논문 17-42-07-17 The Journal of Korean Institute of Communications and Information Sciences '17-07 Vol.42 No.07
https://doi.org/10.7840/kics.2017.42.7.1445

PWM과 레이저를 이용한 디지털 궤도게이지 구현

이 현 재◆, 정 경 택°, 김 변 곤*

Orbital Gauge Using Laser and PWM

Hyun-jae Lee◆, Kyung-taek Chung°, Byun-gon Kim*

요 약

궤도게이지는 열차의 운행에 따른 진동, 횡압력 등으로 궤도의 상태가 변화하기 때문에 현재의 궤도의 상태를 확인하여 선로의 최적의 상태를 유지하여 안전한 열차 운행을 위하여 사용된다. 기존의 궤도게이지는 궤도와 궤도 사이, 궤도와 가드레일 궤도의 거리를 확인하기 위하여 눈금을 확인하여야 하며 수평기는 미세조정나사 등을 사용하여 수동식으로 조작한다. 또한, 터널 안에서 작업이 이루어질 경우 눈금을 확인할 수 없어 후레쉬를 휴대하여만 측정을 할 수 있기 때문에 궤도게이지의 사용 숙련도에 따라 작업시간에 큰 영향이 있었다. 이러한 단점을 보안하기 위하여 레이저 거리센서를 동작시켜 거리를 측정하고 LCD를 사용하여 측정값을 보여주며 수평기도 자이로센서를 사용하여 기우러진 방향을 +와 -로 0도부터 90도까지 표현해준다.

Key Words : Digital Orbital Gauge, Laser, PWM, Arduino, ATmega-128

ABSTRACT

The orbital gauge is used for safe train operation by checking the state of the current orbit and maintaining the optimal condition of the track because the state of the orbit changes due to vibration and lateral pressure of the train. Conventional orbital gauges have confirmed the scales to know the distance between orbits and the distance between orbit and guardrail track, and the leverer has been manually operated using fine adjustment screws. In addition, the working time has been greatly affected by the skill in use of the orbital gauge because it has been possible to make measurement only by carrying the flash to read the scale when the work has been performed in the tunnel. In order to complement this disadvantage, the laser distance sensor is operated to measure the distance and the LCD shows the measured value, and the horizontal gyro sensor is used to express the tilted direction from 0 to ±90 degrees.

Ⅰ. 서 론

열차가 궤도를 지나가면서 진동, 횡압력 등에 의해 궤도의 상태가 조금씩 변한다. 이렇게 변형된 궤도를 안정적인 열차 운행을 위하여 궤도의 상태를 원래대로 되돌릴 필요가 있다. 궤도의 현재 상태를 확인할 수 있는 장비가 궤도 게이지이다[1]. 현재의 궤도 게이지는 아날로그 방식으로 궤도와 궤도간의 거리 및 수평측정을 하여 궤도의 높이와 수평들을 원래의 상태로 보수한다. 궤도 게이지는 아날로그 방식이기 때문에 사용자가 궤도의 종류마다 여러 개의 눈금 중에 궤도에 맞는 눈금을 찾고 그에 맞는 미세조정 나사와 거리 값을 읽어주어 기록자가 편히 궤도의 상태를 적으면서 궤도사이의 거리를 측정할 수 있다. 이러한 이유

◆ First Author : Kunsan National University Department of Electronic Engineering, lee97567@naver.com, 학생회원
° Corresponding Author : Kunsan National University Department of Electronic Engineering, eoe604@kunsan.ac.kr, 종신회원
* Kunsan National University Department of Electronic Engineering, bgkim@kunsan.ac.kr, 정회원
논문번호 : KICS2017-03-075, Received March 14, 2017; Revised May 18, 2017; Accepted July 3, 2017

때문에 궤도 게이지의 숙련자와 미숙련자의 작업 속도와 작업 효율의 차이가 생긴다. 현재의 궤도 게이지를 사용자의 조절이 필요한 미세 조정 나사와 눈금을 없애고 최소한의 조작과 스위치를 누르면 궤도간의 거리, 수평 등을 손쉽게 측정하여 초보자도 쉽게 사용할 수 있고 숙련자도 기존의 측정 절차에서는 변한 부분이 없기 때문에 숙련자와 비 숙련자 모두 작업 효율이 높다. 또한 궤도의 거리값을 기록하는 기록자에게도 거리를 말해주어야 한다. 이때 아날로그 방식의 궤도게이지는 눈금을 읽고 큰소리로 기록자에게 읽어주어야하지만 디지털 궤도게이지는 측정과 동시에 음성으로 출력되기 때문에 신속히 작업을 수행해 나갈 수 있다. 이러한 이점들 때문에 디지털 방식의 궤도게이지를 개발하였다.

