)
34   {
35     px = - 15*cos(direction) + px;
36     py = - 15*sin(direction)+py;
37     frameRate(10);
38   }
39
40   camera(px ,py ,0,0,1,0,0,0,1);
41   rotateX (-gy);
42
43   rotateZ(-gx);
44   rotateY(-direction);
45   //rotateY(gx*0.5);
46
47
48   box(200);
49   box(2000);
50
51   pointLight (0,0,0,0,0,0);
52   fill(35);
53
54
55 void directionEvent(float newDirection) {
56   direction = newDirection;

```

(생략된 부분은 그림 <2-2-3-5-1>과 같음을 밝힘)

<그림 2-2-4-1-1>

내재적 관점과 외재적 관점의 혼합(8~56)



29-30 중력가속도 값을 각도로 나타내는 식이다.

31 앞에서 재차 언급했듯이 사람이 된다면 사람이 전체적으로 받는 가속도는 중력 가속도 외에 다른 가속도가 생기게 된다. 이 가속도의 합의 변화를 통해 우리는 인간이 뛰고 있는지 아닌지를 측정하게 하였다. 평균적으로 인간이 된다면 전체 가속도가 최저 6.대에서 최대 10.대까지 올라가므로 우리는 이러한 점을 이용하여서 인간이 고개를 기울인 것과 인간 전체가 뛰고있는것을 구분했다.

34-35 그렇다면 이제 뛰고있는 인간이 이동할 방향을 결정할 차례이다. 우선적으로 이 프로그램에서 인간은 자기가 바라보고 있는 시선방향으로 움직인다는 전제를 깔았다. 옆을 보면서 앞으로 이동하는 것은 불가능하다. 인간은 날 수 없으므로 프로그램이 시작할 때의 방위각을 기준으로 방위각의 변화한 정도를  $dn$ 에 저장하였다. 이  $dn$ 에 따라서 착용자의 시점을 지정해주는 명령어인 `camera(xposition, yposition, zposition, xheading position, y heading position, z heading position, (법선 벡터))`의 첫 세 변수를  $(15\cos(dn), 15 \sin(dn), 0)$  조절하였다. 그래서 시선처리와 이동이 동시에 가능하도록 하였다.

	
<p>&lt;그림 2-2-4-1-2&gt; 실행화면</p>	<p>&lt;그림 2-2-4-1-3&gt; 실행화면 QR코드</p>

## 2.2.4.2. 마인크래프트의 제어

### 2.2.4.2.1. EV3로 제어

사람이 걷는다는 것을 어떻게 인식할까? 사람이 걸을 때의 가장 큰 요소는 발바닥이 땅에서 떨어졌다가 붙는다는 것이다. 즉, 발과 땅의 거리가 지속적으로 달라진다. 그러므로 발바닥과 땅 사이의 거리를 측정할 때 그 값이 계속해서 바뀐다면 현재 사람이 걷고 있다고 추론할 수 있다. 그러나 프로그래밍에서 값이 계속해서 변하는지 변하지 않는지를 검사하는 것은 값이 특정 기준을 넘는지 검사하는 것보다 훨씬 어렵다. 이러한 문제점을 해결하기 위해 우리 싸컨 대사건은 계속해서 초음파센서의 값을 변수에 저장하고 현재의 초음파센서값과 비교하여 평균 변화율을 측정하였다.

gyro the original second.c | gyro the second.c | gyro.c | punching.....c | **minecraft.c\***

```

1  int temp, temp2;
2  int go, motorgo;
3
4  task gogo()
5  {
6      while(1)
7      {
8          if(go==1)
9          {
10             motor[motorA]=5;
11         }
12         if(go==0)
13         {
14             motor[motorA]=-5;
15         }
16     }
17 }
18
19
20 task tempp()
21 {
22     while(1)
23     {
24         temp=SensorValue[S1];
25         wait1Msec(100);
26         temp2=sensorValue[S1];
27         wait1Msec(100);
28         displaytextline(8, "%d", temp2 - temp);
29         displaytextline(6, "%d", motorgo);
30     }
31 }

```

gyro the original second.c | gyro the second.c | gyro.c | punching.....c | **minecraft.c\***

```

32
33 task main()
34 {
35     starttask(gogo);
36     starttask(tempp);
37     resetMotorEncoder(motorA);
38     resetMotorEncoder(motorB);
39     while(1)
40     {
41         if(abs(temp2 - temp) > 5)
42         {
43             go=1;
44             wait1Msec(1000);
45             displayCenteredBigTextLine(2, "go")
46         }
47         else
48         {
49             go=0;
50             displayCenteredBigTextLine(2, "stop");
51         }
52     }
53 }
54
55
56
57
58 stopalltasks();
59 }
60

```

<그림 2-2-4-2-1-1>

소스코드

대략적으로 설명하자면, temp와 temp2에 계속해서 초음파센서값을 저장하고 그 값이 심하게 요동치면 서보모터를 이용하여 직접 키보드에 있는 버튼을 누르는 식이다. 초기 RobotC로 작성하였다.

	
<p>&lt;그림 2-2-4-2-1-2&gt; 실행 화면</p>	<p>&lt;그림 2-2-4-2-1-3&gt; QR코드</p>

#### 2.2.4.2.2. 아두이노로 제어

1세대 로봇에서는 3차원 객체가 2차원 디스플레이에 표현될 수 있도록 여러 수학적 계산을 하면서 3차원의 센서값을 받아들이는 연구를 하였다. 그리고 2세대 로봇에서는 더 나은 그래픽을 구현하면서 더 많은 시도를 하였고, 결국 가상현실 인터페이스 중에서 외재적 요인과 내재적 요인에 대해 탐구하게 되며 이 연구결과를 토대로 직접 가상현실을 구현할 수 있게 되었다. 결국에 가상현실이라는 것은 우리가 연구했던 내재적 가상현실과 외재적 가상현실, 그리고 실제의 물리적 힘이 다양한 형태로 결합된 것이기 때문이다. 그러나 예술적이고 디자인적인 그래픽 요소를 우리가 구현하기에는 무리가 있기에 적합한 그래픽 플랫폼을 활용해야 했다.

프로세싱은 자바로 구동되고, 자바의 import명령어를 이용하면 windows의 몇몇 기능을 제어할 수 있다. 원래는 아두이노 우노 보드에서는 사용할 수 없는 기능이지만 프로세싱에서 자바 문법을 사용하면 그다지 어려운 일은 아니다.

```

1 import java.awt.*; //instead of just importing all the random java.awt.bl
2 import java.awt.event.*;
3 import java.awt.event.KeyEvent;
4 import java.awt.Robot;
5 import java.awt.AWTException;
6 import processing.serial.*;
7 import processing.opengl.*;
8 import toxi.geom.*;
9 import toxi.processing.*;
10
11 int absMouseX, absMouseY; //place holder variables for the mouseX and mou
12 float IsRunning;
13 float ArduinoIsx, ArduinoIsy;
14 float TH, TH2;
15 int ii=0;|
16 Robot robot;
17 float AccelX;
18 float AccelY;
19 float AccelZ;
20 float ActualAccelX;
21 float ActualAccelY;
22 float ActualAccelZ;
23 float kk;
24
25 float[] accel = new float[3];
26 ToxiclibsSupport gfx;
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42 void setup() {
43     size(100, 100); //setup the window size
44     try { //try to make a new robot, REQUIRES an exception handling catch.
45         robot = new Robot();
46     }
47     catch (AWTException e) { //catch that shit
48         e.printStackTrace();
49         exit();
50     }
51     gfx = new ToxiclibsSupport(this);
52     println(Serial.list());
53
54     // get the first |available port (use EITHER this OR the specific port code

```

<그림 2-2-4-2-1>

마인크래프트 제어 소프트웨어(1~54)

1~9. 필요한 자바 기능들을 불러온다. 여기서는 주로 아두이노의 기능과 마우스, 키보드 컨트롤의 기능이 쓰였다.

11~39. 필요한 변수들을 선언한다.

변수명	형태	기능
absMouseX, absMouseY	int	현재 마우스의 픽셀값을 프로그램과 자바가 인식하고 쓰기 위한 값으로 저장한다.
ArduinoIsx, ArduinoIsy	float	라디안값으로 중심을 축으로 yaw값과 Pitch값을 측정한다. yaw가 x, pitch가 y이다.
TH, TH2	float	ArduinoIsx, ArduinoIsy와 현재 컴퓨터의 차이를 측정함으로서 현재 컴퓨터가 해야 할 일이 무엇인지 판단한다.
Accel[2]	float	teapotPacket[17]의 14~17에서 값을 받아온다. Accel[2]를 보행 여부 측정에 이용한다.
teapotPacket [16]	char	시리얼통신에는 char가 이용된다. 이것은 아두이노에서 센서의 원시값을 받아오고, 프로세싱에서는 이를 분해해서 사용한다.
q[3]	float	teapotPacket[17]을 분해한 1차 원시값이다.
ypr[2]	float	yaw, pitch, roll값이다. ypr[0]=yaw, ypr[1]=pitch, ypr[2]=roll값이다.
<p style="text-align: center;">&lt;표 2-2-4-2-2&gt; 변수값들의 의미</p>		

43. 제어기에 불과하므로 사이즈는 작을수록 좋다.

44~50. 지금부터 자신의 마우스와 키보드 제어권을 프로세싱에 소속된 자바 프로토콜에 넘겨준다.

51~65. 시리얼통신을 시작한다. 기계어로 되어 있는 아스키코드로 전송하기 때문에 사용자는 읽을 수 없는 값이다. 65번줄의 'r'이라는 값으로 시작한다.

79. 현재 로봇이 무슨 일을 해야 하는지 계속 업데이트시켜줘야 한다. 받아온다.

83~87. 최초 1회 마우스를 화면의 중앙으로 옮긴다. 이 컴퓨터를 실행할 때 사용자의 컴퓨터 모니터 픽셀 규격은 1344\*668이었기 때문이다. 원래는 screen.이라는 명령어로 제어할 수 있지만 어째서인지 우리가 실행할 때에는 작동하지 않았다.

99~100. ArduinoIsx, ArduinoIsy는 라디안값이므로 도값으로 변경해줘야지 int(정수)값이 되어 픽셀값으로 지정해주기 편하다.

103~110. 만약에 아두이노가 움직이면 마우스도 따라서 움직일 것이다. ArduinoIsx에 3.7을 곱한 것은 1344를 360으로 나누면 3.7에 근사하고, 668을 360으로 나누면 2에 근사하기 때문이다. 비율에 관한 문제이기 때문에 정확히 맞을 필요는 없지만, 근삿값의 오차 때문에 사용자가 한바퀴 돌아도 360도를 정확히 돌기 힘들 수는 있다.

111~119. accel[2]값은 z축 중력가속도를 측정하는 값으로서, 방향전환을 할 때에도 값의 변화는 있지만 10000은 임계값으로서 넘기지 않는다. 걷는다라는 z축방향으로 요동이 심한 활동이 있을 때에만 10000을 넘는다.

112~147. teapotPacket[17]의 모든 아스키코드값을 분해한다.

