

## SKRIPSI

### PENGEMBANGAN APLIKASI PEMANTAUAN GETARAN GEDUNG MENGGUNAKAN WIRELESS SENSOR NETWORK



Andrianto Chandra

NPM: 2016730017

PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS  
UNIVERSITAS KATOLIK PARAHYANGAN  
2021



**UNDERGRADUATE THESIS**

**DEVELOPMENT OF BUILDING VIBRATION MONITORING  
APPLICATIONS USING WIRELESS SENSOR NETWORK**



**Andrianto Chandra**

**NPM: 2016730017**

**DEPARTMENT OF INFORMATICS  
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES  
PARAHYANGAN CATHOLIC UNIVERSITY  
2021**



## **LEMBAR PENGESAHAN**

### **PENGEMBANGAN APLIKASI PEMANTAUAN GETARAN GEDUNG MENGGUNAKAN WIRELESS SENSOR NETWORK**

**Andrianto Chandra**

**NPM: 2016730017**

**Bandung, 12 Juli 2021**

**Menyetuju,**

**Pembimbing**

**Elisati Hulu, M.T.**

**Ketua Tim Penguji**

**Anggota Tim Penguji**

**Chandra Wijaya, M.T.**

**Rosa De Lima, M.T.**

**Mengetahui,**

**Ketua Program Studi**

**Mariskha Tri Adithia, P.D.Eng**



## **PERNYATAAN**

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

### **PENGEMBANGAN APLIKASI PEMANTAUAN GETARAN GEDUNG MENGGUNAKAN WIRELESS SENSOR NETWORK**

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,  
Tanggal 12 Juli 2021



Andrianto Chandra  
NPM: 2016730017



## ABSTRAK

Kesehatan sebuah bangunan/gedung merupakan salah satu persyaratan teknis yang harus ada saat mengurus IMB. Salah satu parameter dalam pemantauan kesehatan sebuah gedung atau bangunan adalah getaran. Seiring berkembangnya teknologi, getaran pada suatu bangunan dapat dilihat untuk memberikan kemudahan dalam melakukan pengukuran data agar menjadi lebih efektif.

Getaran pada sebuah gedung dapat ditangkap dengan menggunakan sensor *Accelerometer* yang disebar di sisi gedung yang akan membentuk sebuah jaringan sensor nirkabel atau istilah lainnya *Wireless Sensor Network*. *Wireless Sensor Network* (WSN) adalah kumpulan sejumlah node yang diatur dalam sebuah jaringan. Masing-masing node dalam jaringan sensor nirkabel biasanya dilengkapi dengan radio transceiver, mikrokontroler, dan sumber energi seperti baterai.

Pada skripsi ini, akan dibuat sebuah perangkat lunak yang dapat menampilkan hasil pantauan getaran sebuah gedung yang ditampilkan dalam bentuk *graph* dengan menggunakan WSN. Dengan menggunakan perangkat lunak tersebut, dapat diketahui seberapa besar getaran yang terjadi pada sebuah gedung sehingga dapat mengetahui besar kecilnya getaran yang dihasilkan tersebut.

Hasil dari pengujian menunjukkan bahwa aplikasi monitoring getaran gedung berhasil dibangun di WSN. Hasil monitoring bergantung pada letak sensor melakukan *sensing* dan tidak tergantung dengan topologi WSN yang digunakan.

**Kata-kata kunci:** gedung, getaran, WSN, accelerometer, nirkabel



## ABSTRACT

The health of a building is one of the technical requirements that must be in place when applying for an IMB. One of the parameters in monitoring a building or building is vibration. As technology develops, vibrations in a building can be seen to provide convenience in measuring data to be more effective.

Vibrations in a building can be captured using *Accelerometer* sensors which are deployed on the side of the building which will form a wireless sensor network or another term *Wireless Sensor Network*. *Wireless Sensor Network* (WSN) is a collection of nodes. Each node in a wireless sensor network is usually equipped with a radio transceiver, microcontroller, and an energy source such as a battery.

