[센서네트워크]

작성자 김현수 (gustn9515@naver.com)

# 1. 서론

IoT(Internet of Things)는 처음 1988년도에 유비쿼터스 컴퓨팅(Ubiquitous Computing) 이라는 개념으로 소개가 되어 현재까지 널리 알려지게 되었다[1]. IoT는 현재 대세가 되고 있고, 여러분야에서 사용되고 있다. 예를 들면 시간이나 경로 설정만으로 자동으로 청소해주는 무인청소기, 반려견을 위한 밥주는 시간, 놀아주는 시간 등을 설정하면 함께 시간을 보내주는 IoT 프로그램, 심지어 무인자동차까지 발명중이라고 한다[2].

최근에는 IoT 관련 장비들도 많이 나와 있는데 대표적인 하나가 초저가, 초소형 컴퓨터의 하나인 라즈베리 파이(Raspberry Pi)가 있으며 이를 활용한 다양한 IoT 연구가 진행되고 있다. 라즈베리 파이에 온도, 습도, 가스, 기울센서 등을 장착하여 주변 환경 정보를 수집하고, 이를 활용하는데 사용되고 있다. 또 PC(Personal Computer) 응용프로그램 개발에 다양한 프로그래밍 언어가 사용되고 있는데 자바 프로그래밍 언어는 손쉬운 GUI(Graphical User Interface) 개발과 다른 언어에 비해 상대적으로 쉽게 배울 수 있는 측면에서 많이 활용되고 있는 프로그래밍 언어이다[3].

최근 네트워크 이용이 많아지면서 네트워크가 없으면 아무 것도 할 수 없는 일상이 되었다. IoT는 네트워크를 사용하여 일상 생활 속에서 고객 데이터를 수집하고, 수집된 정보를 서버에 전송한다. 그리고 자동으로 어떠한 일을 간편히 처리할 수 있도록 도와준다. 그리고 수집된 데이터들을 이용해서 빅데이터로 분석하는 등 다양한 연구가 시도되고 있다. 그래서 본 문서에서는 IoT 장비를 이용한 데이터의 수집, 전송, 수신, 저장, 시각화에 대한 전반적인 기능을 수행하는 프로그램 개발에 대한 상세한 내용을 다룬다.

본 문서의 구성은 다음과 같다.

2장에서는 IoT와 관련된 연구에 대해 기술한다. 3장에서는 개발 환경에 대한 정보와 환경 설정에 대해 기술한다. 4장에서는 센서 정보 수집 및 그 활용에 대한 상세한 내용을 기술하고, 결론 및 향후 연구과제는 5장에서 기술한다.

# 2. 관련연구

2.1 라즈베리 파이

라즈베리 파이는 초저가, 초소형 컴퓨터로 2012년 1월 29일 출시되었다. 라즈베리 파이 모델로은 1세대 모델 A, 1세대 모델 A+, 1세대 모델 B, 1세대 모델 B+, 2세대 모델 B등이 있다. 최근 사용되고 있는 3세대 및 4세대 모델의 간단한 정보는 표1과 같다.

표1. 라즈베리 파이 3B+, 4 비교

|  |  |  |
| --- | --- | --- |
|  | 라즈베리 파이3 B+ | 라즈베리 파이4 |
| 출시일 | 2018년 3월 14일 | 2019년 6월 24일 |
| SoC | BCM2837BO | BCM2711 |
| RAM | Quad Cortex-A53 @ 1.4GHz | Quad Cortex-A72 (ARM v8) @ 1.5GHz |
| 전원 공급 | 1GB LPDDR2 SDRAM | 1GB, 2GB, 4GB LPDDR4 SDRAM |
| 이더넷 | 5V/2.5A micro USB | 5V/3A USB-C |
| 블루투스 | Gigabit Ethernet(300Mbps) | Gigabit Ethernet |
| USB | Bluetooth 4.2 | Bluetooth 5.0 |
| 영상 음향 | Full size HDMI 포트 | Micro HDMI 포트(최대 4Kp60 지원)x2 |

라즈베리 파이 운영체제는 라즈비안(Raspbian)이다. 라즈베리 파이는 무선 네트워크 및 유선 네트워크가 가능하고, 데스크탑(Desktop)에 있는 데이터를 일부 저장할 수 있다. 그리고 GPIO에 특정한 센서를 달 수도 있으며, 특정 센서의 데이터를 수집하여 보관한다. 보관한 데이터들로 프로그래밍만 할 수 있다면, 어디서든 수집된 데이터들을 볼 수 있다.

라즈베리 파이를 활용한 예로 인스트럭터블(Instructables), 아두이노 GPS(Global Positioning Systme) 동기화 시계, 라즈베리 파이 스트레이텀(Stratum) 1 NTP(Network Time Protocol) 서버, 라즈베리 파이 온도 및 습도 모니터 등이 있다[4]. 다음 그림1은 유선으로 연결된 라즈베리 파이 이미지이다.