```

58     String portName = "COM6";
59
60     // open the serial port
61     port = new Serial(this, portName, 115200);
62
63     // send single character to trigger DMP init/start
64     // (expected by MPU6050_DMP6 example Arduino sketch)
65     port.write('r');
66
67 }
68
69 void draw() {
70     updateRobot(); //update the robot, see the robot function for more info
71     // cameraaa();
72     // robot.mouseMove(absMouseX+second(), absMouseX+second());
73
74     if (millis() - interval > 1000) {
75         // resend single character to trigger DMP init/start
76         // in case the MPU is halted/reset while applet is running
77         port.write('r');
78         interval = millis();
79     }
80
81     // black background
82     background(0);
83     if(ii<1)
84     {
85         robot.mouseMove(683,384);
86         ii++;
87     }
88
89     translate(width/2,height/2);
90     AccelX=accel[0]/8192;
91     AccelY=accel[1]/8192;
92     AccelZ=accel[2]/8192;
93
94     ActualAccelX=AccelX-gravity[0];
95     ActualAccelY=AccelY-gravity[1];
96     ActualAccelZ=AccelZ-gravity[2];
97
98     IsRunning=sqrt(ypr[0]*ypr[0]+ypr[1]*ypr[1]+ypr[2]*ypr[2]);
99     ArduinoIsx=ypr[0]*180/PI;
100    ArduinoIsy=ypr[1]*180/PI;
101    TH=ArduinoIsx-683;
102    TH2=ArduinoIsy-384;
103    if(abs(TH)>0.1 || abs(TH2)>0.1)
104    {
105        robot.mouseMove(round(683+ArduinoIsx*3.7),round(384+ArduinoIsy*2));
106        if(accel[2]<10000)
107        {
108            robot.keyRelease(KeyEvent.VK_W);
109        }
110    }
111    if(accel[2]>10000)
112    {
113        robot.keyPress(KeyEvent.VK_W);

```

<그림 2-2-4-2-2>

마인크래프트 제어 소프트웨어(58~113)

58~65. 포트는 COM6포트를 이용하고, 전송하는 데이터는 주로 char값인 아스키 코드값을 이용한 뒤 이를 가공하여 사용한다. 가공은 MPU6050의 드라이버를 참고하고, 그 외의 구동적 소스코드를 모두 직접 작성했다.



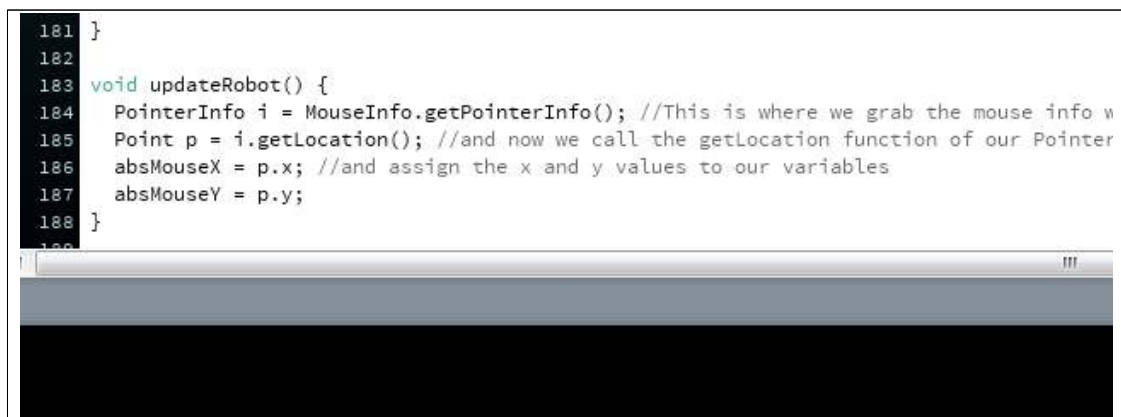
```