Ⅱ. 거리측정기법

2.1 수평 측정

궤도의 수평을 맞추기 위해 궤도게이지에는 기포 수평계가 존재한다. 디지털 궤도게이지에서는 수평을 기포를 관찰할 필요 없이 바로 수평을 알 수 있게 자이로센서를 사용하고 이 센서는 '각속도'를 측정하는 센서이며 특정 축을 기준으로 회전하는 속도를 나타낸다. 각속도 값을 상보필터를 거쳐야한다. 상보필터는 자이로 값을 적분하면 현재의 각도를 얻을 수 있는데 MPU6050에 전원이 들어온 시점을 0값으로 초기값을 잡을 때 그 값에서 얼마나 움직였는지를 알면 초기값과 비교하여 현재의 각도를 알 수 있다. 하지만 센서는 매순간 값이 변하기 때문에 확실한 데이터 값을 얻을 수 없다. 따라서 우리는 센서의 평균적인 값을 현재값으로 생각하여 센서의 값을 사용하게 된다. 상보필터는 기본적으로 식(1)과 같다[2,3].

$$\angle = 0.98*(\angle + gyrData*dt) + 0.02*(accData) \quad (1)$$

수식(1)의 좌변의 $\angle$은 출력할 각도이며, gyrData는 자이로센서의 값을 의미하며, dt는 적분할 시간, accData는 가속도를 이용한 각도 데이터이다. 수식(1)의 우변의 $\angle$은 이전 각도 값을 의미한다. 수식(1)을 통하여 자이로센서의 값에 적분할 시간을 곱하여 각도로 단위를 통일해준 후 더하면 현재의 각도를 얻을 수 있다[4,5].

2.2 거리 측정

레이저 거리측정 방법에는 크게 5가지 방법이 있다. Pulsed LASER, Frequency Modulation, Phase shift(CW), Triangulation, Interferometry 5가지 방법이 있으며, 본 논문에서 사용한 레이저의 거리측정방법은 그림 1의 Triangulation 방법이다. 이 방법이 레이저로 거리를 측정할 때 가장 많이 사용되는 방법이다.

그림 2는 본 논문에서 사용되는 레이저 거리측정 센서로 그림 1의 Triangulation 방법으로 거리를 측정한다. 그림 2의 오른쪽 아래 부분에 레이저 센서가 있으며, 그 위에 렌즈 부분이 그림 1에서의 Linear diode array or position detector 부분에 해당한다. 그림 1을 보면 레이저가 조사되어 상이 맺히면 감지기 렌지가 레이저 포인트를 찾고 렌즈의 뒤의 감지기에 상이 맺게 된다. 감지기에 맺힌 △d와 Θ값에 따라 현재의 거리 값을 계산해 준다[6].

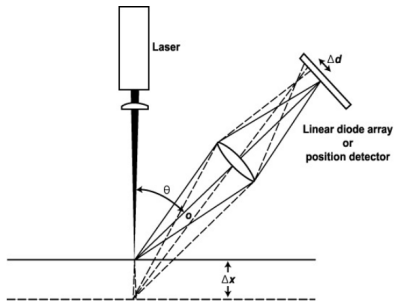

그림 1. Triangulation 측정방법
Fig. 1. Measuring method of Triangulation

그림 2. 레이저 거리측정 센서
Fig. 2. Laser distance measuring sensor

2.3 측정결과의 음성 출력

궤도 보수 작업 시에 궤도게이지를 들고 작업하는 사람 옆에 궤도의 현 상태를 기록 하는 기록자가 있다. 음성 출력의 장점은 기록자에게 빠른 궤도간의 거리를 전달할 수 있고 야간 또는 터널 작업 시에도 거리값을 쉽게 기록 할 수 있다. 음성 출력 방법은 ATmega128의 PWM 단자를 사용하며 음성을 출력하기 위해서는 PCM과정을 거쳐 헤더파일이 만들어 진

다. ATmega128에서 PCM과정을 거쳐 만들어진 파일의 sample rate와 2진수의 크기 등을 고려하여 소스를 작성하고 PWM단자를 통하여 2진수를 보내면 AMP를 통하여 소리가 증폭되어 원하는 숫자가 재생된다.

PCM변조과정을 살펴보면 ATmega128의 PWM 단자를 사용하여 음성을 출력하기 위해서는 여러 과정이 필요하다. 그림 3에서 아날로그 신호를 만드는 과정은 TTL음성서비스 홈페이지를 통하여 원하는 음성을 재생하고 녹음해야 한다. 이때 몇 가지 녹음 옵션을 선택해야 하는데 Sampling Frequency는 8KHz로 설정하고 Bits per sample은 8 bits per sample로 설정해야 한다. 다음 단계로는 양자화기와 부호기 과정이 필요하며 이것을 합쳐서 A/D변환기 라고 한다. A/D변환을 쉽게 해주는 '010 Editor'라는 프로그램이 있다. 녹음한 파일을 이 프로그램으로 열어보면 Sampling Frequency를 8KHz로 분할하여 16진수로 표시한다.

그림 4와 같이 음성의 가운데 지점을 무음으로 보고 16진수 0x80 기준으로 위아래로 위상이 변하는 것을 16진수 파일로 만들어준다. 그림 3의 양자화기와 복호기와 필터 부분을 수신기라 한다. ATmega128의 PWM 단자가 수신기 역할을 하며 1초에 8000개의 16진수 코드를 PWM 단자를 통하여 신호를 생성하여 음성을 출력시킨다.