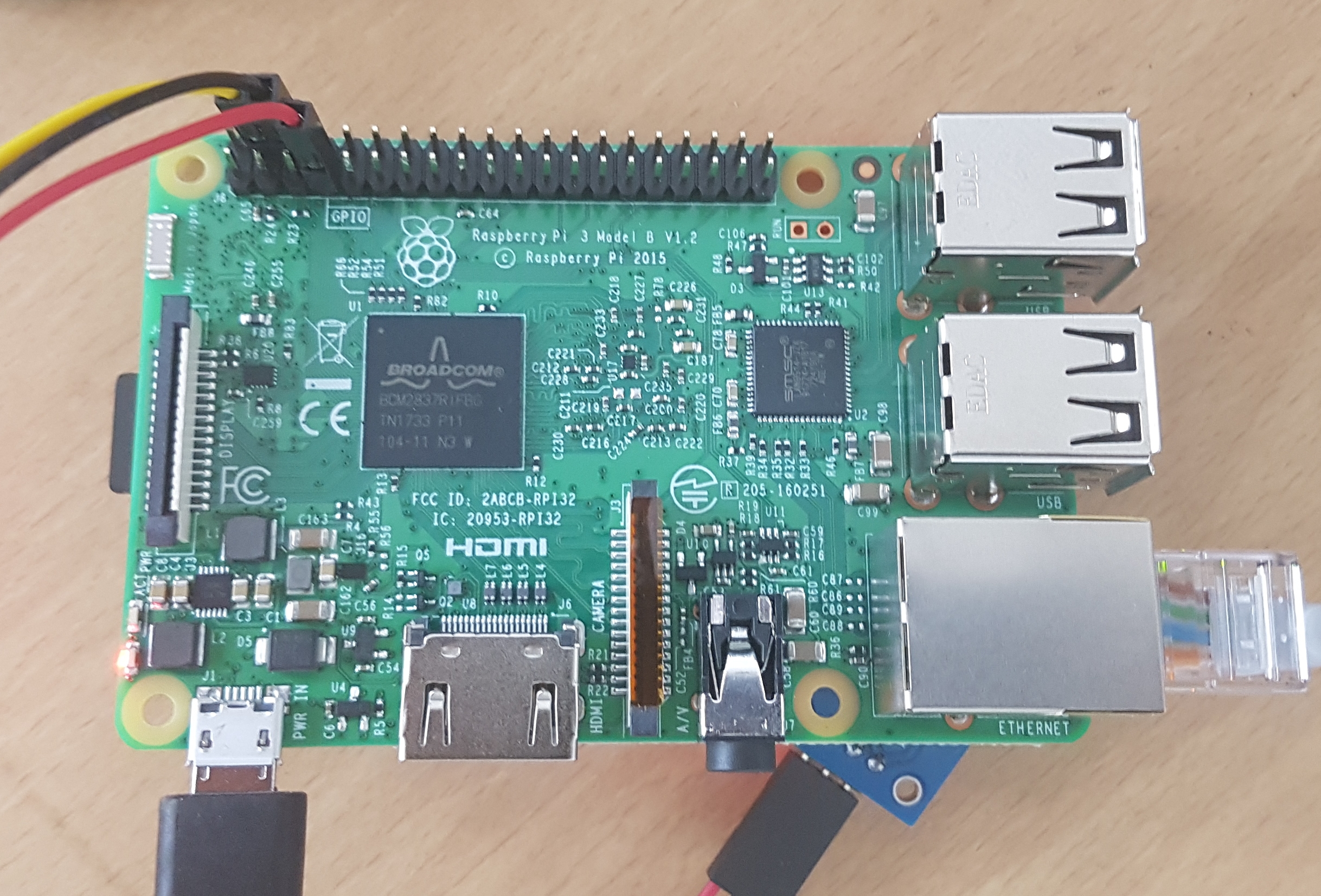


그림1. 실험에 사용된 라즈베리 파이

2.2 센서 정보

온도 및 습도 센서에는 DTH11, DTH22 센서등이 있다. DTH11는 습도의 경우 20~95%, 온도는 0~50℃ 측정할 수 있다. DTH11은 다음 그림2와 같고 정면에서 보았을 때 순서대로 VCC, OUT, NC, GND의 핀을 가지고 있다[5].

\*NC는 not connect라 연결하지 않음을 나타낸다.

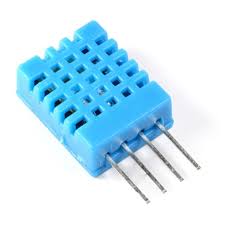


그림2. DTH11 온습도 센서

DTH22 센서는 고정밀 온습도 센서이다. 생김새는 DTH11과 비슷하며, DTH11과의 차이는 다음 표2와 같다[6].

표2. DTH11과 DTH22 비교표

|  |  |  |
| --- | --- | --- |
|  | DHT11 | DHT22 |
| 온도 측정 범위 | 0 ~ +50℃ | -40 ~ +80℃ |
| 온도 정확도 | ± 2 ℃ | ± 0.5℃ |
| 습도 측정 범위 | 20 ~ 90 %RH | 0 ~ 99.9 %RH |
| 습도 정확도 | ± 5 %RH | ± 2 %RH |

이외에도 위치 센서, 모션 센서, 길이, 지름, 크기 센서, 힘 센서, 관성 센서, 진동 센서, 등 다양한 기능에 센서들이 있다[7].

센서들을 라즈베리 파이에 연결 시켜 프로그래밍을 한다면, 대부분 센서 라이브러리(Library)를 다운받아야하고, 센서에 빛이 들어오게 하거나, 특정한 데이터를 수집하는 등 다양한 일을 할 수 있다.

2.3 IoT

IoT는 네트워크가 발달함에 따라 사물에 온도, 습도, 기울기, 압력 등 다양한 센서를 사물에 부착하고, 데이터를 수집한다. 수집된 데이터를 네트워크를 통해 서버에게 전달하여, 데이터를 전달받은 서버가 이를 저장, 분석, 시각화하는 기술이다. 분석된 데이터에 알고리즘을 적용하여 위급 상황 발생시 재난문자와 같은 알림 기능을 활용하여 실생활에 사람의 직감에 의하지 않고 데이터에 기반한 자동화된 서비스를 제공한다. 최근 구글 차트나, JavaFX 차트 등 다양한 기술을 활용하여 데이터를 시각화함으로서 직관적으로 상황파악하는 것이 가능하고, 축적된 데이터는 빅데이터로 분석하여 다양하게 활용되고 있다.

# 3. 환경정보

3.1 개발 환경

표 3. 개발환경 표

|  |  |
| --- | --- |
| 구분 | 상세 |
| OS | Windows 10, 64bit |
| 라즈베리 파이 | 라즈베리 파이 3 B+ |
| 센서 | DTH11 |
| 개발 툴(Tool) | Eclipse Java 2019-09, Eclipse Jee 2019-09 |
| 자바 버전 | jdk1.8.0\_201 |

3.2 환경 설정

3.2.1 라즈베리 파이 GPIO 센서 설정

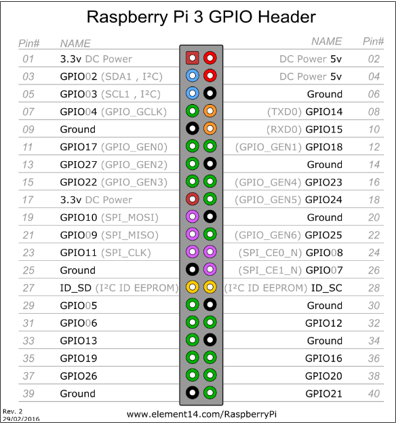


그림 3. 라즈베리 파이 PIN배치도

라즈베리 파이에 센서를 연결하기 위해서는 GPIO 핀에 대한 내용을 이해할 수 있어야 한다. GPIO 핀 설정은 그림3에서 보는 것과 같이 전원(Vcc), 그라운드(GND), 데이터의 3개 영역으로 구분할 수 있다. 센서가 동작하기 위해서는 전원과 그라운드, 데이터 3개가 정확히 설정되어야 하는데, 4번 핀에 센서의 전원을 연결하고, 6번 핀에 접지, 7번 핀에 데이터를 연결하면 하드웨어적인 세팅은 완료된다.