114 }
115 if(accel[2]<100000)
116 {
117     robot.keyRelease(KeyEvent.VK_W);
118 }
119
120 float[] axis = quat.toAxisAngle();
121 }
122 void serialEvent(Serial port) {
123     interval = millis();
124     while (port.available() > 0) {
125         int ch = port.read();
126         print((char)ch);
127         if (ch == '$') {serialCount = 0;} // this will help wit
128         if (aligned < 4) {
129             // make sure we are properly aligned on a 14-byte p
130             if (serialCount == 0) {
131                 if (ch == '$') aligned++; else aligned = 0;
132             } else if (serialCount == 1) {
133                 if (ch == 2) aligned++; else aligned = 0;
134             } else if (serialCount == 12) {
135                 if (ch == '\r') aligned++; else aligned = 0;
136             } else if (serialCount == 13) {
137                 if (ch == '\n') aligned++; else aligned = 0;
138             }
139             //println(ch + " " + aligned + " " + serialCount);
140             serialCount++;
141             if (serialCount == 17) serialCount = 0;
142         } else {
143             if (serialCount > 0 || ch == '$') {
144                 teapotPacket[serialCount++] = (char)ch;
145                 if (serialCount == 17) {
146                     serialCount = 0; // restart packet byte position
147
148                     // get quaternion from data packet
149                     q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
150                     q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
151                     q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
152                     q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
153
154                     accel[0] = ((teapotPacket[10] << 8) | teapotPacket[11]);
155                     accel[1] = ((teapotPacket[12] << 8) | teapotPacket[13]);
156                     accel[2] = ((teapotPacket[14] << 8) | teapotPacket[15]);
157                     for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];
158                     for (int j = 0; j < 3; j++) if (accel[j] >= 2) accel[j] = -4 + accel[j];
159
160                     // set our toxilibs quaternion to new data
161                     quat.set(q[0], q[1], q[2], q[3]);
162
163                     gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
164                     gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
165                     gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];
166
167                     // calculate Euler angles
168                     euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
169                     euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
170                     euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1], 2*q[0]*q[0] + 2*q[3]*q[3] - 1);
171
172                     // calculate yaw/pitch/roll angles
173                     ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
174                     ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] + gravity[2]*gravity[2]));
175                     ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] + gravity[2]*gravity[2]));
176
177                 }
178             }
179         }
180     }

```

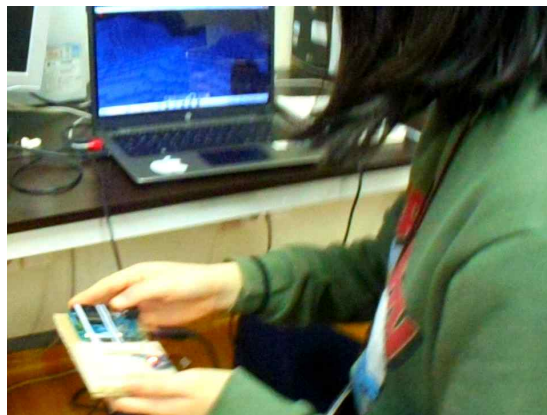
<그림 2-2-4-2-3>

마인크래프트 제어 소프트웨어(



<그림 2-2-4-2-4>

마인크래프트 제어 소프트웨어



<그림 2-2-4-2-2>

실행화면



<QR코드 2-2-4-2-3>

실행화면

## 3. 결론

### 3.1. 결론

가상현실은 지금까지 인류가 만들어낸 예술, 과학, 수학 등 많은 학문 갈래의 발전을 모두 총 집합한 기술체라고 볼 수 있다. 우리는 그중 디스플레이와 사용자의 시선 움직임을 연동시키는 방법에 대해 아주 기초적인 부분부터 응용되어 있는 부분까지 그 원리를 탐구해보았다. 처음의 EV3모형을 사용하여 3차원 좌표계를 2차원 디스플레이로 옮기는 방향에 대해 탐구해보았고 이를 이용한 프로세싱과 아두이노를 활용하여서 인간의 시선 처리를 디스플레이에서 어떻게 표현할 것인지에 대한 좀더 수준 높은 가상현실에 대해 연구하였다. 마지막으로 우리가 가지고 있는 시선처리와 이동에 관한 알고리즘을 종합하여서 내재적 관점과 외재적 관점을 합한 프로그램과 다른 예술적 영역과 결합되었을 때를 보여주기 위해 마인크래프트 게임과의

결합에서 이 기술이 얼마나 더 활용될 수 있을지에 대한 전망을 보여주었다.

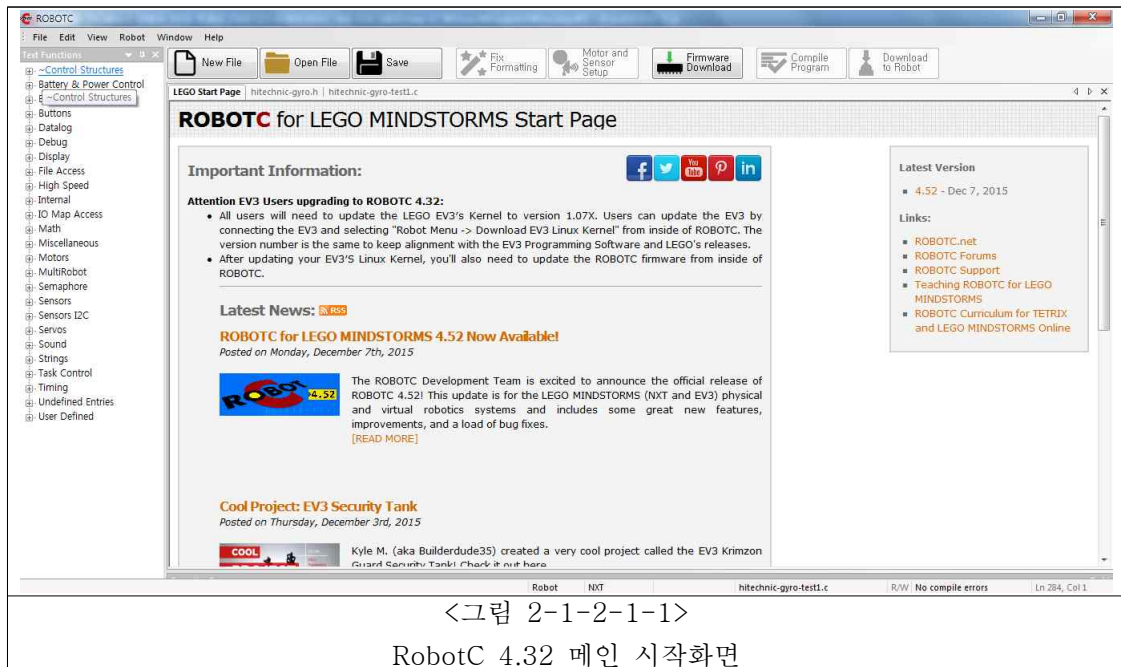
## 4. 참고 문헌

## 5. 부록

드라이버 프로그램을 제외하고, 이 연구에 쓰인 모든 프로그램은 싸컨대사건이 직접 타이핑하여 알고리즘을 생각하고 100% 창작한 프로그램임을 밝힘.

### 5.1.1. 프로그램의 소개

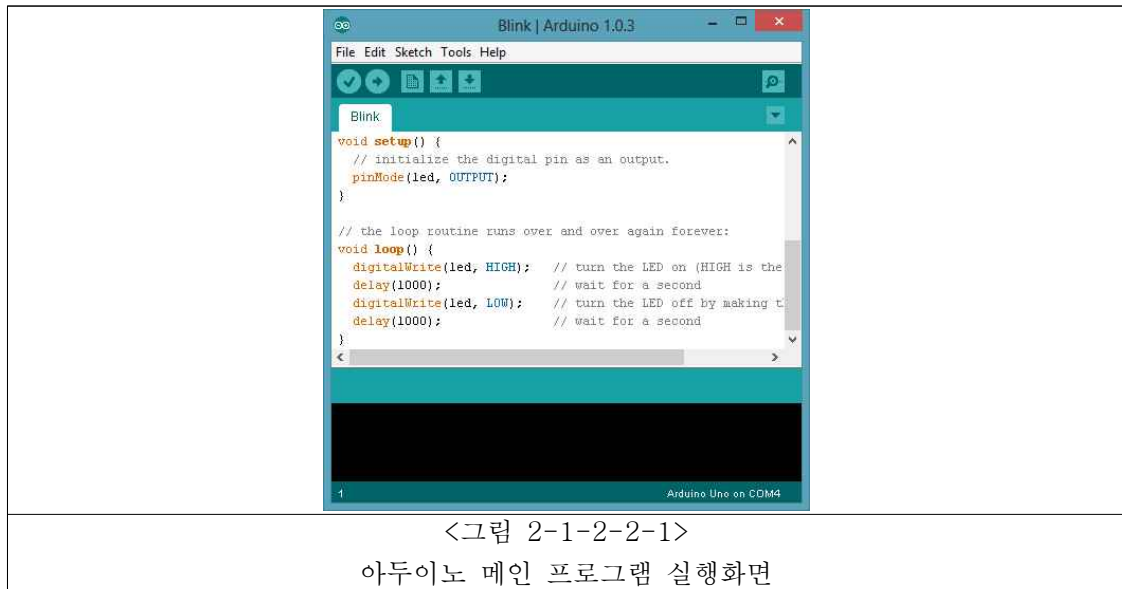
#### 5.1.1.1. RobotC(3.84 ~ 4.32)



RobotC는 카네기멜론대학 로보틱스 아카데미에서 2008년에 개발한 프로그램으로, 제어기로 이 프로그램을 선택한 이유는 다음과 같다.

1. 컴파일-수정 과정이 쉽다.
2. 변수 선언이 간단하다.
3. 2차원배열을 지원한다.
4. 용량이 적다. 이 연구에서 사용된 가장 큰 용량의 프로그램도 3KB이다.
5. HiTECH NXT Gyro Sensor를 지원한다.
6. C언어와 유사한 문법을 가지고 있기 때문에 코딩에 용이하다.

#### 5.1.1.2. -Arduino main Program




아두이노는 .ino형식의 파일로 움직이는 소프트웨어를 지원하며, task를 비교적 자유롭게 쓸 수 있는 RobotC와 달리 한정된 CPU를 가지기 때문에 두 개의 void함수를 사용할 수 있다. 하나는 회로구조상 무한히 반복되는 void loop(){}, 그리고 실행시 초기 한 번만 수행하여 주로 초깃값 설정 등에 이용되는 void setup(){}이 그것이다. 이 연구에서는 아두이노 프로그램 자체가 쓰이기보다는 프로세싱 프로그램을 호출하는 용도로 많이 사용되었다.

### 5.1.1.3. 프로세싱

프로세싱은 아두이노 기반 사용자가 아두이노를 입력장치로 하여 텍스트기반의 그래픽을 좀 더 쉽게 작성하기 위해 만들어진 프로그램이다. 물론 그래픽 외에도 간단한 연산이나 연타 다른 언어들이 할 수 있는 많은 수행을 할 수 있으나 그래픽 방면이 특화되었다. 많은 센서 예제프로그램이 이 프로그램으로 진행된다.

프로세싱의 기반 프로그램은 JAVA 로 운영되며, 모든 문법은 C 객체지향 문법을 기본으로 한다. 그렇기 때문에 컴퓨터에 JAVA 6 이상이 필수적으로 설치되어 있어야 한다. 3차원 가상 현실을 자바 가상머신에서 구현할 수 있다는 것이 특징이다.

프로세싱은 현재 산업디자인등에서 넓게 쓰이고 있고, 포토샵처럼 한정된 작업만 할 수 있는 게 아니라 직접 프로그래밍하는 프로그램 언어로서 C문법 기반 하에 아두이노의 일부 문법과 인터페이스를 차용해서 나왔다. 주로 아두이노 우노 보드와 연계되며, 연계를 위해 아두이노 프로그램에서 프로세싱을 호출해 센서값을 전송할 수 있다.