In this thesis, a software will be created that can display the results of vibration monitoring of a building which is displayed in the form of *graph* by using WSN. By using the software, it can be seen how much vibration occurs in a building so that it can determine the size of the vibration generated.

The results of the test show that the building vibration monitoring application has been successfully built in WSN. Monitoring results depend on the location of the sensor doing *sensing* and does not depend on the WSN topology used.

**Keywords:** building, vibration, WSN, accelerometer, wireless



*Dipersembahkan kepada Tuhan, keluarga, saudara, dan teman-teman yang telah mendukung*



## KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas karunia-Nya, penulis dapat menyelesaikan penyusunan skripsi yang berjudul "Pengembangan Aplikasi Pemantauan Getaran Gedung Menggunakan WSN". Selama penyusunan skripsi ini, penulis menghadapi banyak kendala dan berbagai masalah. Penulis menyadari bahwa penyusunan skripsi ini juga tidak terlepas dari bantuan berbagai pihak, baik langsung maupun tidak langsung. Secara khusus, penulis ingin berterima kasih kepada:

1. Tuhan Yesus atas Anugrah, Berkat, dan Rahmat-Nya.
2. Keluarga yang selalu memberikan dukungan kepada penulis baik berupa doa atau dukungan mental serta materiil.
3. Bapak Elisati Hulu, M.T. selaku dosen pembimbing yang telah membimbing penulis dan memberikan dukungan maupun bantuan kepada penulis dalam proses penyusunan skripsi ini.
4. Bapak Chandra Wijaya, M.T. dan Ibu Rosa De Lima, M.Kom selaku dosen penguji yang telah memberikan kritik dan saran yang membangun sehingga penelitian ini menjadi lebih baik.
5. Teman-teman sejurusan Teknik Informatika UNPAR angkatan 2016 dan teman - teman di luar perkuliahan yang telah menemani penulis dalam menyelesaikan perkuliahan dari awal semester sampai akhir semester.
6. Teman seperjuangan skripsi yang berdosen pembimbing sama dengan penulis, bimbingan bersama, saling membantu, dan saling memberikan dukungan satu sama lain selama menyusun skripsi ini.

Penulis menyadari bahwa penelitian ini masih jauh dari kata sempurna. Oleh karena itu, penulis memohon maaf jika terdapat kesalahan. Penulis juga mengharapkan kritik dan saran yang membangun untuk menyempurnakan penelitian ini. Semoga penelitian ini dapat memberi informasi yang bermanfaat dan menjadi inspirasi untuk penelitian-penelitian berikutnya.

Bandung, Juli 2021

Penulis



# DAFTAR ISI

<b>KATA PENGANTAR</b>	<b>xv</b>
<b>DAFTAR ISI</b>	<b>xvii</b>
<b>DAFTAR GAMBAR</b>	<b>xxi</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	2
1.3 Tujuan . . . . .	2
1.4 Batasan Masalah . . . . .	2
1.5 Metodologi . . . . .	2
1.6 Sistematika Pembahasan . . . . .	2
<b>2 LANDASAN TEORI</b>	<b>5</b>
2.1 Gedung [1] . . . . .	5
2.1.1 Klasifikasi Gedung . . . . .	5
2.1.2 Pemantauan Kesehatan Struktural [2] . . . . .	6
2.2 Getaran [3] . . . . .	8
2.3 <i>Wireless Sensor Network</i> [4] . . . . .	9
2.3.1 Penerapan Wireless Sensor Network di berbagai bidang . . . . .	9
2.3.2 Struktur Node Sensor . . . . .	10
2.3.3 Arsitektur <i>Wireless Sensor Network</i> . . . . .	11
2.3.4 Topologi <i>Wireless Sensor Network</i> . . . . .	13
2.3.5 Protokol <i>Wireless Sensor Network</i> . . . . .	16
2.4 Accelerometer [5] . . . . .	18
2.5 Fourier Transform . . . . .	18
2.5.1 Discrete Fourier Transform (DFT) . . . . .	19
2.5.2 Fast Fourier Transform (FFT) . . . . .	19
2.6 PreonVM . . . . .	20
2.6.1 Fitur PreonVM . . . . .	20
2.6.2 Kelebihan PreonVM . . . . .	21
2.6.3 Class Library PreonVM . . . . .	21
2.6.4 Preon32 . . . . .	22
2.6.5 Spesifikasi Sensor-Sensor Preon32 . . . . .	23
<b>3 ANALISIS</b>	<b>25</b>
3.1 Analisis Aplikasi Pemantauan Getaran Gedung Berbasis <i>Wireless Sensor Network</i>	25
3.1.1 Analisis Arsitektur dan Topologi <i>Wireless Sensor Network</i> . . . . .	25
3.1.2 Analisis Accelerometer . . . . .	26
3.1.3 Analisis Fungsi Aplikasi . . . . .	26
3.1.4 Analisis Kelas . . . . .	30
3.1.5 Analisis Paket/Pesan . . . . .	33

