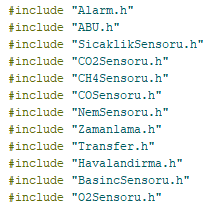
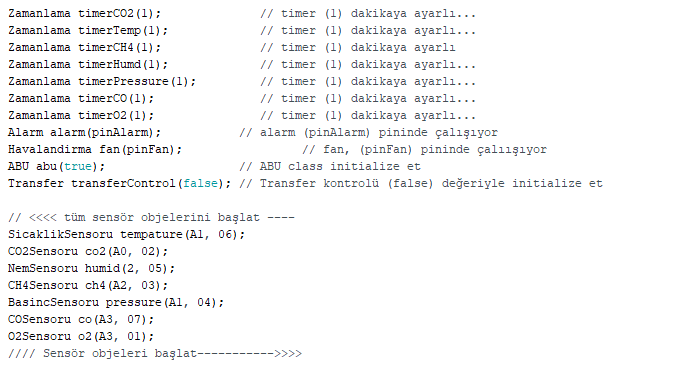
ABU: Bu birim ekran çıktısı üretmemektedir. Aşağıda yalnızca kodlar ve açıklamaları bulunmaktadır.

**MGS\_ABU\_Prototype.ino**: Arduino dili, C ve C++ dillerinden derlenmiş, birçok OOP özelliği içinde bulunduran yarı OOP bir dildir. “ino” dosyası main classı gibi çalışır.

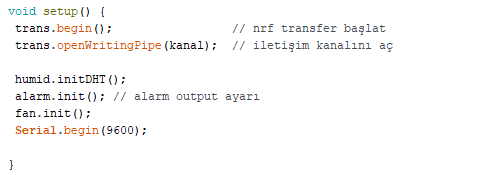
Öncelikle, tüm classlar header ve implementasyon(cpp) dosyası olarak oluşturulmuş ve header dosyaları, main classa include edilmiştir.

Şekil 1: ABU – Main – Tüm classları dahil etme

Şekil 2: ABU- Main Objeleri başlatma

Tanımlanmış sensör classlarından nesne yaratılmış,

Zamanlama classı sensör sayısı kadar örneklenmiştir. Ayrıca Alarm, fan ve transfer nesneleri yaratılmış, Manager objesi olarak görev yapacak olan ABU classından da bir obje üretilmiştir.



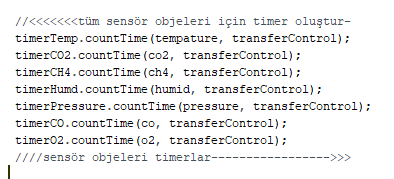
Şekil 3: ABU- Main setup

Arduino dilinde setup fonksiyonunun içinde, Arduino çalışmaya başladığında bir kere çalışacak kodlar bulunur. Burada, pinlerin fonksiyonları ayarlanır. init() fonksiyonlarında alarm ve fan pinleri output, DHT pini input olarak belirlenmiştir.

NRF11 transfer kütüphanesinin begin() ve openWritingPipe() fonksiyonları kablosuz iletişimi başlatmış, kullanılmaya hazır hale getirmiştir.



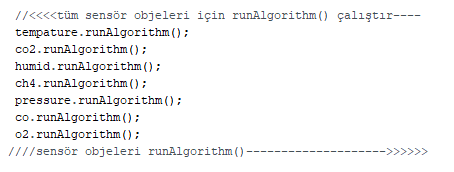
setup() fonksiyonunu çalıştırmayı bir kere bitirdikten sonra, cihaz yeniden başlatılana veya program sonlandırılana kadar loop() fonksiyonu bir sonsuz döngü olarak içindeki kodları çalıştırmaya başlar.

Şekil4: Abu- Main loop

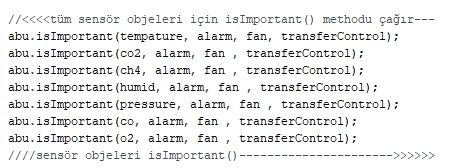
Oluşturduğumuz sensör nesneleri ile nesne sayısı kadar oluşturduğumuz timer nesnesini birleştirip, transferleri yönettiğimiz transferControl nesnesini ekleyerek timer classındaki countTime fonksiyonunu çağırdık. Bu fonksiyon, nesne oluşturulurken verilen dakika kadar sürede bir,

Şekil 5: Abu – Main Timer Başlatma

son aldığı değerleri transfer objesine taşır. Böylece, verilerde bir anomali olup olmamasına bakılmaksızın belirli aralıklarla sensörlerden okunan veriler veri tabanına yazılmak üzere RPI’a taşınır.