 <p>Processing 3.0.1</p> <p>An open project initiated by Ben Fry and Casey Reas. Supported by programmers like you and the nonprofit Processing Foundation, 501(c)(3).</p> <p>© 2012–2015 The Processing Foundation © 2004–2012 Ben Fry and Casey Reas © 2001–2004 Massachusetts Institute of Technology</p>	 <p>AT&amp;T 2:10 PM</p> <p>Sparks</p> <p>sketch Spark</p> <pre>import android.os.Vibrator; ArrayList&lt;Spark&gt; sparks;  void setup() {   size(displayWidth, displayHeight, OPENGL);    sparks = new ArrayList&lt;Spark&gt;(); }</pre> <p>Dexing compiled files...</p> <p>The import java.io.PrintWriter is never used</p> <p>8. WARNING in /storage/emulated/0/build/src/prod import java.io.InputStream; AAAAAAAAAAAAAAAAAAAA</p> <p>The import java.io.InputStream is never used</p> <p>9. WARNING in /storage/emulated/0/build/src/prod import java.io.OutputStream; AAAAAAAAAAAAAAAAAAAA</p> <p>The import java.io.OutputStream is never used</p> <p>10. WARNING in /storage/emulated/0/build/src/prod import java.io.IOException; AAAAAAAAAAAAAAAAAAAA</p> <p>The import java.io.IOException is never used</p> <p>10 problems (10 warnings) ECJ compilation successful Running DX...</p> <p>trouble writing output: already prepared</p>
<p>&lt;그림 2-1-2-3-1&gt; 프로세싱 3 실행화면</p>	<p>&lt;그림 2-1-2-3-2&gt; 안드로이드용 프로세싱 2 APDE 실행화면</p>

#### 5.1.1.4. I2C 통신

I2C 통신은 Inter Integrated Circuit의 약자로서, 마이크로프로세서와 저속 주변장치 사이의 통신을 용도로 Philips에서 개발한 규격이다. 두 통신 라인을 사용하여 TWI(Two Wire Interface)라고 불리기도 한다. I2C는 양방향 오픈드레인 선인 SCL(Serial Clock)과 SDA(Serial Data)로 이루어져 있으며, Master-Slave 형태로 동작한다.

### 5.1.2. 하드웨어의 소개

#### 5.1.2.1. 아두이노 우노 보드


<p>&lt;그림 2-1-3-1-1&gt; 아두이노 우노 보드</p>

아두이노(Arduino)는 오픈 소스를 기반으로 한 단일 보드 마이크로컨트롤러로 완성 된 보드



(상품)와 관련 개발 도구 및 환경을 말한다. 처음에 AVR을 기반으로 만들어졌으며, 아두이노 AVR 계열의 보드가 현재 가장 많이 판매되고 있다. ARM 계열의 Cortex-M0(Arduino M0 Pro)과 Cortex-M3(Arduino Due)를 이용한 제품도 존재한다.

아두이노는 다수의 스위치나 센서로부터 값을 받아들이며, LED나 모터와 같은 외부 전자 장치들을 통제함으로써 환경과 상호작용이 가능한 물건을 만들어 낼 수 있다. 임베디드 시스템 중의 하나로 쉽게 개발할 수 있는 환경을 이용하여, 장치를 제어할 수 있다.

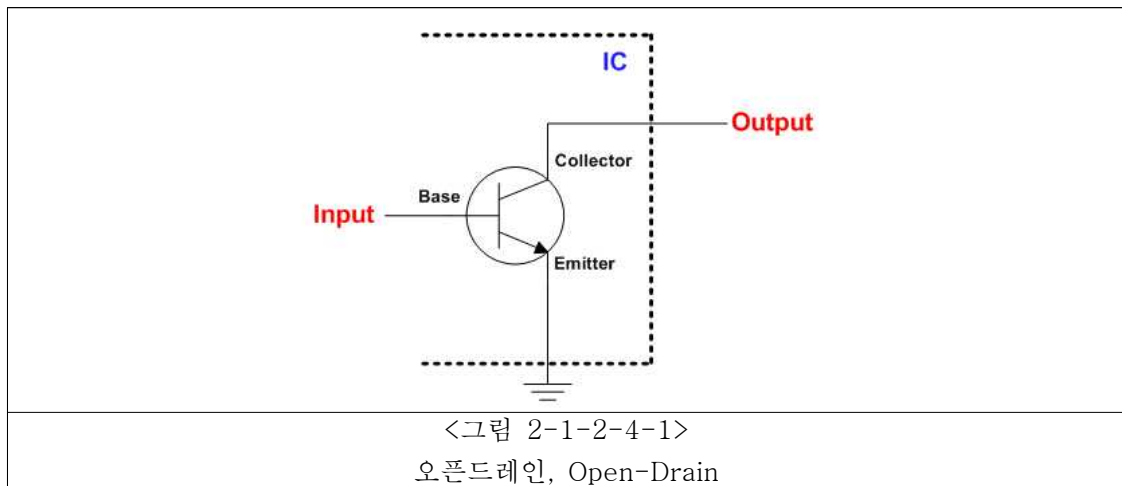
아두이노용 6축 자이로센서는 6가지의 센서값을 가지며 이는 모두 아두이노 프로그램에서 SensorRawValue를 변환해 배열 데이터로 프로세싱에 전송할 수 있다. 6가지의 데이터값은 다음과 같다.

### 5.1.2.2. I2C 통신

I2C 통신은 Inter Integrated Circuit의 약자로서, 마이크로프로세서와 저속 주변장치 사이의 통신을 용도로 Philips에서 개발한 규격이다. 두 통신 라인을 사용하여 TWI(Two Wire Interface)라고 불리기도 한다. I2C는 양방향 오픈드레인 선인 SCL(Serial Clock)과 SDA(Serial Data)로 이루어져 있으며, Master-Slave<sup>2)</sup> 형태로 동작한다. 속도는 다른 방식에 비하여 현저히 느리지만 하드웨어의 구성이 매우 간단하며 대화형 프로토콜<sup>3)</sup>을 만들 수 있다. 또한 하나의 버스에 수많은 <sup>4)</sup>노드를 연결할 수 있다는 장점이 있다.

#### 5.1.2.2.1. 오픈드레인 출력

오픈드레인 출력은 아래 그림과 같이 Transistor의 Collector 핀이 아무것과도 연결되어 있지 않은 통신 회로를 말한다. Input을 Master가 입력하고, Output은 그 값을 받은 Slave들이 출력하는 형태인 Master-Slave 통신 회로이다.



Input에서 High(혹은 True, 1) 신호가 들어오면 Transistor는 ON(회로에 연결)상태가 되고, Output은 GND에 연결되어 Low(혹은 False, 0)를 출력한다.

Input에서 Low 신호가 들어오면 Transistorsms OFF(단락)상태가 되고, 이 회로는 Output

- 2) 두 조정 전원을 접속하여 한 쪽을 Master, 다른 쪽을 Slave로 지정하여 Master가 Slave를 제어할 수 있도록 동작하는 통신 방식을 말한다. - 네이버 지식백과, '마스터슬레이브동작'
- 3) protocol, <컴퓨터> 컴퓨터와 컴퓨터 사이, 또는 한 장치와 다른 장치 사이에서 데이터를 원활히 주고받기 위하여 약속한 여러 가지 규약(規約). 이 규약에는 신호 송신의 순서, 데이터의 표현법, 오류 검출법 따위가 있다. [비슷한 말] 통신 규약. - 네이버 지식백과, '프로토콜'
- 4) node, <컴퓨터> 데이터 통신망에서, 데이터를 전송하는 통로에 접속되는 하나 이상의 기능 단위. 주로 통신망의 분기점이나 단말기의 접속점을 이른다. - 네이버 지식백과, '노드'

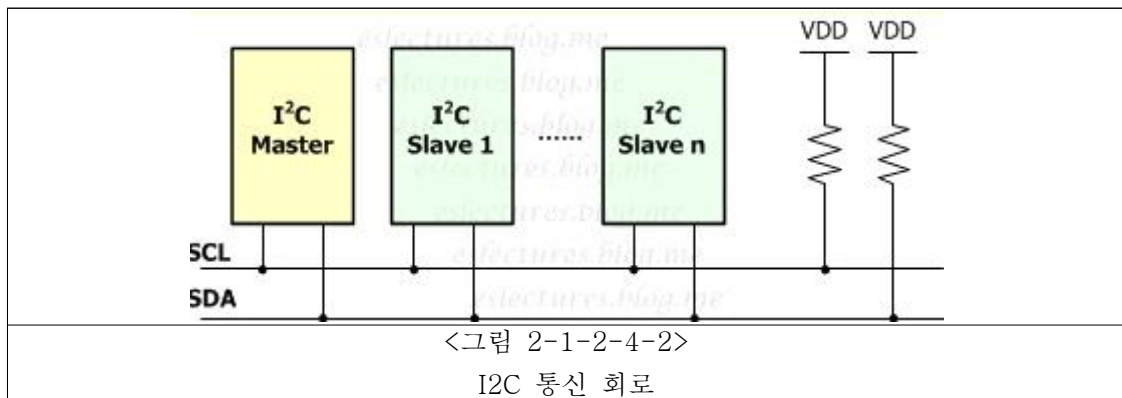
으로 Hi-Impedence(높은 저항)상태가 된다. 이 상태에서는 회로가 단락되어(부동 상태, floating state) Collector의 전압을 정의할 수 없으므로 여기에 Pull-up 저항<sup>5)</sup>을 달아서 출력 전압을 높여 Output이 High가 되도록 한다.

오픈드레인 회로로 여러 장치(Transistor)가 연결되어 있을 경우 각 장치가 내보내는 Output 중 하나라도 Low라면 회로 전체의 논리 상태도 Low가 된다. 이렇게 선(Wire, 회로)만으로 OR상태를 도출하므로 wired-OR라고 부르기도 한다.

즉, 입력이 Low 일 때 Pull-Up 저항 때문에 High 가 되며 Transistor에 의해 연결이 끊어진 상태가 된다. 따라서 여러 개의 Device가 연결되어도 신호의 충돌이 생기지 않는다. 참고로 Open-Collector를 사용하지 않을 경우 여러 개의 Device에서 Low또는 High 신호를 보내주면 단락(short, 합선)이 발생할 수도 있다.

이와 같이 오픈드레인 회로는 여러 장치를 연결하여 양방향 통신을 가능하게 한다(wired-OR). 그리고 다른 전압을 사용하는 장치에도 데이터 전송이 가능하다.

I2C 통신에 사용되는 SDA는 데이터용 선이고 SCL은 통신의 동기를 위한 클럭용 선이다. I2C 통신이 시작되면 Master는 SDA로 Slave Address를 전송한다. 전송 후 <sup>6)</sup>ACK를 기다리는데, 이때 여러 개의 Device 중에 Slave Address에 해당하는 Device가 SDA Line을 Low로 만들면 Master는 해당 Device가 존재한다고 판단하게 되고 통신을 시작한다. 이처럼 통신 전에 Slave Address를 알아내야 하는 이유는 모든 장치들의 SDA와 SCL이 연결되어 있기 때문이다. 이 Slave Address의 길이는 7bit 이며, 따라서 하나의 Master는 최대 128개의 Slave와 연결될 수 있다.



### 5.1.2.3. 시리얼 통신

시리얼은 거의 모든 PC에서 표준으로 사용되는 디바이스 통신 프로토콜이다.

- 5) 디지털 회로에서 논리적으로 High상태를 유지하기 위해서 신호의 입·출력 단자와 전원 사이에 접속하는 저항을 말한다. 입력단 회로인 경우 입력 논리 값을 H로 올바르게 인가하기 위해 사용되며, 출력단 회로의 경우 출력 전류를 증대시키거나 해당 회로가 오픈 드레인 회로인 경우 사용한다. 이와 반대로 Pull-Down 저항은 논리적으로 Low상태를 유지하기 위해서 신호의 입·출력 단자와 전원 사이에 접속하는 저항을 말한다.
- 6) 《데이터 전송》 송신 측에 대하여 수신 측에서 긍정적 응답으로 보내지는 전송 제어용 캐릭터. 정확하게 접속되었다. '데이터가 오차 없이 수신되었다'라는 내용을 전달한다. - 네이버 지식백과, 'ACK [acknowledge]'

시리얼 포트(시리얼 통신에서 사용되는 포트)는 정보의 바이트를 한 번에 한 비트 씩 순차적으로 송수신한다. 한 번에 전체 바이트를 동시에 전달하는 병렬 통신과 비교하면 시리얼 통신은 속도가 느리지만 훨씬 간단하며, 최대 1.2km의 장거리에도 사용할 수 있다.

시리얼 통신은 Tx(송신용), Rx(수신용), GND(그라운드용)의 세 가지 전송 라인을 통해 진행된다. 시리얼 통신을 위해서는 다음과 같은 사항이 잘 갖춰져야 한다.

#### 5.1.2.3.1. 보드레이트

보드레이트(baud rate)는 송신 장치의 초당 비트 전송 숫자로 이해할 수 있다. 즉, 초당 통신 속도를 측정하는 수치이며, 클럭(Clock) 주기라고도 불린다. 두 장치의 보드레이트를 일치시키지 않으면 서로가 명령을 이해할 수 없게 되어 버린다.

#### 5.1.2.3.2. 데이터 비트

데이터 비트는 전송되는 실제 데이터 비트의 측정값을 의미한다. 어떤 정보를 전송하느냐에 따라 어떤 세팅을 선택할 지를 결정해야 한다. 예를 들어, 표준 ASCII는 0 ~ 127 (7 비트)의 값을 가진다. 확장된 ASCII는 0 ~ 255 (8 비트)를 사용한다. 전송하려는 데이터가 단순 텍스트 (표준 ASCII)일 경우, 패킷당 7비트의 데이터를 보내면 통신에 무리가 없다.

패킷은 단일 바이트 전송을 의미하며, 시작/정지 비트, 데이터 비트, 패리티가 포함된다. 실제 비트의 수는 선택된 프로토콜에 따라 달라지므로 모든 경우를 포괄하는 "패킷"이라는 용어를 사용한다.

#### 5.1.2.3.3. 정지 비트

정지 비트는 단일 패킷에 대한 통신의 종료를 알리는 데 사용되며, 일반적인 값은 1, 1.5, 2 비트이다. 데이터는 모든 라인을 통해 클럭되며 각 디바이스에는 고유의 클럭이 있기 때문에 두 개의 디바이스는 동기화가 되지 않을 가능성이 있다. 따라서 정지 비트는 전송의 종료를 알려줄 뿐 아니라 클럭 속도 오류를 방지하기 위한 완충 역할을 한다.

#### 5.1.2.3.4. 아두이노와 프로세싱을 이용한 시리얼 통신

##### 5.1.2.3.4.1. 아두이노

##### 5.1.2.3.4.1.1. 통신 기반 작업 설정