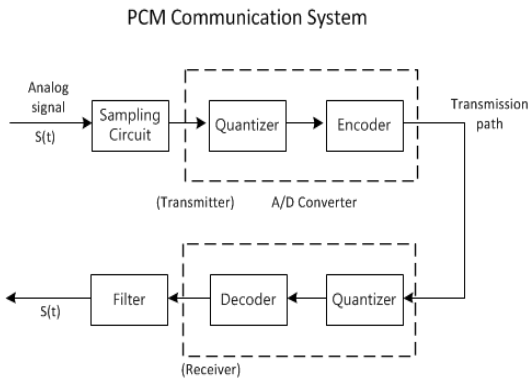

그림 3. PCM 통신 시스템
Fig. 3. PCM Communication System

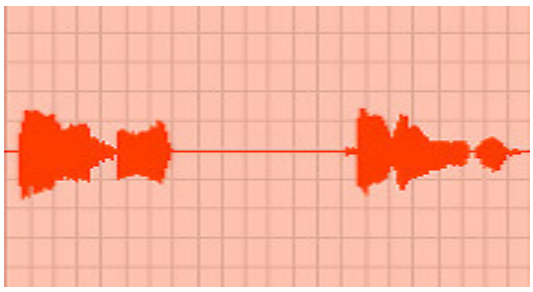

그림 4. 아날로그 신호 샘플
Fig. 4. Analog signal samples

$$\frac{f_{ocs}}{d_r} = \frac{16*10^6}{2000} = 8000Hz \leftrightarrow \frac{1}{8000}sec = 1.25*10^{-4}\text{sec} \quad (2)$$

수식 (2)에서 f_{ocs}는 시스템 주파수, d_r은 분주비이다. 수식 (2)는 ATmega128 소스코드 작성시에 필요한 수식이다. ATmega128의 기본 주파수는 16MHz이다. 처음 음성 파일을 녹음할 때 Sampling Frequency를 8KHz로 설정하였기 때문에 사용자 주파수가 8KHz로 만들어 주는 분주비를 찾아야한다. 16MHz일 경우 2000의 분주비를 가지게 되면 8KHz의 주파수를 만들 수 있다. 주파수의 역수를 취하면 시간을 얻을 수 있으며 8KHz를 역수는 125usec가 나온다. 즉, 125usec마다. PWM 단자를 통하여 16진수 신호를 보내면 음성을 출력할 수 있게 한다. 이 PWM 신호를 MINI AMP를 통하여 신호를 증폭시켜 스피커를 통해 재생시키면 디지털신호를 다시 아날로그 신호로 바꾸어 재생되는 것을 확인 할 수 있다.

Ⅲ. 아날로그 궤도게이지 사용방법

그림 5는 아날로그 궤도게이지의 궤도 사이의 거리는 측정하는 방법이다. 왼쪽부분의 다리를 궤도 한쪽에 붙이고 오른쪽의 움직이는 다리를 반대쪽 궤도 끝으로 붙이면 동그라미 부분의 눈금을 통해 궤도간의 거리를 확인해야 한다. 그림 5의 동그라미 부분의 눈금은 그림 6과 같다. 그림 6과 같이 눈금의 값이 확실하지 않기 때문에 정확한 값을 빠르게 읽는 것이 어렵다.

그림 7은 수평을 측정하는 방법이다. 궤도를 올려놓은 상태에서 미세조정 나사를 돌려 수평자의 기포를 중앙에 위치시킨 후 눈금 값을 보고 궤도의 수평을 조절한다.

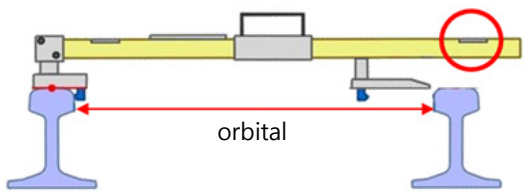

그림 5. 아날로그 궤도게이지 거리측정
Fig. 5. Analog orbital gauge distance measurement

그림 6. 아날로그 궤도게이지 거리측정 눈금
Fig. 6. Analog orbital gauge Distance measurement scale

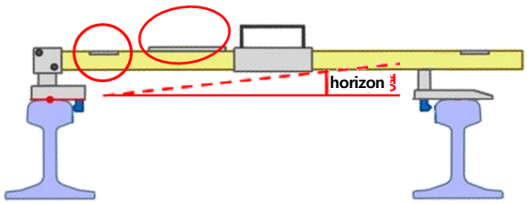

그림 7. 아날로그 궤도게이지 수평측정
Fig. 7. Analog orbital gauge horizontal measurement

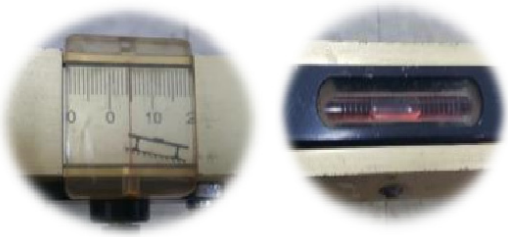

그림 8. 아날고그 궤도게이지의 수평눈금과 수평자
Fig. 8. Horizontal scale and horizontal scale of analog orbital gauge

그림 8의 왼쪽 부분은 그림 7의 왼쪽 동그라미의 수평 눈금에 해당하고, 그림 8의 오른쪽 부분은 그림 7의 오른쪽 동그라미의 수평자에 해당한다. 따라서 아날로그 방식의 불편함과 난해함을 해소하고, 시간 절약과 정확도를 높일 수 있도록 처음으로 디지털 궤도게이지를 개발한다.

Ⅳ. 동작회로

아날로그 궤도 게이지는 미세 조정 나사와 다수의 눈금들 중 상황에 맞는 눈금을 찾아 확인해야 하는 불편함이 있다. 하지만, 미세 조정 나사와 다수의 눈금 대신 LCD화면과 레이저와 자이로 센서와 버튼을 이용하여 궤도사이의 거리를 쉽게 얻을 수 있다. 센서를 제어하기 위하여 아두이노를 사용하였다.