Sensörlerin, okudukları veriyi değerlendirip raw datayı, information haline getirdiği fonksiyondur. Sensörlerden okunan veriler birleştirilir, sensörün bağlamı içinde ilişkili hale getirilir.

Şekil 6: ABU- Main runAlgorithm() fonksiyonunu çağırma



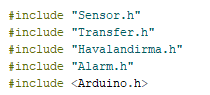
Şekil 7: ABU – Main sensör verilerini ayıklama

isImportant() fonksiyonu, runAlgorithm() fonksiyonundan aldığı information’ı, anlamlandırarak knowledge haline getirir ve yalnızca önemli knowledge’ı RPI’a iletilmek üzere transferControl objesine taşır.

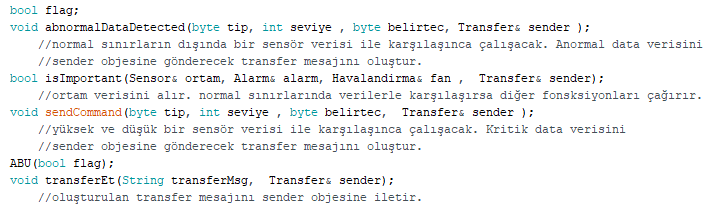
Burada yalnızca önemli verilerin(normalden düşük ölçüm, normalden yüksek ölçüm, anormal ölçüm) seçilerek iletilmesinin ve diğer verilerin belirli zaman aralıkları ile iletilmesinin sebebi cihazların ve iletim modülünün veri yoğunluğu yaşayarak karşılıklı iletişimde tıkanmasının önüne geçmektir. Yalnızca anlamlı bir wisdom üretmeye yardımcı olacak bilgiler RPI ve ABU modülleri arasında aktarılır.

**ABU.h**

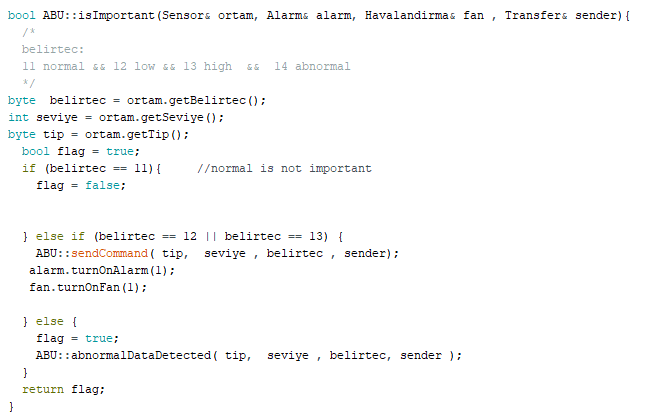
ABU classı, Arduino kodlarının çekirdeğini oluşturan, sensörlerden gelen tüm verilerin yönetildiği manager’dır.

Bu özelliğinden dolayı, yönetimi altında bulundurduğu tüm classların header dosyalarını, kendi header’ında içermektedir.

Şekil 8: ABU- Abu.h gerekli headerleri ekleme

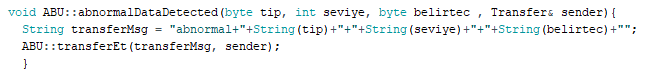


Şekil 9: Abu- abu.h ABU içindeki fonksiyonların titleları



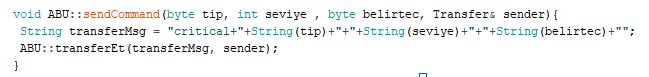
Şekil 10: ABU- ABU.cpp Sensörlerden gelen veri türüne göre işlem yapma

Main penceresinde her sensör objesi için çağırılan isImportant() fonksiyonu, anormal veya kritik seviyelerde bir nesneyle çağrılması halinde abnormalDataDetected() ve send Command() fonskiyonlarını çağırır.