```

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
// #include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// ADO low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// ADO high = 0x69
MPU6050 mpu; //MPU6050 mpu(0x69); // <-- use for ADO high

```

<그림 2-1-2-4-3>

아두이노 시리얼 통신 - 통신 기반 작업 설정

아두이노에서 시리얼 통신을 이용하기 위해서는 I2Cdev.h 와 Wire.h 의 라이브러리 파일이 필요하다. 라이브러리 파일은 특정 작업을 실행하기 위해 프로그램의 외부에서 불러오는 파일이다.

또한 MPU6050의 데이터를 변환해주는 MPU6050\_6Axis\_MotionApps20.h 라이브러리 파일이 필요하다. 그리고 MPU6050 센서의 ADO 포트가 GND에 연결되었을 경우(ADO Low), MPU6050 mpu; 명령어로 센서를 연결하고, VCC(3.3V)에 연결되었을 경우 MPU6050 mpu(0x69); 명령어로 센서를 연결한다.

#### 5.1.2.3.4.2. 센서에서 출력할 데이터 형식 지정

MPU6050의 센서 값을 가공하여 다양한 형태의 값을 통신에 사용할 수 있다. 우리는 아두이노가 가공하고 있는 센서 값이 아니라 프로세싱에서 최종적으로 가공된 개별적인 값을 원하기 때문에 #define OUTPUT\_TEAPOT 명령어만 주석 해제를 하여 사용하였다. (다음장)

```

// #define OUTPUT_READABLE_QUATERNION
// #define OUTPUT_READABLE_EULER
// #define OUTPUT_READABLE_YAWPITCHROLL
// #define OUTPUT_READABLE_REALACCEL
// #define OUTPUT_READABLE_WORLDACCEL

```

위와 같은 명령어들이 있는데, 이들은 사람이 읽을 수 있도록 값들을 묶어서 시리얼 모니터에 보여준다. 인간의 언어 형태로 묶어서 전송된 데이터는 이후에 센서 값 처리가 어렵다.