3.2.2 라즈베리 파이 설치

라즈베리 파이는 공식 홈페이지(https://www.raspberrypi.org)에서 라즈비안 이미지를 다운받아 설치가 가능하다. 최신 라즈비안 이미지를 다운받아 마이크로 SD에 운영체제를 옮겨야 하는데, 이를 위해 Win32 Disk Imager를 이용하였다. 이 프로그램을 이용해 라즈베리 파이를 마이크로 SD에 이미지를 탑재한 후 라즈베리 파이에 메모리만 옮겨 꼽으면 라즈베리 파이 구동이 완성된다.

3.2.3wiringPI

wiringPi는 DTH11 센서를 사용하기 위해 사용할 수 있는 오픈소스를 원격으로 간단히 설치하고 활용할 수 있다.

|  |
| --- |
| $ sudo apt-get install git-core  $ git clone https://github.com/wiringpi/wiringpi  $ cd wiringPi  $ sudo ./build  $ gpio –v  $ sudo gpioreadall  $ gcc sensor.c –o sensor -lwiringPi |

그림 4. wiringPi 설치 순서

센서 정보를 사용할 수 있는 오픈소스를 이용해 수집된 정보를 네트워크로 전송하는 코드를 추가하는 것으로 설정이 완료된다

# 4. 센서 정보 수집 및 활용기술 개발

온습도 센서를 통해 데이터를 받아온다. 받아온 데이터를 UDP(User Datagram Protocol) 방식으로 서버측에 보낸다. UDP방식은 신뢰성이 낮은 방식으로서 보낸 데이터에 대해 잘 보냈는지 확인 하지 않는다.

4.1 전체 시스템 형상

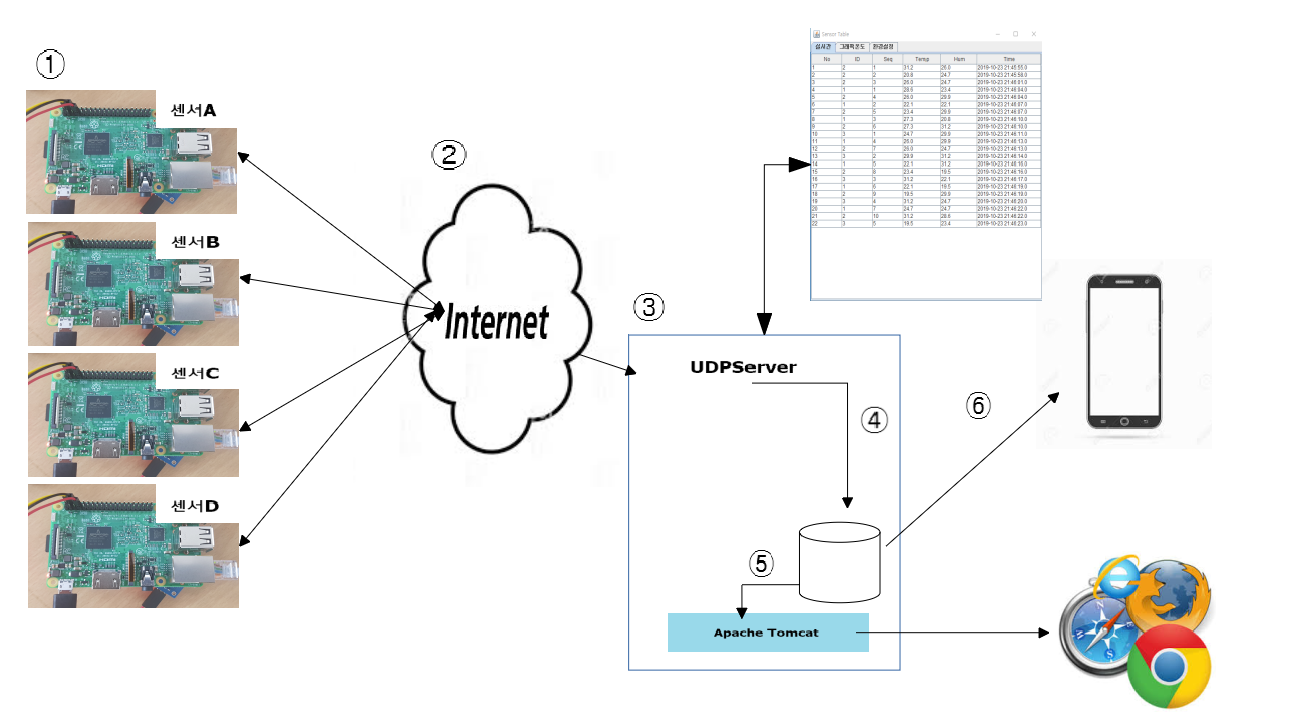


그림5. 전체 시스템 형상

센서네트워크의 전체적인 시스템 형상은 그림 5에 도시하였다. 그림에서 보는 바와 같이 DTH11 센서를 장착한 4개의 라즈베리 파이에서 온도, 습도 정보를 수집한다. 수집 온도, 습도정보를 UDP를 이용해 그림 4와 같은 형태의 데이터구조로 변화하여 관리시스템으로 전송한다(1). 이때 라즈베리 파이는 라즈비안에서 동작하기 때문에 C언어를 이용한 소켓프로그램으로 구성한다. 네트워크를 통해 전달된 UDP 센서 데이터 정보를 관리 서버가 수신하는데, 수신 영역은 자바의 소켓으로 수신한다(3). 수신한 데이터를 즉시 테이블 형태로 결과를 보여주여줄 뿐만 아니라, 향후 데이터 분석을 위해 데이터베이스에 저장한다(4). 서버에서 동작하는 아파치서버에 의해 웹서버가 동작하여 사용자가 웹브라우저를 통해 PC나 모바일에서 요청하는 경우 웹서비스를 통해 센서 정보를 확인한다(6).