Şekil 11: ABU- ABU.cpp anormal verileri işleme

Anormal bir data tespit edildiğinde çağırılan abnormalDataDetected() fonksiyonu, anormal data ile ilgili bir transfer mesajı oluşturarak doğrudan RPI’a transfer edilmek üzere gönderilmesi için transferEt() fonksiyonunu çağırır.



Şekil 12: ABU – Abu.cpp kritik verileri işleme

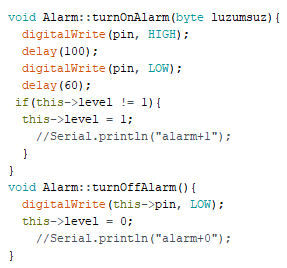
Kritik sınırları aşan bir data tespit edildiğinde çağırılan abnormalDataDetected() fonksiyonu, anormal data ile ilgili bir transfer mesajı oluşturarak doğrudan RPI’a transfer edilmek üzere gönderilmesi için transferEt() fonksiyonunu çağırır.

.

 Şekil 13- ABU- ABU.cpp verileri transfer etmek üzere iletme

transferEt() fonksiyonu, kendisine iletilen transfer mesajını Transfer classının transferEt() fonksiyonuna iletir.

Alarm.cpp

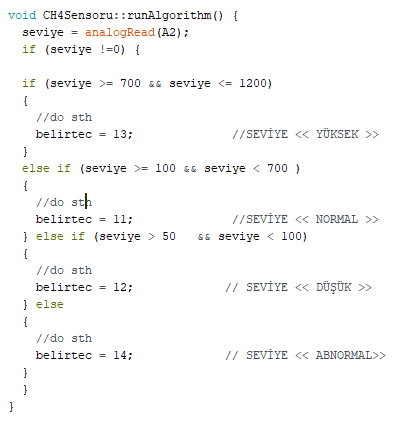
 Alarmın açılıp kapanması durumlarını ayarlayan fonksiyonlardır.

Alarm seviyesi nesnenin attribute’u olan level değişkeninde tutulur.

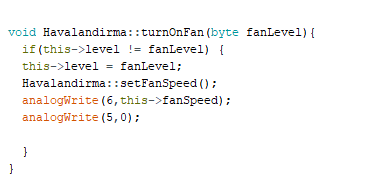
Ayrıca alarmın açık kapalı olması durumlarında, fiziksel olarak alarm pinine gidecek akımı ayarlar.

Şekil 14- ABU- Alarm.cpp Alarmı açma ve kapatma

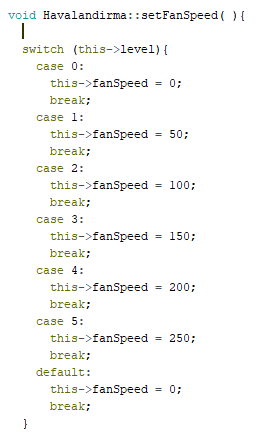
<Tip>Sensoru.cpp : Sensor classları, abstract Sensor.h classından türetilerek üretilmiştir. Her sensör için ayrı class kullanılmış olmasının nedeni, kod okunurluğunu artırmak, seviyelere daha kolay müdahale edebilmek, ileride yapılması planlanan ve yapılabilecek geliştirmelerde sensör bazında eklenen değişikliklerin diğer sensörlerde bir farklılığa yol açmadan doğrudan koda eklenebilmesi ve çalıştırılabilmesidir.

Burada CH4 sensörünün çalışması görülmektedir. Sensörden okunan veriye göre nesnenin belirteç attribute’unu; 11, 12, 13, 14 değerlerinden biri olarak ayarlar. Burada, belirteç değerinin byte olarak belirlenmiş olmasının sebebi, minimum bellek alanı kullanımı sağlamaktır. Arduino üzerinde tüm değişkenler, minimum bellek kullanımı sağlayacak şekilde kodlara indirgenmiştir.

Şekil 14 – ABU-CH4Sensoru.cpp Sensörden okunan veriyi yorumlama

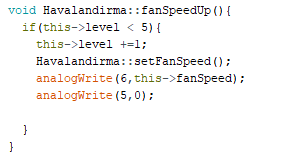
**Havalandırma.cpp**

Havalandırmayı parametre olarak verilen levelle başlatır.