```
// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
// #define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
// #define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT
```

<그림 2-1-2-4-4>

아두이노 시리얼 통신 - 원하는 센서 출력 데이터 형식 설정

#### 5.1.2.3.4.2.1. 전송할 데이터 패킷 형성

```
// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuintStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorInt16 aa; // [x, y, z] accel sensor measurements
VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements
VectorFloat gravity; // [x, y, z] gravity vector
float euler[3]; // [psi, theta, phi] Euler angle container
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[17] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, 'W', 'r', 'W', 'n' };
```

<그림 2-1-2-4-5>

아두이노 시리얼 통신 - 전송할 데이터 패킷 형성

맨 아랫줄 바로 위(주석 제외)까지는 각 데이터들이 몇 비트인지 지정하고 전송할 데이터의 이름을 정한다. 맨 아랫줄에서는 패킷이 어떻게 구성되어 있는지를 지정한다.

'\$'는 아스키코드로서 데이터 패킷 전송의 시작을 알리고, 'Wr'과 'Wn'는 데이터 패킷 전송의 종료를 알리고 캐리지 리턴<sup>7)</sup>, 줄 바꿈을 해준다.

#### 5.1.2.3.4.2.2. 통신의 시작

```
// =====  
// ===                INITIAL SETUP                ===  
// =====  
  
void setup() {  
    // join I2C bus (I2Cdev library doesn't do this automatically)  
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE  
        Wire.begin();  
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)  
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE  
        Fastwire::setup(400, true);  
    #endif  
  
    // initialize serial communication  
    // (115200 chosen because it is required for Teapot Demo output, but it's  
    // really up to you depending on your project)  
    Serial.begin(115200);  
    while (!Serial); // wait for Leonardo enumeration, others continue immediately
```

<그림 2-1-2-4-6>

아두이노 시리얼 통신 - 시리얼 포트, 보드레이트 지정

Wire.begin();을 통해 데이터 전송의 시작을 알리고, Serial.begin(115200);을 통해 보드레이트를 지정하여 시리얼 통신(특히 여기서는 시리얼 모니터<sup>8)</sup>와 콘솔<sup>9)</sup>에 출력하는 것)의 시작을 알린다.

#### 5.1.2.3.4.2.3. 데이터 전송

uint8\_t teapotPacket[17] = { '\$', 0x02, 0,0, 0,0, 0,0, 0, 0, 0, 0x00, 0x00, 'W', 'r', 'n' }; 여기서 지정한 0의 개수가 바로 전송하는 데이터의 개수이다. 아래 사진에서 teapotpacket[] = 로 지정한 것들이 전송할 데이터이다. 이 데이터들이 순차적으로 0에 대응되어 전송된다. (다음장)

7) 커서를 그 줄의 맨 앞으로 옮기는 것.

8) 시리얼 통신 중에 열어서 통신의 상태를 확인할 수 있는 아두이노의 창 이름이다. 시리얼 모니터에 무언가를 출력하게 하기 위해서도 특정한 명령어가 필요하며, 이것은 프로세싱으로 전송하는 데이터 전송 과정과는 다르다. 데이터를 전송하여 출력하는 것과 데이터를 전송하여 값을 처리하는 것은 상당히 다르다.

9) 아두이노의 시리얼 모니터와 같이 프로세싱에서 시리얼 통신 상태를 확인할 수 있는 작은 창이다. 시리얼 모니터라고 생각하면 된다.

```

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0];
    teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4];
    teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8];
    teapotPacket[7] = fifoBuffer[9];
    teapotPacket[8] = fifoBuffer[12];
    teapotPacket[9] = fifoBuffer[13];
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    //teapotPacket[10] = fifoBuffer[28];
    //teapotPacket[11] = fifoBuffer[29];
    //teapotPacket[12] = fifoBuffer[32];
    teapotPacket[13] = fifoBuffer[33];
    teapotPacket[14] = fifoBuffer[36];
    teapotPacket[15] = fifoBuffer[37];
    Serial.write(teapotPacket, 17);
    teapotPacket[14]++; // packetCount, loops at 0xFF on purpose

#endif

```

<그림 2-1-2-4-7>

아두이노 시리얼 통신 - 데이터 전송

#### 5.1.2.3.4.3. 프로세싱

##### 5.1.2.3.4.3.1. 통신 기반 작업 설정

```

import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;

```

<그림 2-1-2-4-8>

프로세싱 시리얼 통신 - 통신 기반 작업 설정

import는 아두이노의 #define과 같이 외부 라이브러리를 삽입하는 명령어이다.

#### 5.1.2.3.4.3.2. 전송 받을 데이터 형식 지정

```
Serial port; // The serial port
char[] teapotPacket = new char[17]; // InvenSense Teapot packet
int serialCount = 0; // current packet byte position
int aligned = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] accel = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];
```

<그림 2-1-2-4-9>

프로세싱 시리얼 통신 - 전송 받을 데이터 형식 지정

데이터 처리를 통해 받은 값들을 어떤 변수에 저장할지 정한다. 특히 가속도, 오일러 각, 중력 벡터는 세 가지로 항상 묶여 오기 때문에 배열로써 처리한다.

#### 5.1.2.3.4.3.3. 데이터 처리