<b>4 PERANCANGAN</b>	<b>35</b>
4.1 Perancangan Interaksi Antar Node . . . . .	35
4.1.1 Diagram Sequence "Check Online Node Status" . . . . .	35
4.1.2 Diagram Sequence "Start Sensing" . . . . .	36
4.1.3 Diagram Sequence "Stop Sensing" . . . . .	37
4.1.4 Diagram Sequence "Exit Program" . . . . .	37
4.2 Perancangan Kelas Aplikasi . . . . .	38
4.2.1 Package SensorNode . . . . .	38
4.2.2 Package BaseStation . . . . .	40
4.2.3 Package Controller . . . . .	42
4.2.4 Kelas DataFrequency . . . . .	45
4.2.5 Kelas RenderChart . . . . .	46
4.2.6 Kelas StartChart . . . . .	47
4.2.7 Kelas Chart . . . . .	48
4.2.8 Kelas ChartAmplitude . . . . .	49
4.2.9 Kelas ChartFrequency . . . . .	50
4.3 Perancangan Input dan Output . . . . .	52
4.4 Perancangan Antar Muka Untuk Visualisasi . . . . .	53
4.5 Perancangan Pseudocode Aplikasi . . . . .	54
4.5.1 Start Chart . . . . .	54
4.5.2 ChartAmplitude & ChartFrequency . . . . .	55
4.5.3 Controller . . . . .	57
<b>5 IMPLEMENTASI DAN PENGUJIAN</b>	<b>59</b>
5.1 Implementasi . . . . .	59
5.1.1 Lingkungan Implementasi . . . . .	59
5.1.2 Hasil Implementasi Kelas . . . . .	59
5.1.3 Hasil Implementasi Antar Muka Visualisasi Hasil <i>Sense</i> . . . . .	64
5.2 Pengujian . . . . .	66
5.2.1 Pengujian Fungsional . . . . .	67
5.2.2 Pengujian Eksperimental . . . . .	71
5.2.3 Kesimpulan Hasil Pengujian . . . . .	77
5.2.4 Masalah dalam pengujian . . . . .	77
<b>6 KESIMPULAN DAN SARAN</b>	<b>79</b>
6.1 Kesimpulan . . . . .	79
6.2 Saran . . . . .	79
<b>DAFTAR REFERENSI</b>	<b>81</b>
<b>A KODE PROGRAM</b>	<b>83</b>
<b>B HASIL EKSPERIMEN DI MALL PASKAL 23 SQUARE</b>	<b>97</b>
B.1 Topologi Star . . . . .	98
B.1.1 SensorA . . . . .	98
B.1.2 SensorB . . . . .	99
B.1.3 SensorC . . . . .	100
B.1.4 SensorD . . . . .	101
B.1.5 SensorE . . . . .	102
B.2 Topologi Tree . . . . .	103
B.2.1 SensorA . . . . .	103
B.2.2 SensorB . . . . .	104