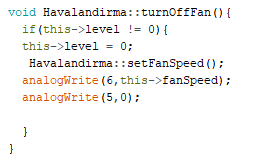
Şekil 15 – ABU – Havalandırma.cpp Havalandırmayı başlatma

0-5 arasında verilen havalandırma seviyesini fanın bağlı olduğu pine iletilecek akıma göre dönüştürür.

Şekil 16 – ABU Havalandırma.cpp Fan hızını ayarlama

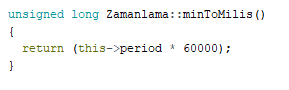


Fan leveli 5 değilken, fen levelini bir artırır ve setFanSpeed() fonksiyonunu çağırır.

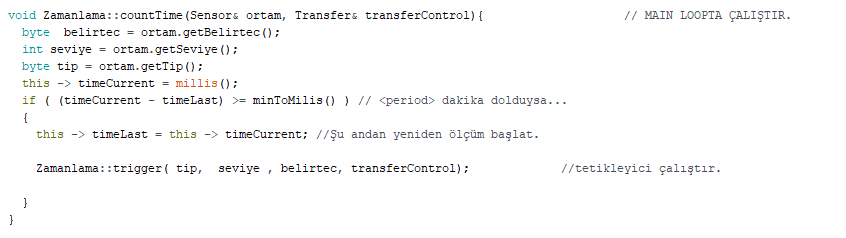
Şekil 17 – ABU- Havalandırma.cpp Havalandırma seviyesini yükseltme

Fanı kapatır, level ve fanSpeed’i 0 olarak güncelleyerek fanın çalışmasını fiziksel olarak durdurur.

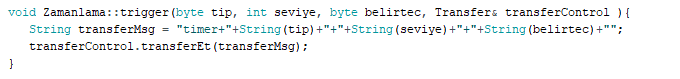
Şekil 18- ABU- Havalandırma.cpp Havalandırmayı kapatma



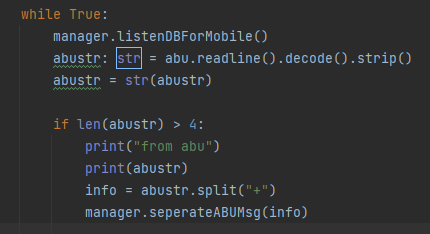
Şekil 19 – ABU – Zamanlama.cpp Dakika cinsinden girilen veriyi milisaniyeye çevirme



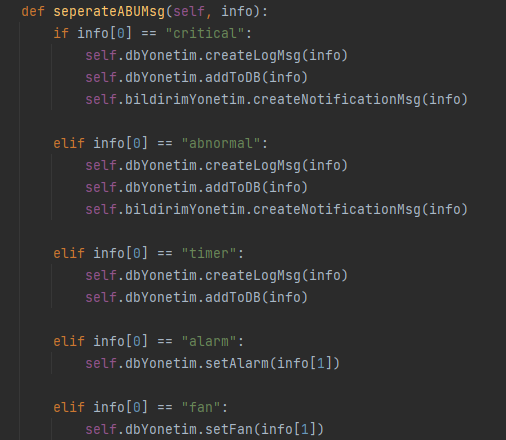
Şekil 20 – ABU – Zamanlama.cpp belirlenen periyodda zaman sayacıyla tetikleyiciyi çalıştırma



Şekil 21 – ABU – Zamanlama.cpp Tetiklendiğinde verileri transfer etmek üzere aktarma



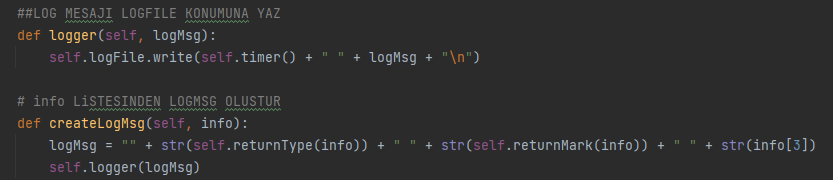
Şekil 22- RPI- Main.py Veritabanının ve ABU’nun dinlenip eylemlerin planlanması