그림 9는 아두이노와 연결되어있는 센서와 소자들

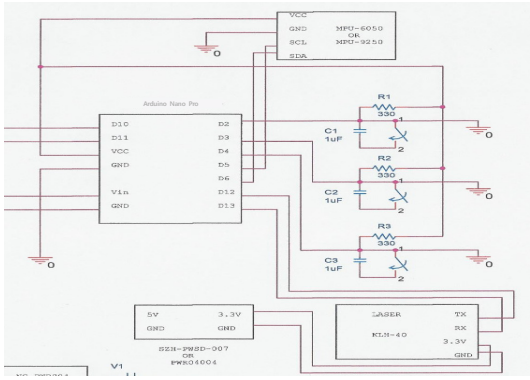

그림 9. 아두이노 회로도
Fig. 9. Arduino Schematic

의 회로도이다. 거리를 측정하기 위한 KLH-40(레이저 거리측정 센서)를 사용하며 수평기를 사용하기 위해 MPU-9250 (자이로센서)를 사용하였다. KLH-40은 입력 전압이 3.3V을 사용한다. 아두이노 나노 제품은 3.3V는 불안전하기 때문에 SZH-PWSD-007을 사용하여 5V를 3.3V로 변환하여 KLH-40에 전압을 공급해준다. 거리측정과 수평을 측정하기 위한 동작 스위치가 3개 있으며 2개는 거리측정용 스위치이고 1개는 현재의 수평을 측정해주는 스위치로 이루어져 있다. 아두이노에서는 레이저 센서에서 값을 받아 레이저 센서의 피드백 데이터를 지우고 순수 거리 데이터를 저장하며 자이로 센서에서는 수평 값을 필터링하여 처리한 후에 ATmega128과 UART를 사용하여 거리 값과 수평 값을 넘겨준다.

그림 10은 ATmega128에 연결된 장치들 크게 두가지가 있다. 음성을 출력하기 위한 스피커, 거리 값과

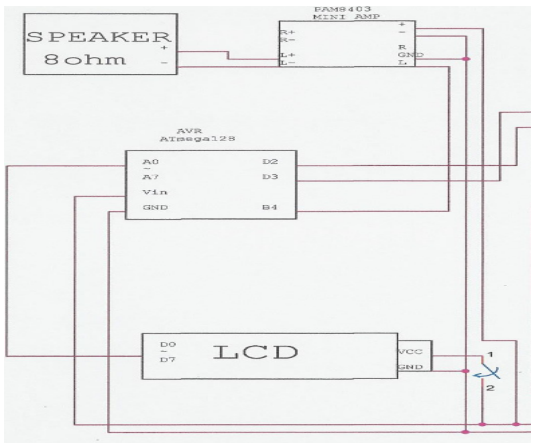

그림 10. ATmega128 회로도
Fig. 10. ATmega128 Schematic

수평 값을 보여주기 위한 LCD가 있다. ATmega128에서 PWM 단자를 통하여 나오는 음성데이터를 증폭시키기 위한 PAM8403_MINI_AMP가 ATmega128과 연결되어 있다. 그 외 아두이노에서 받은 거리 값과 수평 값을 보여주기 위해 LCD와 연결되어 있으며 터널에서 작업할 때 LCD가 안 보이는 것을 대비하여 LCD 백라이트를 온/오프할 수 있도록 하였다.

Ⅴ. 제작과정

제작과정으로는 4단계로 나눠서 진행하였으며 1. 전체 회로도 제작, 2. H/W 설계, 3. 소스 코딩 4. 마무리 및 결과 순으로 진행하였다. OrCad를 이용하여 전체 회로도를 제작 하였고, 회로도를 토대로 하드웨어 설계를 하였다. 레이저센서를 보호하기 위해 123D 프로그램을 이용하여 레이저 케이스를 3D프린터를 사용하여 제작하였다. 자이로센서를 연결 하고 상보필터를 사용하여 수평을 측정하고 수평 데이터와 레이저의 거리계산 데이터를 아두이노에 대한 코딩을 작성하였다. ATmega128에는 스피커와 LCD를 연결하였고 음성 출력 PCM변조 방식을 사용하여 PWM단자를 통해 음성을 출력하였다. ATmega128의 툴로는 “AVR Studio4”를 사용하였고, 아두이노 컴파일러 툴로는 “arduino”를 사용하였다.

5.1 전체 회로도 제작

아두이노와 ATmega128에 연결해야 할 포트위치 파악 및 센서의 종류를 찾아보고 전체적인 회로도를 구성할 수 있었다. 그림 11은 OrCad를 사용한 회로도면이다.

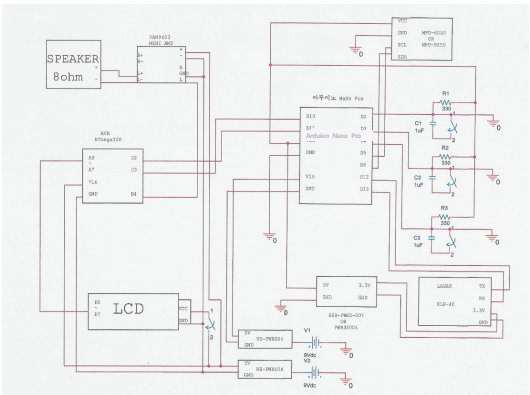

그림 11. 디지털 궤도게이지 전체 회로도
Fig. 11. Digital Orbital Gauge Full Schematic

5.2 H/W 설계

그림 12는 실제 궤도 게이지의 눈금 부분과 수평기 부분을 빼내고 궤도게이지 내부에 선을 넣어 작업하였고, 아두이노 본체와 궤도와 궤도를 측정하기 위한 버튼이 있으며 아두이노에 연결된 레이저와 자이로센서를 실제 궤도게이지 안에 넣기 전단계의 사진이다.