B.2.3	SensorC . . . . .	105
B.2.4	SensorD . . . . .	106
B.2.5	SensorE . . . . .	107
<b>C</b>	<b>HASIL EKSPERIMEN DI ROOFTOP GEDUNG 10 UNPAR</b>	<b>109</b>
C.1	Topologi Star . . . . .	110
C.1.1	SensorA . . . . .	110
C.1.2	SensorB . . . . .	111
C.1.3	SensorC . . . . .	112
C.1.4	SensorD . . . . .	113
C.1.5	SensorE . . . . .	114
C.2	Topologi Tree . . . . .	115
C.2.1	SensorA . . . . .	115
C.2.2	SensorB . . . . .	116
C.2.3	SensorC . . . . .	117
C.2.4	SensorD . . . . .	118
C.2.5	SensorE . . . . .	119



## DAFTAR GAMBAR

2.1 Ilustrasi penerapan <i>Wireless Sensor Network</i> . . . . .	10
2.2 Struktur Node Sensor . . . . .	11
2.3 Ilustrasi penerapan <i>Wireless Sensor Network</i> . . . . .	12
2.4 Arsitektur <i>flat</i> pada <i>Wireless Sensor Network</i> . . . . .	12
2.5 Arsitektur <i>single hop</i> pada <i>Wireless Sensor Network</i> . . . . .	13
2.6 Arsitektur <i>multi hop</i> pada <i>Wireless Sensor Network</i> . . . . .	13
2.7 Topologi <i>point-to-point</i> . . . . .	14
2.8 Topologi <i>bus</i> . . . . .	14
2.9 Topologi <i>tree</i> . . . . .	15
2.10 Topologi <i>star</i> . . . . .	15
2.11 Topologi <i>ring</i> . . . . .	15
2.12 Topologi <i>partially connected mesh</i> . . . . .	16
2.13 Topologi <i>fully conneted mesh</i> . . . . .	16
2.14 Layer pada <i>Wireless Sensor Network</i> . . . . .	17
2.15 Kategori <i>Fourier Transform</i> dan contoh bentuk sinyal . . . . .	18
2.16 Contoh visual <i>Decimation In Time</i> dengan panjang 8 . . . . .	20
2.17 <i>Virtual Machine</i> yang membuat aplikasi dapat dijalankan secara independen . . . . .	21
2.18 Preon32 Board . . . . .	23
3.1 <i>Wireless Sensor Network</i> dengan Topologi <i>Star</i> . . . . .	26
3.2 Arsitektur dan Topologi WSN . . . . .	26
3.3 Diagram Use Case Aplikasi . . . . .	28
3.4 Diagram Kelas NodeSensor . . . . .	31
3.5 Diagram Kelas BaseStation . . . . .	32
3.6 Diagram Kelas Tester . . . . .	32
4.1 Diagram Sequence "Check Online Node" . . . . .	35
4.2 Diagram Sequence "Sense" . . . . .	36
4.3 Diagram Sequence "Stop Sensing" . . . . .	37
4.4 Diagram Sequence "Exit" . . . . .	37
4.5 Kelas Diagram lengkap package SensorNode . . . . .	38
4.6 Perancangan Kelas Accelerometer . . . . .	39
4.7 Perancangan Kelas SensorManager . . . . .	39
4.8 Kelas Diagram lengkap package BaseStation . . . . .	41
4.9 Perancangan Kelas BaseStation . . . . .	41
4.10 Kelas Diagram lengkap package Controller . . . . .	43
4.11 Perancangan Kelas Complex . . . . .	44
4.12 Perancangan Kelas DataFrequency . . . . .	45
4.13 Perancangan Kelas FFT . . . . .	46
4.14 Perancangan Kelas RenderChart . . . . .	46
4.15 Perancangan Kelas StartChart . . . . .	47
4.16 Perancangan Kelas Chart . . . . .	48
4.17 Perancangan Kelas ChartAmplitude . . . . .	49