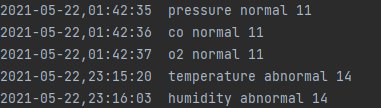
Şekil 23 – RPI – Manager.py Abu’dan gelen string mesajın çözümlenmesi

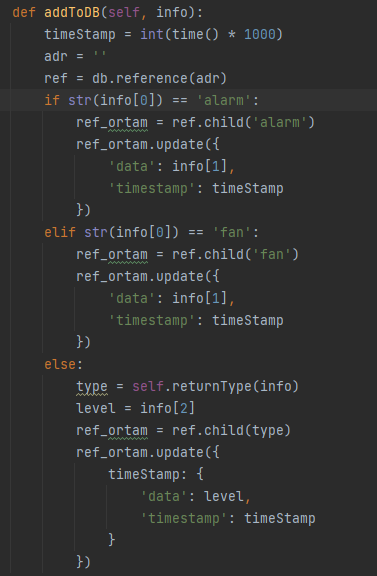


Şekil 24 – RPI – Manager.py Veritabanının ilgili tablolarını değişiklikler için dinleyecek fonksiyonun çağrılması



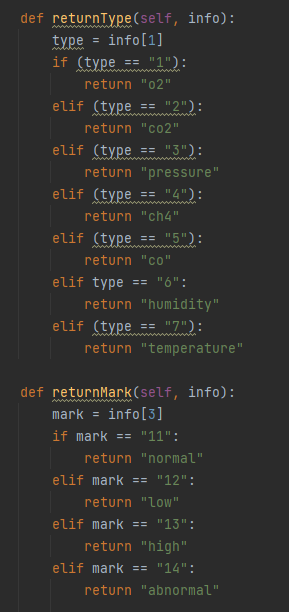
Şekil 25 – RPI – DBYonetim.py Arduino’dan gelen mesajların log dosyasına kaydı



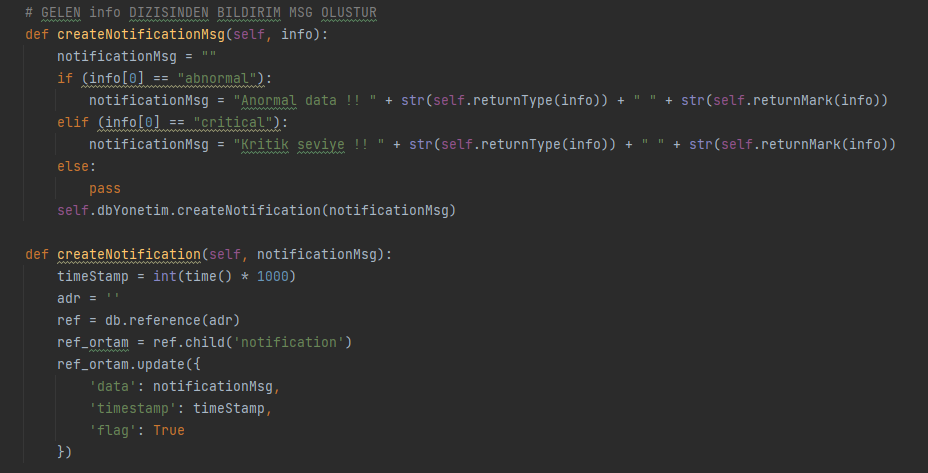
Şekil 26 – RPI – logfile.txt log dosyası örneği

ABU’dan gelen bilgiler doğrultusunda veritabanına aktarılacak veriler alarm, fan ve ortam koşulları olarak gruplanıp veritabanına anlık timestamp verisi ile birlikte yazılır.

Şekil 27 - RPI – DBYonetim.py Verilerin veritabanına kaydı



ABU’da byte formatında kodlanmış veriler RPI üzerinde işlem görmeden, veritabanına ve log dosyalarına aktarılmadan önce çözümlenir.

Şekil 28 - RPI – DBYonetim.py ABU’dan gelen verilerin çözümlenmesi

Şekil 29 - RPI – BildirimYonetim.py ABU’dan gelen kritik verilerin bildirim olarak hazırlanıp gönderilmesi