그림 13은 ATmega128의 H/W사진이며 LCD 연결

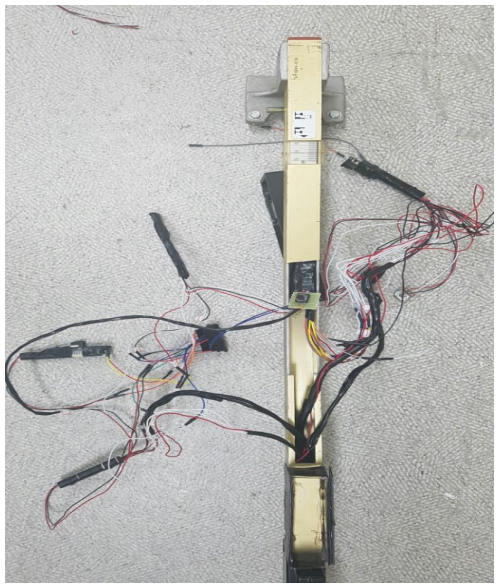

그림 12. 궤도게이지의 아두이노 위치와 구성
Fig. 12. Arduino position and composition of orbital gauge

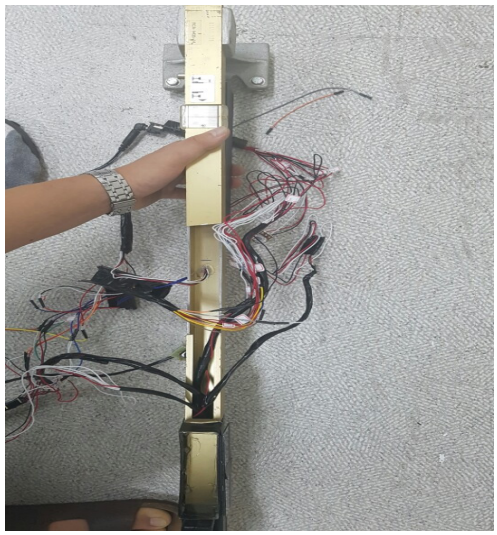

그림 13. 궤도게이지의 ATmega128의 위치와 구성
Fig. 13. ATmega128 position and composition of orbital gauge

부분과 소리 신호를 증폭하기 위한 AMP가 있으며 수평기 버튼과 궤도와 가드레일의 길이를 측정하는 버튼이 있다.

5.3 소스 코딩

그림 14는 아두이노 소스의 헤더파일을 선언하는 부분이며 아두이노와 레이저, 아두이노와 ATmega128은 UART통신을 사용하여 데이터를 주고받는다. Wire.h, MPU6050.h, I2Cdev.h는 자이로 센서로 정확한 수평 측정을 위해 상포필터를 사용하였고 상포필터에 필요한 헤더파일들이다.

그림 15는 상포필터를 통해 얻은 수평 결과 값을 Gyro[0]에는 -,+ 즉, 좌우로 상수를 넣어주고 Gyro[1]

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include "Wire.h"
#include "MPU6050.h"
#include "I2Cdev.h"

#define pi 3.141592
#define RADIANS_TO_DEGREES 180/3.14159
#define fs 131.0;
MPU6050 mpu;

SoftwareSerial laser(12, 13);
SoftwareSerial avr(10, 11);
```

그림 14. 아두이노 헤더파일 부분
Fig. 14. Header file part of Arduino

```
avr.write(0x69);
Gyrosave = angle_y;
constrain(Gyrosave, -90, 90);
if (Gyrosave < 0)
{
  Gyrosave = -Gyrosave;
  Gyro[0] = 10;
  Gyro[1] = Gyrosave / 10;
  Gyro[2] = Gyrosave % 10;
}

else
{
  Gyro[0] = 11;
  Gyro[1] = Gyrosave / 10;
  Gyro[2] = Gyrosave % 10;
}
Serial.println(Gyrosave);
avr.write(Gyro[0]);
avr.write(Gyro[1]);
avr.write(Gyro[2]);

sw = 0;
```

그림 15. 아두이노의 수평데이터 취득 방법
Fig. 15. How to obtain horizontal data of Arduino

은 00 ~ 90사이의 10의 자리숫자를 저장하고 Gyro[2]에는 1의 자리숫자를 저장 후 ATmega128에게 데이터를 전송해준다.

그림 16은 아두이노에서 레이저를 제어하는 부분의 소스이며 레이저 모듈의 측정을 위한 명령어를 define으로 정의한 후에 명령어를 보내고 그 결과 값을 변수에 저장한다. 이 때, 받은 데이터 값에 오류 여부를 판단하여 오류가 있을시 다시 측정 명령을 내린다. 오류가 검출되지 않을 때까지 계속해서 측정하며 오류가 없을시 측정을 종료한다[7].

그림 17은 ATmega128의 헤더파일이다. 한글 음성출력을 위하여 많은 양의 사용자 헤더파일을 추가하였고 사용자 헤더파일 아래에 있는 해더파일은 UART통신과 PWM, 인터럽트를 사용 그리고 LCD 제어를