4.18 Perancangan Kelas ChartFrequency . . . . .	50
4.19 Perancangan Kelas SampleData . . . . .	51
4.20 Perancangan Kelas Sensor . . . . .	51
4.21 Rancangan antar muka tampilan grafik amplitudo dan frekuensi sensor . . . . .	53
5.1 Tampilan awal aplikasi . . . . .	65
5.2 Grafik hasil implementasi visualisasi hasil <i>sense</i> . . . . .	66
5.3 Tampilan awal aplikasi . . . . .	67
5.4 Pengujian "Check Node Online Status" . . . . .	68
5.5 Pengujian "Start sensing" . . . . .	69
5.6 Pengujian input selain angka '3' saat "Start sensing" . . . . .	70
5.7 Pengujian "Stop sensing" . . . . .	70
5.8 Pengujian "Exit Program" . . . . .	71
5.9 Topologi star yang digunakan ( <i>Single hop</i> ) . . . . .	71
5.10 Topologi tree yang digunakan . . . . .	72
5.11 Denah peletakan sensor node dengan topologi star . . . . .	72
5.12 Denah peletakan sensor node dengan topologi tree . . . . .	73
5.13 Denah peletakan sensor node dengan topologi tree . . . . .	75
5.14 Denah peletakan sensor node dengan topologi tree . . . . .	75
B.1 Grafik Monitoring getaran SensorA dengan topologi star . . . . .	98
B.2 Grafik Monitoring getaran SensorB dengan topologi star . . . . .	99
B.3 Grafik Monitoring getaran SensorC dengan topologi star . . . . .	100
B.4 Grafik Monitoring getaran SensorD dengan topologi star . . . . .	101
B.5 Grafik Monitoring getaran SensorE dengan topologi star . . . . .	102
B.6 Grafik Monitoring getaran SensorA dengan topologi tree . . . . .	103
B.7 Grafik Monitoring getaran SensorB dengan topologi tree . . . . .	104
B.8 Grafik Monitoring getaran SensorC dengan topologi tree . . . . .	105
B.9 Grafik Monitoring getaran SensorD dengan topologi tree . . . . .	106
B.10 Grafik Monitoring getaran Sensor E dengan topologi tree . . . . .	107
C.1 Grafik Monitoring getaran SensorA dengan topologi star . . . . .	110
C.2 Grafik Monitoring getaran SensorB dengan topologi star . . . . .	111
C.3 Grafik Monitoring getaran SensorC dengan topologi star . . . . .	112
C.4 Grafik Monitoring getaran SensorD dengan topologi star . . . . .	113
C.5 Grafik Monitoring getaran SensorE dengan topologi star . . . . .	114
C.6 Grafik Monitoring getaran SensorA dengan topologi tree . . . . .	115
C.7 Grafik Monitoring getaran SensorB dengan topologi tree . . . . .	116
C.8 Grafik Monitoring getaran SensorC dengan topologi tree . . . . .	117
C.9 Grafik Monitoring getaran SensorD dengan topologi tree . . . . .	118
C.10 Grafik Monitoring getaran SensorE dengan topologi tree . . . . .	119

# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Kesehatan sebuah bangunan/gedung merupakan salah satu persyaratan teknis yang harus ada saat mengurus IMB. Izin Mendirikan Bangunan (IMB) adalah sebuah perizinan yang diberikan oleh Kepala Daerah kepada pemilik bangunan untuk membangun baru, mengubah, memperluas, mengurangi, dan/atau merawat bangunan sesuai dengan persyaratan administratif dan persyaratan teknis yang berlaku. IMB ini sangat penting khususnya untuk bangunan atau gedung yang bertingkat seperti gedung-gedung yang ada di Universitas Katolik Parahyangan. Salah satu gedung yang ada di Universitas Katolik Parahyangan adalah gedung 10 yang termasuk juga area *rooftop*.