4.2 센서 정보 수집, 전송

|  |
| --- |
| typedef struct \_date\_t{  int id; // raspberry pi id  int seq; // sequence id  float temperature;  float humidity;  }data\_t; |

그림6. 송수신 데이터 구조

그림 6는 라즈베리 파이에서 수신한 데이터를 C언어의 구조체로 담아 전송하는 데이터 단위를 보여주고 있다. C언어에서 구조체를 통해 전송하기 때문에 자바에서 수신한 데이터는 엔디언(Endian) 문제를 처리해야 이종 시스템간 데이터를 원활하게 처리할 수 있다. 이렇게 수신한 데이터를 자바에서 구조체를 대신하여 Sensor 클래스로 변환하여 처리하고, 그 구조는 그림6와 완전히 일치한다. 여기서 사용된 id는 각 센서의 번호 또는 라즈베리 파이에 대한 고유식별자로 사용되고, seq는 각 라즈베리 파이당 전송하는 순서를 의미한다.

|  |
| --- |
| float get\_temperature(){  // wiringPi 온도 수집  return temperature;  }  float get\_humidity(){  // wiringPi 습도 수집  return humidity;  } |

그림 7. 온습도를 가져오는 함수 구조

그림7는 DTH11 센서에서 wiringPi 모듈을 이용해 온습도 정보를 수집하고 리턴하는 함수를 보여주고 있다. 사용된 DTH11 센서로 테스트 한 결과 약 2~30% 정도의 온습도 수집시 에러율을 보였다. 이는 저가 장비를 사용하였기 때문에 향후 정밀하고 에러율이 거의 없는 장비로 대체할 필요성이 있다.

|  |
| --- |
| int main(int argc, char \*\*argv){  data\_t data;  int seq = 1;  클라이언트가 IP 주소, port 번호, id번호를 보내준다.  클라이언트가 보내준 IP 주소, port 번호를 통해 UDP 소켓(Socket)을 만든다.  while(true){  seq++;  data.id = 클라이언트가 보내준 id번호  data.seq = seq;  data.temperature = get\_temperature();  data.humidity = get\_humidity();  소켓을 통해 data를 보내준다.  sleep(3000); //3초 동안 재운다.  }  socket.close(); //socket close  return 0;  } |

그림 8. main 함수, 데이터를 전송하는 구조

실험에서는 4개의 라즈베리 파이를 사용하였고, 향후 실생활에 반영하기 위해서는 더 많은 수의 라즈베리 파이가 사용될 가능성이 있기 때문에, 코드를 유연하게 만들기 위해 main 함수 실행시 아규먼트를 사용하여 장비번호를 부여함으로써 하나의 프로그램으로 모든 장비에서 활용할 수 있도록 하였고 사용방법은 “./sensor 서버IP 서버Port 장비번호”와 같은 형식은 총 4개의 인자를 갖고 수행하게 된다. 입력받은 인자를 이용해 수신할 서버와 서버의 포트번호를 소켓으로 연결하고, data\_t 구조체에 수집된 정보와 장비 아이디 정보등을 실어 전송하는 행위를 반복적으로 수행한다.

4.3 서버 수신, 저장

라즈베리 파이가 리눅스(라즈비안)에서 C언어로 UDP 소켓을 이용하여 서버인 관리 컴퓨터에 데이터를 전송한다. 관리서버는 윈도우에서 동작하는 Java 기반의 소켓을 이용하기 때문에 데이터 처리 단위가 각각 바이트와 Char로 차이가 발생하는 문제가 생긴다. 이를 위해 1바이트 단위로 처리하는 구조체 데이터에 대한 오더링 문제 해결이 필수적이다. 수신받은 데이터는 화면에 표시하는 것은 물론이고, 향후 데이터 분석을 위해 데이터베이스에 저장하는 일까지 담당하게 된다.

|  |
| --- |
| private DatagramSocket socket; //create socket  private DatagramPacket packet; //create packet  private int port = 10001;  private DB db;  class Sensor{  int id; //raspberry pi id  int seq; //sequence id  float temperature;  float humidity;  } |

그림 9. 송수신 데이터 구조

그림 9은 수신 데이터 처리를 위한 UDP 소켓, 패킷에 대한 변수 선언과, C언어에서 구조체로 전송한 데이터를 자바에서 저장하기 위한 Sensor 클래스를 보여주고 있다. 이는 보낸 데이터를 똑같은 형태로 처리해야하기 때문에 라즈베리 파이의 data\_t 구조체와 동일한 구조를 갖는 자바의 클래스이다.

|  |
| --- |
| public UDPServer(DB db){  this.db = db; //get DataBase  socket = new DatagramSocket(port); //connect socket  Thread thread = new Thread(new Runnable(){  public void run(){  receive();  }  });  tread.start();  } |

그림 10. 서버 를 만드는 구조

그림 10은 윈도우에서 동작하는 관리서버에서 UDP 서버기능을 보이고 있다. 수신받은 데이터를 데이터베이스에 저장하기 위해 파라미터로 접속한 데이터베이스 정보를 받아 처리한다. 소켓을 연결한 후 수신을 담당하는 모듈인 receive() 기능을 독립된 쓰레드로 실행함으로써 GUI가 블록킹 당하는 상황을 배제했다.

|  |
| --- |
| public void receive(){  try{  while(true){  패키지를 통해 데이터를 받는다.  그림10 과 같이 받은 데이터를 비트연산을 통해 바꿔준다.  데이터를 데이터베이스 넣는다.  데이터베이스에서 데이터를 꺼내와서 출력해준다.  }catch(Exception e){  try{  db.close; //db disconnect  catch(Exception e){}  }  } |

그림 11. 데이터 저장 및 송수신 구조

그림 11는 UDP 서버의 수신부를 구체적으로 보여주고 있다. 데이터처리 방식이 다른 C언어와 Java사이의 이종간 데이터통신을 위해 오더링 문제를 해결하는 과정이 필수적이다. 수신된 데이터를 데이터베이스에 넣고, 최종 등록된 정보를 다시 가져옴으로써 데이터베이스에 저장이 성공했다는 사실을 직관적으로 확인할 수 있다.