```

135 void serialEvent(Serial port) {
136     interval = millis();
137     while (port.available() > 0) {
138         int ch = port.read();
139         print((char)ch);
140         if (ch == '$') {
141             serialCount = 0;
142         } // this will help with alignment
143         if (aligned < 4) {
144             // make sure we are properly aligned on a 14-byte packet
145             if (serialCount == 0) {
146                 if (ch == '$') aligned++;
147                 else aligned = 0;
148             } else if (serialCount == 1) {
149                 if (ch == 2) aligned++;
150                 else aligned = 0;
151             } else if (serialCount == 12) {
152                 if (ch == '\r') aligned++;
153                 else aligned = 0;
154             } else if (serialCount == 13) {
155                 if (ch == '\n') aligned++;
156                 else aligned = 0;
157             }
158             //println(ch + " " + aligned + " " + serialCount);
159             serialCount++;
160             if (serialCount == 17) serialCount = 0;
161         } else {
162             if (serialCount > 0 || ch == '$') {
163                 teapotPacket[serialCount++] = (char)ch;
164                 if (serialCount == 17) {
165                     serialCount = 0; // restart packet byte position
166
167                     // get quaternion from data packet
168                     q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
169                     q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
170                     q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
171                     q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
172                     accel[0] = ((teapotPacket[10] << 8) | teapotPacket[11]);
173                     accel[1] = ((teapotPacket[12] << 8) | teapotPacket[13]);
174                     accel[2] = ((teapotPacket[14] << 8) | teapotPacket[15]);
175                     for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];
176                     for (int j = 0; j < 3; j++) if (accel[j] >= 2) accel[j] = -4 + accel[j];
177
178                     // set our toXilibs quaternion to new data
179                     quat.set(q[0], q[1], q[2], q[3]);

```

<그림 2-1-2-4-10>

프로세싱 시리얼 통신 - 데이터 처리

uint8\_t teapotPacket[17] = { '\$', 0x02, 0,0, 0,0, 0,0, 0, 0, 0, 0x00, 0x00, 'Wr', 'Wn' };에서 '\$'가 인식되면 데이터의 첫 부분임을 인식한다. 그 다음 0이 인식되면 이것을 데이터를 저장하기 시작하고 'Wr', 'Wn'이 인식되고 통신의 한 단위가 종료된다.

데이터의 저장은 168~176에서 이루어진다.

이로써 아두이노와 프로세싱의 시리얼 통신의 한 과정이 종료된다.

#### 5.1.2.4. MINDSTORM NXT(LEGO No. 9797), EV3(LEGO No. 45544)

왜 LEGO MINDSTORMS NXT가 로봇을 실제로 구현하는 데 유리할까?<sup>10)</sup>

1. 재활용성이 크다. 부품을 다시 사용할 수 있다.



2. 결합성능이 크다. 모든 부품들은 서로 호환된다.
3. 조립속도가 다른 로봇구현기보다 훨씬 빠르다.
4. 용접, 접착제, 가공이 필요없다.
5. 다른 로봇구현기보다 저렴하다

## 5.2. Driver Program(Open Source)

### 5.2.1. -HiTechnic Gyro Sensor RobotC EV3 Drive (Programmed by Xander)

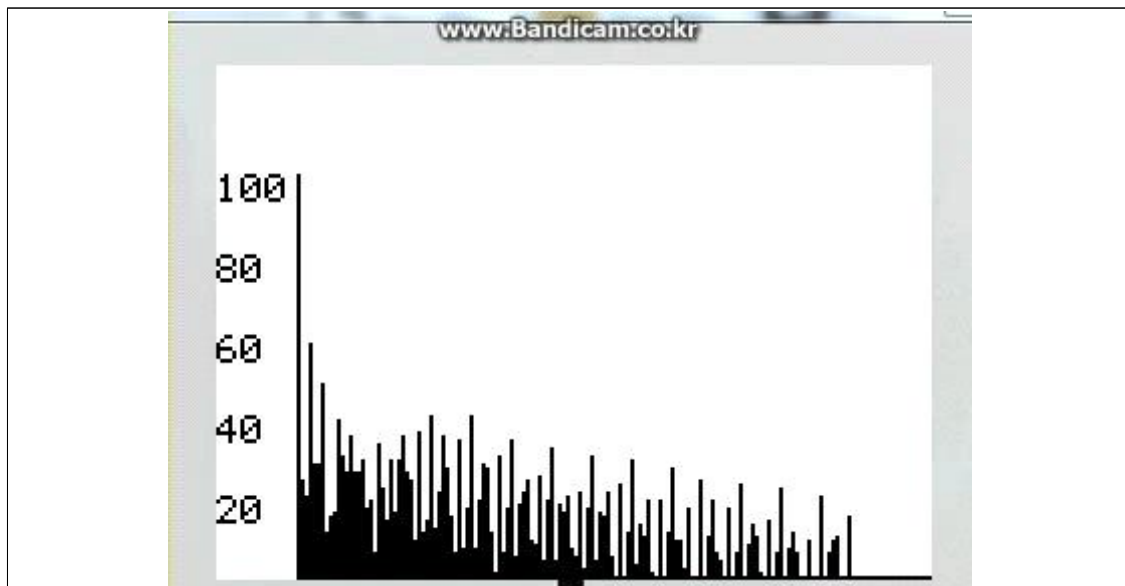
<pre> 40  #pragma systemFile 41  #include "hitechnic-sensormux.h" 42 43  #ifndef __COMMON_H__ 44  #include "common.h" 45  #endif 46 47  // This ensures the correct sensor types are used. 48  #if defined(NXT) 49  TSensorTypes HTGyroType = sensorAnalogInactive; 50  #elif defined(EV3) 51  TSensorTypes HTGyroType = sensorLightInactive; 52  #endif 53 54  typedef struct 55  { 56      tI2CData I2CData; 57      float rotation; 58      float offset; 59      bool smux; 60      tMUXSensor smuxport; 61  } tHTGYRO, *tHTGYROPtr; 62 63  bool initSensor(tHTGYROPtr htgyroPtr, tSensors port); 64  bool initSensor(tHTGYROPtr htgyroPtr, tMUXSensor muxsensor); 65  bool readSensor(tHTGYROPtr htgyroPtr); 66  bool sensorCalibrate(tHTGYROPtr htgyroPtr); 67 68  float HTGYROreadRot(tSensors link); 69  float HTGYROstartCal(tSensors link); 70  float HTGYROreadCal(tSensors link); 71  // void HTGYROsetCal(tSensors link, short offset); 72 73  #ifdef HTSMUX_SUPPORT 74  float HTGYROreadRot(tMUXSensor muxsensor); 75  float HTGYROstartCal(tMUXSensor muxsensor); </pre>	<pre> task main () {     displayTextLine(0, "HT Gyro");     displayTextLine(1, "Test 1");     displayTextLine(5, "Press enter");     displayTextLine(6, "to set relative");     displayTextLine(7, "heading");      sleep(2000);     eraseDisplay();      // Create struct to hold sensor data     tHTGYRO gyroSensor;      // Initialise and configure struct and port     initSensor(&amp;gyroSensor, S1);      time1[T1] = 0;     while(true) {         if (time1[T1] &gt; 1000) {             eraseDisplay();             displayTextLine(1, "Resetting");             displayTextLine(1, "offset");             sleep(500);              // Start the calibration and display the offset             sensorCalibrate(&amp;gyroSensor);              displayTextLine(2, "Offset: %f", gyroSensor.off);             playSound(soundBlip);             while(bSoundActive) sleep(1);         }     } } </pre>
<p>&lt;그림 2-1-2-1-1&gt; Xander(RobotC개발자)의 Gyrodriver (hitechnic-gyro.h)</p>	<p>&lt;그림 2-1-2-1-2&gt; #include "Gyrodriver.h"의 적용예 (hitechnic-gyro-test.c)</p>

Gyrodriver.c의 구동원리를 다 해설하거나 Gyrodriver.c자체를 다 읽는 것은 이 보고서의 요지를 흐릴 뿐만 아니라 의미도 없기 때문에 그냥 간략한 원리만 설명한다. Gyrodriver.c는 Xander가 작성한 287줄의 프로그램이다. 자동적으로 fGyrobias라는 변수로 오차를 수정하고, 가장 중요한 fOldangle과 fGyroangle이라는 변수를 생성한다. 이 변수들은 <그림2-2-4>에서와 같이 프로그래머에 의해 코딩되는데, float형 변수로 fOldangle의 초깃값을 -1로 두고, fGyroangle(각속도값)이 fOldangle과 같은지 체크하다가 같지 않으면 fOldangle값에 fGyroangle값을 저장한다. 이 과정에 무한히 빠르게 반복되면서(대략적으로 자이로센서의 샘플링 딜레이값인 3ms, 즉 1000분의 3초에 한 번 정도이다. sensorValue[]함수를 쓰지 못하

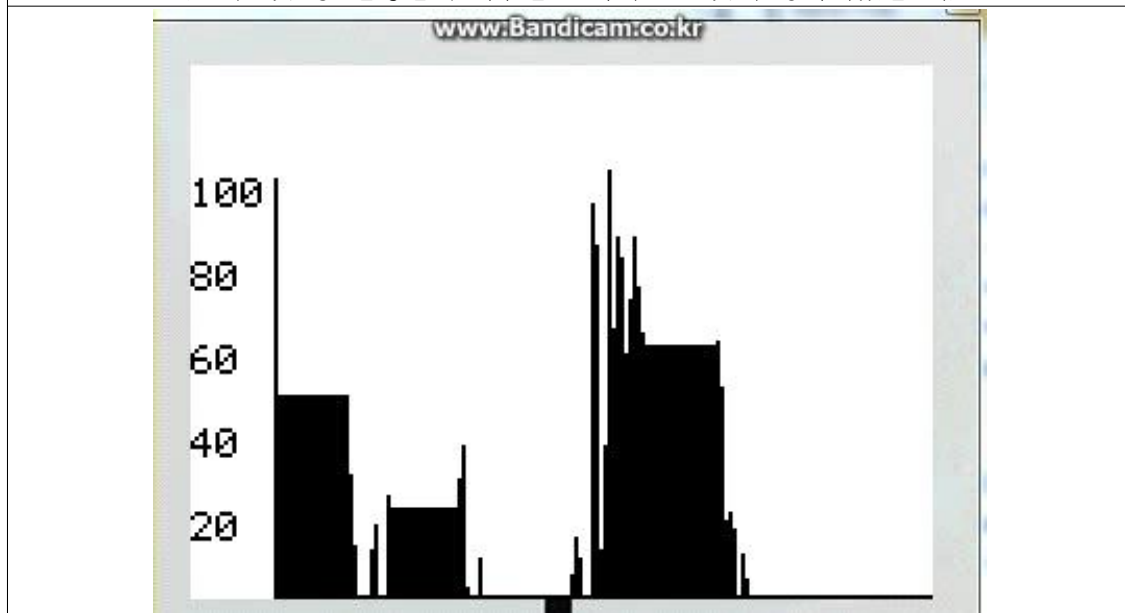
기 때문에 이 딜레이값도 프로그래머가 직접 넣어줘야 한다) “바로 전의 각속도값”과 “지금의 각속도값”이 달라진 정도를 계산하고, 그래서 각이 변한 정도의 값을 얻으며, 다음 측정을 위해 1000분의 3초 안에 “바로 전의 각속도값”에 “지금의 각속도값”을 저장하는 원리이다. 기본적인 원리는 이렇지만, 각속도측정은 이 보고서에서 크게 다뤄질 부분은 아니므로, 단순히 `if( fGyroangle != fOldangle ) { 센서값은 (long) fGyroangle; fOldangle = fGyroangle; wait1MSec(3);}`이라는 명령어가 센서값을 도출해 내는 명령어라는 걸 생각하고 지나가기만 해도 좋다. 또한, 딜레이가 있기 때문에 이 task는 다른 task와는 구별되어야만 하며, 따라서 멀티태스킹으로 코딩하는 것이 좋다.

그러나 이 센서에는 치명적 단점이 있는데, 그것이 바로 센서값 표류 현상이다.





그래프 <2-2-2-1-5> 표류하는 자이로센서값  
(그네로봇 등 일정한 움직임을 반복하는 로봇에 장착시켰을 때)



그래프 <2-2-2-1-6> 표류하는 자이로센서값(흔들었을 때)

QR 코드 <2-2-2-1-7> 양(+)으로 표류하는 자이로센서값	QR 코드 <2-2-2-1-8> 음(-)으로 표류하는 자이로센서값
---	---

위의 그래프를 간단히 설명하자면 y축은 센서값이고 x축은 시간이다.  
다음과 같이 센서는 특히나, 초기화 과정에서 충격을 주거나 처음 센서를 장착할 때 안정된 과정이 아니었을 경우에 양으로 표류하기도 하고 음으로 표류하기도 한다. <그래프 2-2-3>은 자이로센서의 표류값을 보여주는데, 약 50의 값을 가진 자이로센서를 들어 잠시동안 흔든 뒤 똑같은 자리에 똑같은 각도로 놓았을 때 자이로센서가 보여주는 변화폭이다. 스스로가 정지하고 있는지 운동하고 있는지는 센서값을 통해 알 수 있지만, 정확히 어느 부분에 어떤 각도로 있는지는 큰 충격을 줬을 때 제대로 판단하지 못한다.

이런 변화를 예방하기 위해, 센서값이 5이상으로 표류하면 그 센서는 문제가 있다고 보고 다시 처음부터 안정한 상태에서 초기화를 하기 위해 프로그램을 종료시킨다.

이 그래프를 그리는 코드는 다음과 같다.