Salah satu parameter dalam pemantauan kesehatan sebuah gedung atau bangunan adalah getaran. Getaran yang terjadi pada sebuah bangunan dapat dipengaruhi oleh dua faktor. Faktor yang pertama adalah faktor dari bumi dan faktor kedua adalah faktor dari dalam gedung itu sendiri. Getaran merupakan salah satu faktor penyebab gempa bumi dimana terjadi pada kerak bumi sebagai gejala aktivitas tektonis maupun vulkanis. Pada umumnya getaran ini diakibatkan oleh adanya pergeseran lempeng pada permukaan bumi sehingga dapat terjadi gelombang gempa bumi. Getaran yang berasal dari gedung itu sendiri dapat dicontohkan dengan adanya mesin bertenaga besar yang terdapat dalam gedung tersebut seperti lift. Mesin dengan tenaga yang besar ini perlahan-lahan dapat menyebabkan sebuah gedung merasakan sebuah getaran yang lama kelamaan membuat gedung ini menjadi tidak stabil dan membuat kesehatan gedung menjadi memburuk.

Seiring berkembangnya teknologi, getaran pada suatu bangunan dapat dilihat untuk memberikan kemudahan dalam melakukan pengukuran data agar menjadi lebih efektif. Proses ini digunakan sebagai pengamatan, perekaman, dan pengevaluasian dalam hal ini adalah getaran pada sebuah bangunan atau gedung untuk menilai kesehatan secara berkelanjutan. Getaran pada sebuah gedung dapat ditangkap dengan menggunakan sensor *Accelerometer* yang disebar di sisi gedung yang membentuk sebuah jaringan sensor nirkabel atau istilah lainnya *Wireless Sensor Network*.

*Wireless Sensor Network* (WSN) adalah kumpulan sejumlah node yang diatur dalam sebuah jaringan. Masing-masing node dalam jaringan sensor nirkabel biasanya dilengkapi dengan radio transceiver, mikrokontroler, dan sumber energi seperti baterai. Banyak aplikasi yang bisa dilakukan menggunakan jaringan sensor nirkabel, misalnya pengumpulan data kondisi lingkungan, *security monitoring*, dan *node tracking scenarios*. Penggunaan WSN dapat menjadi alternatif untuk melakukan pemantauan getaran sebuah gedung. Dengan adanya WSN, pemantauan getaran pada sebuah gedung dapat dilakukan dengan lebih mudah sehingga dapat memungkinkan pengguna mendapatkan informasi seperti amplitudo secara *real time* dengan tingkat akurasi yang tinggi.

Pada skripsi ini, akan dibuat sebuah perangkat lunak yang dapat menampilkan hasil pantauan getaran sebuah gedung yang ditampilkan dalam bentuk *graph* dengan menggunakan WSN. Dengan menggunakan perangkat lunak tersebut, dapat diketahui seberapa besar getaran yang terjadi pada sebuah gedung sehingga dapat mengetahui besar kecilnya getaran yang dihasilkan tersebut. Nilai yang didapatkan adalah amplitudo dan frekuensi. Hasil dari kedua nilai ini akan mempengaruhi apakah sebuah gedung itu masih dalam keadaan sehat ataupun tidak.

## 1.2 Rumusan Masalah

Masalah-masalah yang ingin diselesaikan dalam skripsi ini adalah sebagai berikut:

- Bagaimana sensor *Accelerometer* bekerja?
- Bagaimana *Wireless Sensor Network* bekerja?
- Bagaimana membangun aplikasi pemantauan getaran gedung dengan menggunakan jaringan *wireless sensor*?

## 1.3 Tujuan

Tujuan-tujuan dari pembuatan perangkat lunak adalah sebagai berikut:

- Mempelajari cara kerja sensor *accelerometer*.
- Mempelajari cara kerja *Wireless Sensor Network*.
- Membangun aplikasi pemantauan getaran gedung menggunakan *Wireless Sensor Network* (WSN).