|  |
| --- |
| public void converterByte(byte[] buffer) {  byte[] temp = new byte[4];  System.*arraycopy*(buffer, 0, temp, 0, 4);  sensor.id = byteToInt(temp);  sensor.id = swap(sensor.id);  System.*arraycopy*(buffer, 4, temp, 0, 4);  sensor.seq = byteToInt(temp);  sensor.seq = swap(sensor.seq);  System.*arraycopy*(buffer, 8, temp, 0, 4);  sensor.temperature = byteToFloat(temp);  sensor.temperature = swap(sensor.temperature);  System.*arraycopy*(buffer, 12, temp, 0, 4);  sensor.humidity = byteToFloat(temp);  sensor.humidity = swap(sensor.humidity);  }  public int byteToInt(byte[] arr){  return (arr[0] & 0xff)<<24 | (arr[1] & 0xff)<<16 |(arr[2] & 0xff)<<8 | (arr[3] & 0xff)  }  public float byteToFloat(byte[] bytes){  int intBits = bytes[0] << 24 | (bytes[1] & 0xFF) << 16 | (bytes[2] & 0xFF) << 8 | (bytes[3] & 0xFF);  return Float.*intBitsToFloat*(intBits);  }  public short swap(short x) {  return (short)((x << 8) | ((x >> 8) & 0xff));  }  public char swap(char x) {  return (char)((x << 8) | ((x >> 8) & 0xff));  }  public int swap(int x) {  return (int)((swap((short)x) << 16) | (swap((short)(x >> 16)) & 0xffff));  }  public long swap(long x) {  return (long)(((long)swap((int)(x)) << 32) | ((long)swap((int)(x >> 32)) & 0xffffffffL));  }  public float swap(float x) {  return Float.*intBitsToFloat*(swap(Float.*floatToRawIntBits*(x)));  }  public double swap(double x) {  return Double.*longBitsToDouble*(swap(Double.*doubleToRawLongBits*(x)));  } |

그림 12. 받은 데이터를 비트연산 하는 구조

그림12는 오더링 문제를 해결하기 위해 받아온 데이터를 비트 연산을 통해 변환하고, 변환한 데이터를 교환(swap)을 통해Sensor 클래스에 있는 인스턴스(Instance) 변수에 대입하는 일을 처리한다.

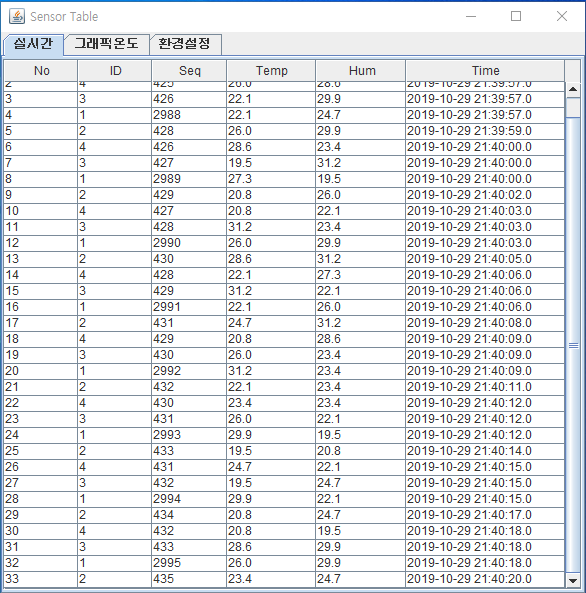


그림 13. 실시간 데이터를 받아오는 그림(Java)

그림 13는 라즈베리 파이에서 수집한 정보를 C언어로 구현되어 구조체로 보내온 데이터를 정확하게 소수 이하까지 표시 할 수 있는 테이블(Table) 형태로 보여지고 있다. 테이블에서 No는 데이터를 받아오는 순서 이며, 아이디(ID)는 센서의 아이디(센서 번호)이다. Seq(Sequence)는 센서가 지금까지 보낸 데이터의 양이며, Temp, Hum, Time은 보는 것과 같이 온도, 습도, 시간을 나타낸다.

|  |
| --- |
| private float tempArray[];  private float humArray[]  private Timestamp timeArray[]; // Current Time  private boolean isDeadArray[]; //Sensor is Dead or Alive?  private Color **color // create Color**  **private int width;**  **private int height;**  **private int interval;**  **private DataBase db;**  public Graphic(int width, int height, int interval,DataBase db) {  **this.width = width;**  **this.db = db;**  **this.height = height;**  **tempArray = new float[4];**  **humArray = new float[4];**  **isDeadArray = new boolean[4];**  **timeArray = new Timestamp[4];**  **getData(); //GetData in DataBase Method**  **color = new Color [31];**  **//Graph Renew**  **Thread thread = new Thread(**new Runnable() {  public void run() {  while(true) {  try {  Thread.*sleep*(5000);  getData();  repaint(); //Graph Repaint  }catch(Exception e) {  break;  }  }  }  }); // Thread End  thread.start();    } // Constructor End  public void getData() {  try {  // 네 개의 센서 번호(id)에 맞는 데이터 추출  for(int i=1; i<=4; i++) {  String sql = "select \*from sensor\_table where id = '"+ i +"' and time > date\_sub(NOW(), INTERVAL 10 SECOND) order by idx desc limit 1"; //DataBase Query  rs = db.getStmt().executeQuery(sql);  if(rs.next()){  tempArray[i-1] = rs.getFloat("temperature");  humArray[i-1] = rs.getFloat("humidity");  timeArray[i-1] = rs.getTimestamp("time");  isDeadArray[i-1] = false;  }else{  tempArray[i-1] = 0;  isDeadArray[i-1] = true;  }  }  }catch(Exception e) {}  }// Get Data from Query in DataBase Method  public void changeSetting(int width,int height,int interval) {  this.width = width;  this.height = height;  this.interval = interval;  } // Get Setting Data  public void paint(Graphics g) { //Override Method  getData();  g.setColor(new Color(0xFF, 0x0,0x0));  int gap = 0;  int first = 200;  for(int i=1; i<isDeadArray.length+1; i++) {  g.drawString(String.*valueOf*(i), first+gap, 30);  gap+=50;  }  first = 210;  gap = 0;  for(int i=0; i<isDeadArray.length; i++) {  if(isDeadArray[i]) {  g.setColor(new Color(0xFF, 0x0,0x0));  }else {  g.setColor(new Color(0x29, 0xFF,0x22));  }  g.fillRect(first+gap, 10, 30, 30);  gap+=50;  }  for(int i=1; i<width; i = i+interval){//Drawing Graph Width  for(int j=1; j<height; j = j+interval){//Hight  // 센서에서 특정 지점까지의 거리 계산  double lena =(Math.*sqrt*((i-xa)\*(i-xa)+(j-ya)\*(j-ya)));  double lenb =(Math.*sqrt*((i-xb)\*(i-xb)+(j -yb)\*(j-yb)));  double lenc =(Math.*sqrt*((i- xc)\*(i-xc)+(j-yc)\*(j-yc)));  double lend =(Math.*sqrt*((i-xd)\*(i-xd)+(j-yd)\*(j-yd)));  // 거리의 합  int lenTotal = (int)(lena + lenb + lenc + lend);  // 각 센서까지의 거리 비율 계산  lena = lenTotal / lena;  lenb = lenTotal / lenb;  lenc = lenTotal / lenc;  lend = lenTotal / lend;  // 각 센서당 가중치 부여  double a1=((double)lena/(double)(lena+lenb+lenc+lend))\* (int)tempArray[0];  double a2=((double)lenb/(double)(lena+lenb+lenc+lend))\* (int)tempArray[1];  double a3=((double)lenc/(double)(lena+lenb+lenc+lend))\* (int)tempArray[2];  double a4=((double)lend/(double)(lena+lenb+lenc+lend))\* (int)tempArray[3];  // 예상 기온 계산  int expected = (int)(a1 + a2 + a3 + a4);  int testInt = (int)Math.*floor*(expected);  switch(testInt){  case 10:color[10] = new Color(0,0,255);break;  // 10도는 파란색, 20도는 흰색, 30도는 빨간색으로 색상을  // 직관적으로 이해할 수 있는 색상표를 배치  case 30:color[30] = new Color(255,0,0);break;  default:break;  }  g.setColor(color[testInt]);  g.fillRect((int)((600-width)/2)+i,(int)((500-height)/2)+j, interval, interval);  }  }  } |