```

1  task xy()
2  {
3      while(true)
4      {
5
6          DrawLine(20,0,200,0);
7          DrawLine(20,0,20,100);
8          DisplayStringAt(0,20,"20");
9          DisplayStringAt(0,40,"40");
10         DisplayStringAt(0,60,"60");
11         DisplayStringAt(0,80,"80");
12         DisplayStringAt(0,100,"100");
13         DisplayStringAt(200,100,"%d",SensorValue[S1]);
14     }
15 }
16
17 task downdown()
18 {
19     while(1)
20     {
21         motor[motorA]=-100;
22         wait1Msec(500);
23         motor[motorA]=100;
24         wait1Msec(50);
25         motor[motorA]=80;
26         wait1Msec(50);
27         motor[motorA]=60;
28         wait1Msec(50);
29         motor[motorA]=40;
30         wait1Msec(50);
31         motor[motorA]=20;
32         wait1Msec(50);
33         motor[motorA]=0;
34         wait1Msec(250);
35     }
36 }
37
38 task main()
39 {
40     int v;
41     int i;
42     i=0;
43     starttask(xy);
44     starttask(downdown);
45     while(i<200)
46     {
47         v=50+sensorValue[S1];
48         DrawLine(20+i, 0, 20+i, v);
49         wait1Msec(300);
50         i++;
51     }
52     stoptask(xy);
53 }
54

```

름)이 된다.

<5-2--2-121-2r2-eqw-rasfd-f-qefw-2-rq-q2-qfw->코드

2. 모든 태스크는 독립적이어야 하기 때문에 그래프의 좌표평면을 그리는 태스크를 따로 작성하고, task main에서 stoptask(태스크 동작 중단)명령을 내려주면 되기 때문에 무한 반복시켰다. 쓰는 태스크는 모두 3개로, 로봇을 작동시키는 태스크, 그래프를 그리는 메인 태스크, 좌표평면을 그리는 xy태스크로 이루어져 있다.

6~7. xy좌표평면축을 그린다. 6~12번 라인은 굳이 무한루프 안에 들어갈 필요는 없긴 하다.

8~12. 그래프의 이해를 높이기 위해 y축에 값을 표시해준다. 이 그래프를 처음 만들 때에는 초음파센서, 라이트센서, 사운드센서의 그래프를 그리기 위해 코딩한 것이라 다른 센서에 적용할 때 조금 어긋날 수는 있겠지만 그 정도는 충분히 감안해서 볼 수 있으리라 생각하고, 더 많은 센서에 범용해서 쓸 수 있는 게 중요하다고 생각했다.

13. 더 보기 편하게 하기 위해 화면 상단부에 현재의 센서값을 항상 표시하게 했다.

17~36. 반복된 운동으로 센서값을 추출하는 데에 최적화된 프로그램이다. 충격을 방지하기 위해 일부러 충격량을 최소화할수 있게 모터출력을 조금씩 낮췄다. 무한루프안에 넣었다. 태스크의 제어는 메인함수에서 할 것이다.

49. 0.3초에 한 번씩 샘플링하여 그래프에 표시한다. 200번 반복한다. v는 센서값(y값), I는 x값(시간의 흐름)이 된다.

## 5.2.2. Galaxy Note 1.0 Embedded Acceleration Sensor JAVA Driver(Programmed by Antoine Vianey)

```

1 import java.lang.reflect.*;
2 import java.util.List;
3
4 import android.content.Context;
5 import android.hardware.Sensor;
6 import android.hardware.SensorEvent;|
7 import android.hardware.SensorEventListener;
8 import android.hardware.SensorManager;
9
10 /**
11  * Android Accelerometer Sensor Manager Archetype
12  * @author antoine vianey
13  * under GPL v3 : http://www.gnu.org/licenses/gpl-3.0.html
14  */
15 public class AccelerometerManager {
16     /** Accuracy configuration */
17     private float threshold = 0.2f;
18     private int interval = 1000;
19
20     private Sensor sensor;
21     private SensorManager sensorManager;
22     // you could use an OrientationListener array instead
23     // if you plans to use more than one listener
24     // private AccelerometerListener listener;
25
26     Method shakeEventMethod;
27     Method accelerationEventMethod;
28

```

(중략)<sup>11)</sup>

<그림 2-1-2-3-1>

Galaxy Note Embedded Acceleration Sensor JAVA Driver

(Programmed by Antoine Vianey)

### 5.2.3. Galaxy Note Embedded Digital Compass Sensor JAVA Driver

11) 더 보기 편한 전체 소스코드는 첨부 소스코드 2를 참조하기 바란다.

<pre> 1 import java.lang.reflect.*; 2 import java.util.List; 3 4 import android.content.Context; 5 import android.hardware.Sensor; 6 import android.hardware.SensorEvent; 7 import android.hardware.SensorEventListener; 8 import android.hardware.SensorManager; 9 10 public class CompassManager { 11     private Sensor sensor; 12     private SensorManager sensorManager; 13 14     Method compassEventMethod; 15     Method directionEventMethod; 16 17     private Boolean supported; 18     private boolean running = false; 19 20     Context context; 21 22     public CompassManager(Context parent) { 23         this.context = parent; 24 25         try { 26             compassEventMethod = 27                 parent.getClass().getMethod("compassEvent", new Class[] { Float.TYPE, Float.TYPE, Float.TYPE }); 28         } catch (Exception e) { </pre>	
(중략)	
<그림 2-1-2-4-1>	
Galaxy Note Embedded Digital Compass Sensor JAVA Driver	

## 5.3. 기본적인 명령어 설명

### 5.3.1. RobotC

task main() {} : main program 이다. 로봇은 {}안의 명령어만을 수행한다.

drawline(x1,y1,x2,y2) : (x1,y1)에서 (x2,y2)좌표 까지 선을 긋는 명령어이다.

while() {} : () 의 조건에 부합하면 {}안에 있는 명령어를 를 반복하는 명령어이다.

SensorValue[SensorPort] : []안에 있는 Sensor Port에 있는 센서값을 읽어오는 명령어이다.

DisplayTextLine(줄번호,“내용”,변수): : 로봇의 디스플레이에 원하는 문자열을 출력한다.

### 5.3.2. Processing

void setup() : 한 번만 반복하는 초기 설정을 만드는 함수다. 이 함수 안에 size()같은 명령어가 들어간다.

void draw() : 계속 반복하는 함수로 대부분 그래픽을 구현하는데 만들어지며, Input이나 timer 제어에 따라 계속해서 모양이 바뀌도록 설정할 수 있는데, 이 때 background()함수가 이용된다.

translate(x,y) : 원점을 바꿔준다. 바꾸지 않으면, 화면의 왼쪽 상단이 원점이 된다.

background(RGB Value) : 바탕색을 지정해준다. 이게 없으면 이전 프레임의 도형이 없어지지 않는다.

line(x1,y1,x2,y2) : RobotC의 drawline에 해당하는 함수이다.

`fill( RGB value )` : 구현한 그래픽의 색깔을 지정한다.

`vertex( x,y,z,u,v )` : 다양한 그래픽을 구현하기 위해 정점을 지정한다. 명령어로 지정하기 힘든 환 형태의 종류의 경우 곡선의 모든 점들을 vertex로 지정해줘야 한다. u,v벡터는 텍스처 지정에 쓰인다.

`texture()` : 구현한 그래픽에 이미지 파일을 입힐 수 있다.

`textureMode()` : 텍스처 함수와 정점 함수가 동시에 쓰였을 때 어떻게 u,v벡터를 지정해 줄지를 정한다.

`beginShape()` : vertex로 그래픽 물체 지정을 시작한다.

`endShape()` : vertex로 그래픽 물체 지정을 종료한다.

All Rights Reserved.

CCL허가를 받은 오픈소스와 프로그램을 제외한 이 문서의 저작권은 고은진, 윤태웅, 황현조, 싸컨대사건, 로봇동아리 상상, 안산동산고등학교에 있습니다. 저작권자의 동의 없는 유포 및 재배포는 법적으로 처벌받을 수 있습니다.