```
for (i = 0; i < 5; i++)                    // Laser single measurement
{
  laser.write(LaserCheck[i]);
}

for (j = 0; j < 11; j++)                   // Save measurement
{
  Receive[j] = laser.read();
}

if (Receive[6] != 0x2E)                    // Error detection
{
  avr.write(0x61);
  delay(700);
  continue;
}
```

그림 16. 아두이노의 레이저 거리 측정 방법
Fig. 16. How to measure laser distance of Arduino

```
#include <avr/pgmspace.h>
#include "One.h"
#include "Two.h"
#include "Three.h"
#include "Fuor.h"
#include "Five.h"
#include "Six.h"
#include "Seven.h"
#include "Eight.h"
#include "Nine.h"
#include "Zero.h"
#include "null.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <compat/deprecated.h>
#include "tpk_lcd8.h"
#define SAMPLE_RATE 8000;
```

그림 17. ATmega128 헤더파일 종류
Fig. 17. ATmega128 Header File Type

위한 헤더파일이다.

그림 18은 사용자 헤더파일중 하나의 파일이며 '구' 라는 숫자를 읽어주는 파일의 샘플링이다[8].

그림 19는 아두이노에서 현재의 상태를 LCD에 표시하기 위해 ATmega128에게 특정 상태마다 특정 숫자를 보낸다. 그 특정 숫자에 해당하는 문자를 LCD에 글자를 적어준다[9].

```
unsigned int LNine = 3472;

unsigned char AudioNine[] PROGMEM = {
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7E,
    0x7D, 0x7C, 0x7B, 0x7C, 0x7F, 0x80, 0x83, 0x85,
    0x86, 0x86, 0x85, 0x83, 0x81, 0x80, 0x7E, 0x7E,
    0x7E, 0x7F, 0x80, 0x81, 0x83, 0x83, 0x83, 0x83,
    0x82, 0x80, 0x80, 0x7F, 0x7E, 0x7E, 0x7F, 0x80,
    0x80, 0x81, 0x81, 0x81, 0x81, 0x80, 0x80, 0x7F,
    0x7F, 0x7E, 0x7E, 0x7F, 0x80, 0x80, 0x80, 0x81,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x7F, 0x7E, 0x7D, 0x7D, 0x7E, 0x7E, 0x80, 0x80,
    0x81, 0x83, 0x82, 0x82, 0x82, 0x81, 0x81, 0x80,
    0x80, 0x81, 0x82, 0x82, 0x83, 0x82, 0x82, 0x81,
    0x81, 0x80, 0x80, 0x80, 0x80, 0x81, 0x82, 0x82,
    0x82, 0x81, 0x80, 0x7F, 0x7F, 0x7E, 0x7E, 0x7F,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x7F,
    0x7F, 0x7F, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7E,
    0x7E, 0x7D, 0x7D, 0x7F, 0x80, 0x81, 0x82, 0x81,
    0x80, 0x80, 0x7F, 0x7F, 0x7E, 0x7F, 0x7F, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x7F, 0x7F, 0x7F, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x81, 0x81, 0x81,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x81, 0x81,
    0x80, 0x80, 0x7F, 0x80, 0x81, 0x81, 0x82, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x81, 0x81, 0x82, 0x82,
    0x81, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x80,
    0x80, 0x81, 0x82, 0x80, 0x80, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x7F, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x80, 0x80, 0x81, 0x80, 0x80,
    0x80, 0x7F, 0x7E, 0x80, 0x80, 0x80, 0x80, 0x7F,
    0x80, 0x80, 0x7F, 0x7F, 0x7F, 0x80, 0x82, 0x80,
    0x80, 0x80, 0x80, 0x82, 0x82, 0x7F, 0x7F, 0x7F,
    0x80, 0x80, 0x81, 0x80, 0x7F, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x80, 0x7F, 0x80, 0x7F, 0x81, 0x81,
    0x82, 0x81, 0x80, 0x7F, 0x7E, 0x7E, 0x83, 0x82,
    0x83, 0x83, 0x82, 0x80, 0x7D, 0x7E, 0x7E, 0x7E,
    0x80  0x81  0x82  0x81  0x81  0x80  0x80  0x80
```

그림 18. ATmega128 음성 사용자 헤더파일
Fig. 18. ATmega128 voice user header file