그림 14. 데이터베이스를 통한 그래프를 그리는 구조

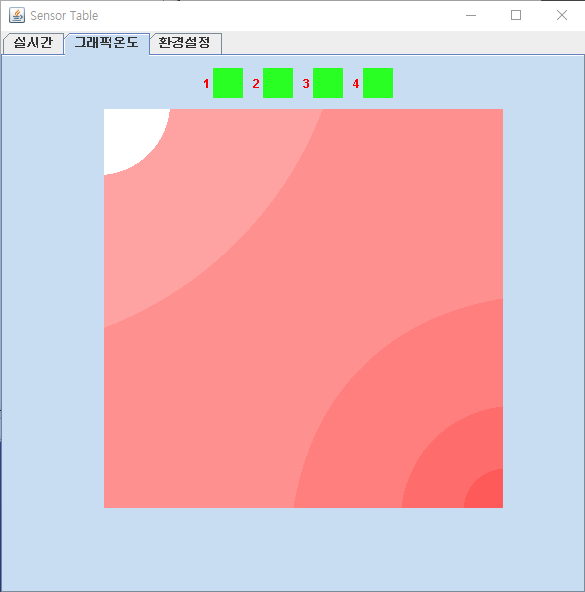


그림 15. 등고선 그래프 (Java)

그림 14는 등고선 그래프의 가로 길이, 세로 길이, 간격, 데이터베이스의 정보 등 다양한 변수들을 초기화 해주는 생성자가 있으며, 초기화 된 변수 들에 의해 데이터 베이스 질의를 통해 데이터를 가져온다. 가져온 데이터들은 수식을 통해 정밀하게 온도를 표현한다. 이때 이 공간 어디든 온도의 값을 정밀하게 알 수 있게 표현 되어 있는 것이 그림 15이다. 네 개의 센서의 상태가 정상을 나타내는 초록색으로 상태를 표시하고 있고, 각 지점의 온도 계산은 네 개의 센서 정보에서 계산된다.

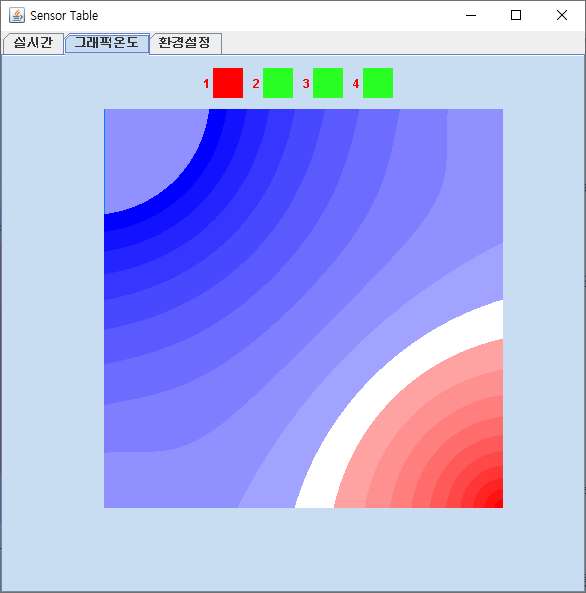


그림 16. 그래프 첫번째 센서 데이터를 수신 못하는 경우(Java)

그림 16은 1번 센서가 중지되어 있는 보습을 보이고 있다. 1번 센서가 동작하지 않기 때문에, 각 지점의 온도 계산은 1번을 제외한 2, 3, 4번 센서 정보만으로 계산된다.

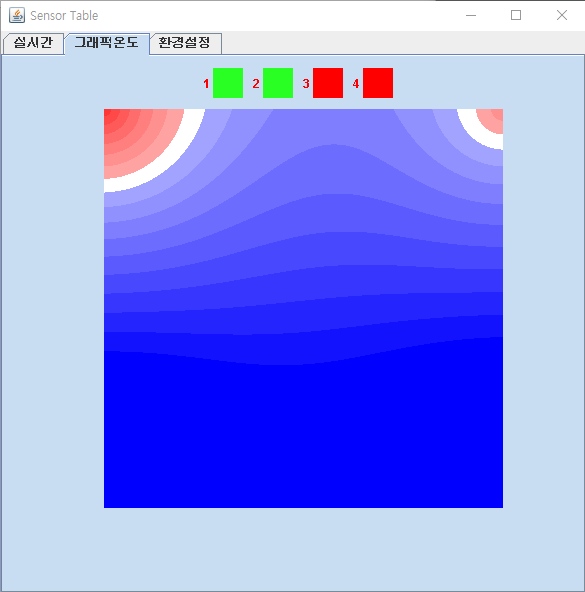


그림 17. 그래프 3, 4번 센서 데이터를 수신 못하는 경우 (Java)

센서가 동작하지 않는 경우, 5초 후 등고선그래프 탭(Tab)이 새로고침되며, 현재시간과 비교하여 10초가 지나도 불구하고 데이터를 수신하지 못한다면 그림 17과 같이 센서가 중지되었다고 판단하고, 나머지 데이터만을 가져와 등고선 그래프를 그리게 된다.