```
if (see[0] == 0x60) {lcd_display_clear(); lcd_cursor_home(); lcd_string("Normal Check"); see[0] = 0x00; break;}
else if(see[0] == 0x61) {lcd_display_clear(); lcd_cursor_home(); lcd_string("ReChecking"); see[0] = 0x00; break;}
else if(see[0] == 0x62) {lcd_display_clear(); lcd_cursor_home(); lcd_string("End Checking"); break;}
else if(see[0] == 0x63) {lcd_display_clear(); lcd_cursor_home(); lcd_string("G Step1 Check"); see[0] = 0x00; break;}
else if(see[0] == 0x64) {lcd_display_clear(); lcd_cursor_home(); lcd_string("ReChecking"); see[0] = 0x00; break;}
else if(see[0] == 0x65) {lcd_display_clear(); lcd_cursor_home(); lcd_string("Step1 End"); see[0] = 0x00; break;}
else if(see[0] == 0x66) {lcd_display_clear(); lcd_cursor_home(); lcd_string("G Step2 Check"); see[0] = 0x00; break;}
else if(see[0] == 0x67) {lcd_display_clear(); lcd_cursor_home(); lcd_string("ReChecking"); see[0] = 0x00; break;}
else if(see[0] == 0x68) {lcd_display_clear(); lcd_cursor_home(); lcd_string("Step2 End"); break;}
else if(see[0] == 0x69) {lcd_display_clear(); lcd_cursor_home(); lcd_string("Gyro"); break;}
else break;
```

그림 19. 아두이노에서 받은 현재 상태를 LED표시
Fig. 19. LED status display from Arduino

Ⅵ. 구 현

그림 20은 디지털 궤도게이지의 사진이다. 왼쪽 네모난 박스가 레이저 상자이며 버튼을 누르면 레이저가 동작하여 오른쪽 검정색 다리부분에 레이저 점이 찍혀 궤도와 궤도간의 거리를 측정한다.

그림 21은 궤도와 궤도사이의 거리를 측정한 결과이며 위와 같이 거리가 측정될 경우 스피커에서는 '일사 공 칠' 이라고 숫자를 읽어준다. ATmega128은 사용자가 추가할 수 있는 헤더파일의 크기가 정해져있기 때문에 숫자를 끊어서 읽어야만 했다.

그림 22는 왼쪽의 직사각형 상자가 레이저 케이스이다. 1번 버튼을 누르면 레이저에서 레이저가 발생하여 반대편 기둥에 상이 맺힌다. 그림 22의 오른쪽 사진이 레이저상이 맺히는 부분이다. 측정이 끝나면 그림 21과 같이 LCD에 거리 값이 표시되고 거리 값을 스피커를 통해 음성이 출력된다.

그림 20. 디지털 궤도게이지
Fig. 20. Digital orbital gauge

그림 21. 궤도와 궤도 사이의 거리 측정 결과
Fig. 21. Distance between orbits and orbits Measurement result

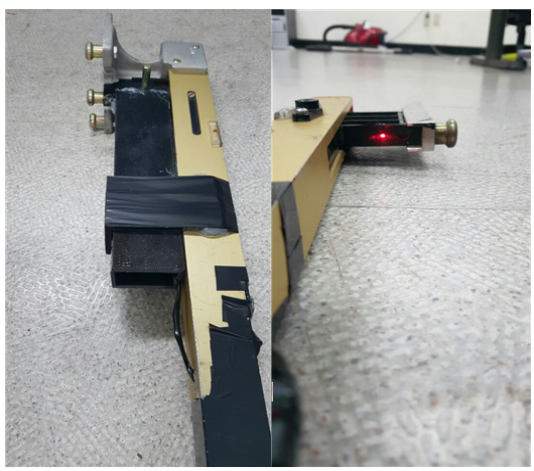

그림 22. 단일모드 측정 방법
Fig. 22. Single mode measurement method

그림 23은 궤도와 가드 레일의 거리 측정 결과이다. 가드 레일은 열차가 커브길에서 회전하다. 바퀴가 빠질 경우 열차의 바퀴가 멀리 벗어나지 않게 막는 커브길에 추가되는 궤도이다. 가드레일 궤도를 측정할 경우 그림 19와 같은 결과가 나올 때 스피커로 '오 오 일 삼 오 공' 이라고 음성이 출력된다.

측정 순서는 다음과 같다 그림 24와 같은 상태에서 2번 버튼을 누른다. 1단계 측정이 끝나면 LCD에 1stop end 라고 출력 된다. 2단계는 그림 25와 같이 레이저 케이스 왼쪽에 금색 기둥을 잡고 오른쪽으로 민다. 밀고 있는 상태에서 2번 버튼을 다시 한 번 누르면 측정을 시작하고 측정이 모두 끝나면 그림 23과 같이 LCD에 가드 레일과 궤도의 거리, 궤도와 궤도

그림 23. 궤도와 가드레일 거리 측정 결과
Fig. 23. Trajectory and guardrail distance measurement results

그림 24. 궤도와 가드레일 거리 측정방법 1
Fig. 24. How to measure orbit and guardrail distance 1

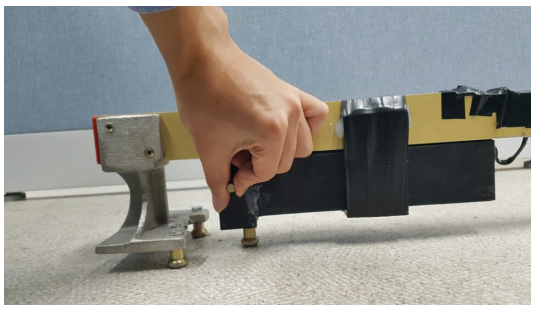

그림 25. 궤도와 가드레일 거리 측정방법 2
Fig. 25. How to measure orbit and guardrail distance 2

그림 26. 수평 측정 결과
Fig. 26. Horizontal measurement result

그림 27. 왼쪽(윗면) 오른쪽(측면)
Fig. 27. Left (Top) Right (Side)

간의 거리를 표시할 수 있다.

그림 26은 수평을 측정해주는 3번 버튼을 누른 때의 결과 그림이다. 기존에는 수평 미세 조정 나사를 돌리면서 기포를 움직여 수평을 확인하였지만 그런 복잡한 조작 없이 평지를 기준으로 -1,0,1 값이 나오며 각도에 따라 왼쪽으로 쏠리면 -가 각도 앞에 출력되고, 오른쪽으로 쏠리면 +가 출력된다.