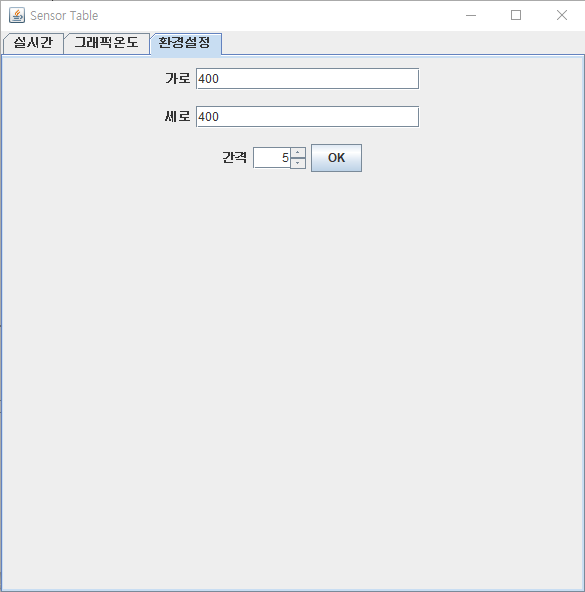


그림 18. 등고선 그래프 환경설정 (Java)

그림 18은 실제 환경에 적용할 때 정방형의 방구조 뿐만 아니라 다양한 형태의 공간에서 이용된다. 이를 위해 화면상에 표출되는 공간의 모양도 실제와 유사하게 설정하기 위해 공간의 가로 및 세로 길이를 조정이 필요하다. 단위는 Cm로 하여 설정이 가능하다. 뿐만 아니라 공간의 크기가 넓어질수록 계산양이 기하급수적으로 증가할 수 있기 때문에 등고선의 각 점을 표시하는 간격조절이 가능하도록 설정하였다.

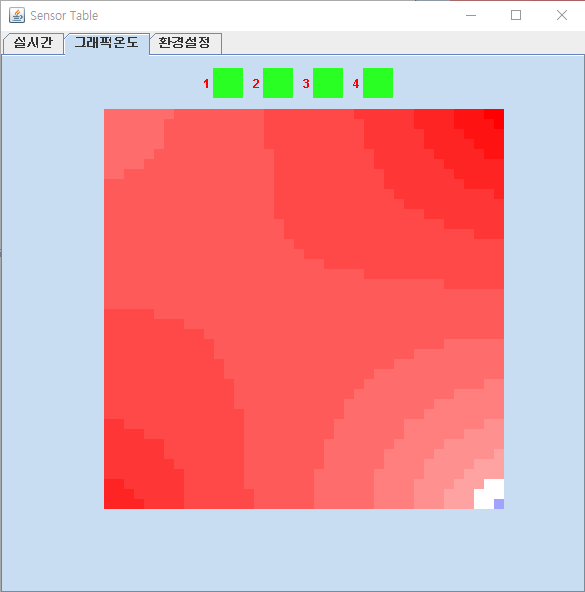


그림 19. 등고선 그래프 환경설정을 이용하여 간격을 조정했을 경우(Java)

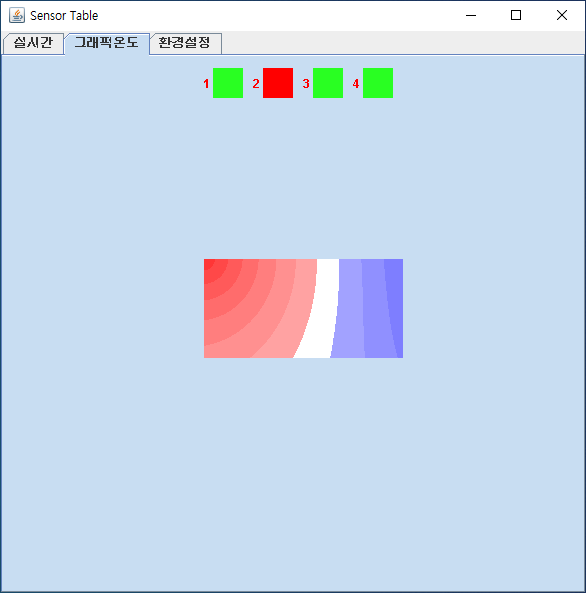


그림 20. 등고선 그래프 환경설정을 이용하여 가로 길이를 조정했을 경우(Java)

그림 19에선 환경설정 탭을 이용하여 간격을 조정했을 때의 그림이며, 그림 20도 마찬가지로 환경설정 탭을 이용해 가로길이, 세로길이를 조정했을 때의 그림이다. 그림에서 보는 바와 같이 표시 간격이 커지는 경우 화면의 정밀도는 떨어지지만, 시스템에서 계산하는 양이 제곱에 반비례하여 줄어드는 장점이 있다. 시스템에 부하를 가하지 않고 대략적인 온도 변화의 추세를 살펴볼 수 있다. 그림 20는 공간이 협소한 작은 공간으로 설정했을 때의 모습을 보이고 있다.

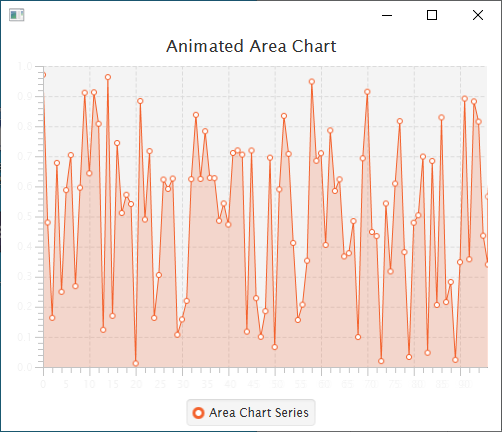
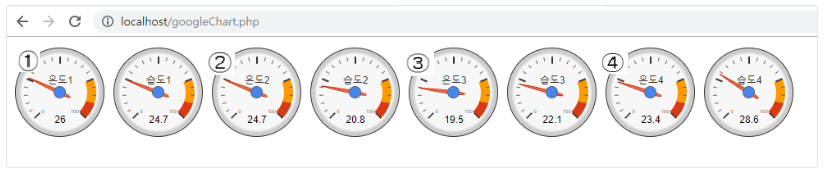


그림 21. 온습도 데이터를 통한 차트 애니메이션(Animation)

그림 21은 최근 대세가 될 가능성이 있는 자바 FX를 이용하여 실시간으로 받아오는 데이터를 차트로 보여 준 것이다. 자바 FX에는 다양한 종류의 차트들이 있다. 예를 들어 라인차트(LineChart), 버블차트(BubbleChart), 바차트(BarChart) 등이 있다. 그림 21의 차트 말고도 여러 차트를 이용한다면 실시간 데이터를 직관적으로 보여주거나 통계를 내는 등 다양한 형태로 사용이 가능하다. 그림 21은 직관적인 데이터 표현을 위해 실제 센서 정보를 활용하지 않고, 랜덤으로 값을 받은 모양을 출력하고 있다.

4.4 웹을 이용한 정보 확인

 그림 22. 온습도 게이지(Gauge)차트 홈페이지

데이터 베이스에 있는 온습도에 대한 최신 데이터 값을 가져온다. 온습도 데이터를 아파치 톰캣(Apache tomcat)을 이용하여 웹 상에 그림 22과 같이 자바스크립트나 JQuery 언어를 사용하여 게이지 차트 등 다양한 형태의 차트들로 직관적으로 표현 할 수 있다.

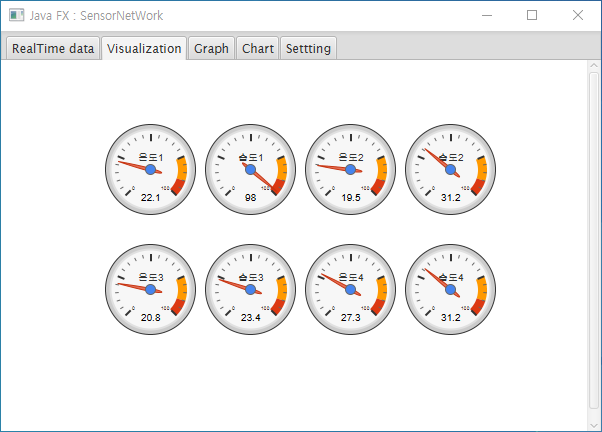


그림 23. 자바FX 온습도 데이터를 웹뷰로 보여주는 그림 (JavaFX)

최근 대세가 될 가능성이 높아 보이는 자바FX를 이용해 웹뷰(WebView)라는 컨트롤(Control)로 웹을 보여 줄 수 있다. 자바FX로 프로그래밍 한다면 실시간으로 받아오는 온습도 데이터를 통해 웹으로도 보여 줄 수 있다. 그림 22, 그림 23는 구글 차트의 게이지 차트를 이용하여 온습도 변화를 보여주고 있다.

# 5. 결론 및 향후 연구

IoT는 사용자에 따라 수많은 데이터를 받을 수 있고, 다양한 형태의 데이터들을 전달 받을 수 있다. 수많은 데이터들은 빅데이터를 통해 통계를 낼 수 도 있고, 여러 센서를 사용하거나 기발한 발상을 통해 인간을 좀 더 편리하게 만들 수 도 있다. 때론 아픈 사람을 위해 사람보다 더 빠르게 도움요청을 할 수 도 있을 것이다.

IoT는 인터넷을 통해 데이터를 수신하여 수많은 정보들을 데이터베이스를 이용해 웹이나, 핸드폰 등 다양한 형태의 스마트 기기들에게도 보여 줄 수 있다.

IoT 장비인 라즈베리 파이를 이용한 방의 온습도를 구하는 프로그램을 자바 뿐만 아니라 자바 FX로도 만들어 보았다. 이 프로그램은 사용자의 방 모습에 따라 환경설정을 이용하여 사용자의 요구사항에 맞게 변경할 수 있다. 또한 방의 온습도를 직관적으로 알기 쉽게 등고선 그래프나, 게이지 차트 등으로 나타내고 있다.

이 프로그램은 3차원 입체 공간 상 한 지점에 온습도를 표현하지 못하는 단점이 있다. 또한 데이터를 무한적으로 빠르게 받을 경우나, 컴퓨터를 사용하지 못 할 때 센서가 전부 동작하지 않거나 오작동을 할 때의 대비가 되어있지 않다. 특수한 상황일 경우에 문자를 통해 사용자에게 알려주는 것도 하나의 방법이다.

참고 문헌

[1] “새롭지만 전혀 새로울 것이 없는 IoT의 역사 - 브런치”, https://brunch.co.kr/ @neovis/4 2019-11

**[**2] “구글 무인 자동차 – 위키백과, 우리 모두의 백과사전”, [https://ko.wikipedia.org/ wiki/구글\_무인\_자동차](https://ko.wikipedia.org/%20wiki/구글_무인_자동차) 2019-11

[3] “라즈베리 파이 – 위키백과, 우리 모두의 백과사전”, [https://ko.wikipedia.org/ wiki/라즈베리\_파이](https://ko.wikipedia.org/%20wiki/라즈베리_파이) 2019-11

[4] “14가지 기발한 라즈베리 파이 프로젝트들 – ITWorl Korea”, http://www.itworld. co.kr/print/88471 2019-11

[5]”DTH11 아두이노 온도, 습도 센서 알아보기/아두이노코딩교육”, https:// m.blog.naver.com/PostView.nhn?blogId=roboholic84&logNo=221186233842&proxyReferer=https%3A%2F%2Fwww.google.com%2F 2019-11

[6]”고정밀 온습도센서(DHT22)와 아두이노로 온도 습도 측정하기”, http:// www.iamamaker.kr/ko/tutorials/%EC%98%A8%EC%8A%B5%EB%8F%84%EC%84%BC%EC%84%9C-dht22-%EC%95%84%EB%91%90%EC%9D%B4%EB%85%B8%EB%A1%9C-%EC%98%A8%EB%8F%84-%EC%8A%B5%EB%8F%84-%EC%B8%A1%EC%A0%95%ED%95%98%EA%B8%B0/ 2019-11

[7]”모든 센서 종류 리즈스 – 센서가 정말 많네요”, <https://smartiot.tistory.com/38> 2019-11