그림 27은 실제 연구개발한 디지털 궤도게이지에 각종 센서와 표시장치를 장착하여 완성된 정면도와 측면도이다.

Ⅶ. 결 론

기존의 아날로그 궤도 게이지는 궤간 및 수평 측정과 미세 조정 나사 등의 수동식 조작부 취급과 게이지의 사용 숙련도에 영향을 받는 단점이 있으나, 본 제품은 레이저 거리 측정기를 이용해 기존의 눈금 값 판독을 디지털방식으로 대체함으로써 측정 시간 단축, 정확도 향상 및 사용자의 편리함 등의 장점을 가진다.

그림 28은 아날로그 궤도게이지와 디지털 궤도게이지의 1개 측점 검측 소요시간을 비교한 그래프이다. 정확도(mm)와 측정 시간을 1/4로 단축할 수 있음을 알 수 있다. 디지털 궤도 게이지를 사용하여 1개 측점을 검측하는데 15초를 단축할 수 있고, 1개 분기기(10개 측점)를 측정하면 150초가 단축된다. 현재의 실정에서, 평균적으로 1년 동안 본선 6000틀 정도를 4회 점검하며 측선의 경우 5600틀 정도를 2회 점검한다. 따라서 측점을 기준으로 이를 계산해보면, (본선 6000틀 * 연 4회 + 측선 5600 * 연 2회) * 150초 = 5,280,000초 ≒ 1466시간을 단축할 수 있다. 또한 이 시간을 경제적 절약 측면에서 보면 디지털 궤도 게이지 사용 측정을 전사 확대 시행 시 연간 44백만원 정도의 경제적 절감 효과를 얻을 수 있다. 또한, 아날로그 궤도 게이지와 측정 방법이 유사하여 기존에 아날로그를 사용하던 사용자도 쉽게 사용법을 익힐 수 있다. 또한, LCD 자체 광원 기능이 있어 아날로그 궤도게이지가 가진 야간작업이나 터널 작업 시에 조명 기구를 휴대해야 했던 문제점을 해결해주는 장점이 있다.. 실제 적용할 수 있는 디지털 궤도 게이지는 시간 단축, 정확도 향상, 편리함 등의 많은 장점을 갖도록 설계 개발하였다.

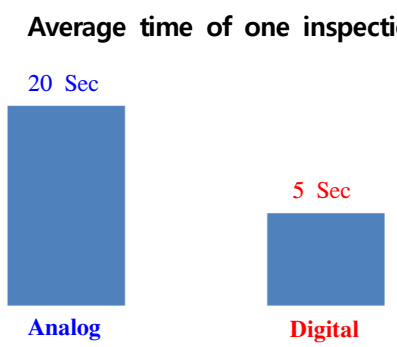

그림 28. 검측 소요 시간
Fig. 28. Time required for inspection

References

[1] G. C. Shin, D. J. Lee, H. W. Oh, Y. S. Kang, and Y. G. Park, “A study on the progress of Track Irregularity by track structure,” *Korean Soc. for Railway*, pp. 272-279, Oct. 2012.

[2] H. K. Seo, B. K. Seong, and D. H. Kim, “Moving distance measurement algorithm using an accelerometer and gyro sensor,” in *Proc. KSME*, pp. 147-148, Korea, Apr. 2014.

[3] H. S. Jung, Y. H. Choi, and J. B. Park, “Posture control of quadruped robot using gyroscope,” in *Proc. KIEE*, pp. 1836-1837, Korea, Jul. 2010.

[4] Y. H. Kim, Y. U. Yun, N. M. Kim, and Y. O. Kim, “Classification of motions by using characteristics of mobile accelerometer sensor and gravity sensor,” in *Proc. KICS ICC 2015*, pp. 718-719, Korea, 2015.

[5] S. J. Min, S. T. Choe, and W. D. Cho, “A study on the detection of sitting, standing posture using single 3-axis accelerometer,” in *Proc. KICS ICC 2015*, pp. 692-693, Korea, 2015.

[6] S. C. Yu, “Study on the laser range finder Communications compatibility,” in *Proc. Korean Soc. for Railway*, pp. 832-835, Korea, Nov. 2013.

[7] Gyeong-Yong Heo, Arduino with Internet of Things(사물인터넷을 품은 아두이노), JPUB, 2009.

[8] Sang-Seol Lee, Microcontroller AVR ATmega 128(마이크로컨트롤러 AVR ATmega128), HANBIT ACADEMY, 2013.

[9] Jin-Hwan Kim, Dong-Sik Kim, Easy to learn

AVR ATmega128 Microcontroller(쉽게 배우는 AVR ATmega128 마이크로컨트롤러), LIFE AND POWER PRESS, 2014.

이 현 재 (Hyun-jae Lee)

2012년~현재 : 군산대학교 전자공학과 재학

<관심분야> IoT 응용 시스템, AVR, 펌웨어 임베디드 리눅스 시스템

정 경 택 (Kyung-taek Chung)

1982년 : 전북대학교 전자공학과 공학사

1984년 : 전북대학교 대학원 전자공학과 공학석사

1994년 : 전북대학교 대학원 전자공학과 공학박사

1995년~1997년 : 영국 로보로 대학 선임연구원

1990년~현재 : 군산대학교 전자공학과 교수

<관심분야> IoT 응용 시스템, 임베디드 시스템

김 변 곤 (Byun-gon Kim)

1990년 : 한국항공대학교 항공전자공학과 공학사

1997년 : 전북대학교 대학원 전자공학과 공학석사

2003년 : 전북대학교 대학원 전자공학과 공학박사

2005년~현재 : 군산대학교 전자공학과 교수

<관심분야> IoT 응용 시스템, RFID 센서 네트워크