## 1.4 Batasan Masalah

Penelitian ini dibuat berdasarkan batasan-batasan sebagai berikut:

- Sensor yang digunakan sebagai penelitian hanya sensor untuk mengukur getaran.
- Sensor digunakan untuk mengukur getaran pada gedung-gedung yang ada di Universitas Katolik Parahyangan dan Mall Paskal 23 Square.
- Topologi yang akan digunakan dalam penelitian ini adalah topologi *star* dan topologi *tree*
- Fokus utama penelitian ini adalah membangun aplikasi yang menangkap hasil *sensing* pada sensor *Accelerometer*

## 1.5 Metodologi

Berikut adalah metode penelitian yang digunakan dalam penelitian ini:

- Melakukan studi literatur mengenai *Wireless Sensor Network*.
- Mempelajari cara kerja sensor *Accelerometer*.
- Mempelajari sistem *Structural Health Monitoring* (SHM).
- Mempelajari pemrograman pada Wireless Sensor Network dengan Bahasa Pemograman JAVA.
- Membangun infrastruktur jaringan *Wireless Sensor Network* (WSN).
- Mengimplementasi kode program pada sensor *accelerometer*.
- Melakukan pengujian aplikasi pemantauan di sekitar gedung yang ada di Universitas Katolik Parahyangan

## 1.6 Sistematika Pembahasan

Setiap bab dalam penelitian ini memiliki sistematika penulisan yang dijelaskan sebagai berikut:

Bab 1 Pendahuluan, yaitu membahas mengenai gambaran umum penelitian yang dilakukan ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika pembahasan.

Bab 2 Dasar Teori, yaitu membahas teori-teori yang mendukung berjalannya penelitian ini.

Bab 3 Analisis, yaitu membahas mengenai analisis dari aplikasi yang akan dibuat seperti fungsi-fungsi apa saja yang terdapat dalam aplikasi yang akan dibuat.

Bab 4 Perancangan, yaitu membahas diagram kelas dari aplikasi yang akan dibuat, membahas juga beberapa fungsi-fungsi penting yang akan digunakan dalam pembuatan aplikasi ini.

Bab 5 Implementasi dan Pengujian, yaitu membahas implementasi dari hasil rancangan dan pengujian dari aplikasi WSN yang telah dibuat dan pengujian aplikasi yang telah dibuat.

Bab 6 Kesimpulan, yaitu membahas kesimpulan dari hasil pengujian dan saran untuk pengembangan selanjutnya.



## BAB 2

### LANDASAN TEORI

#### 2.1 Gedung [1]

Berdasarkan KBBI(Kamus Besar Bahasa Indonesia), gedung dapat diartikan sebagai bangunan tembok dan sebagainya yang berukuran besar sebagai tempat kegiatan, seperti perkantoran, pertemuan, perniagaan, pertunjukan, olahraga, dan sebagainya. Secara umum, gedung didefinisikan sebagai sebuah struktur buatan manusia yang terdiri atas dinding dan atap yang didirikan secara permanen di suatu tempat. Gedung juga memiliki beragam bentuk, ukuran, dan fungsi, serta telah mengalami penyesuaian sepanjang sejarah yang disebabkan oleh beberapa faktor, seperti bahan bangunan, kondisi cuaca, harga, kondisi tanah, dan alasan estetika.

##### 2.1.1 Klasifikasi Gedung

- Berdasarkan tingkat kompleksitas

- Gedung Sederhana

Bangunan sederhana merupakan bangunan gedung dengan karakter sederhana, serta memiliki kompleksitas dan teknologi yang sederhana. Bangunan gedung sederhana meliputi gedung kantor dengan jumlah s.d. 2 lantai dengan luas maksimal mencapai  $500\text{m}^2$ .

- Gedung Tidak Sederhana

Bangunan gedung tidak sederhana merupakan bangunan gedung yang memiliki karakter, kompleksitas dan teknologi yang tidak sederhana. Bangunan gedung tidak sederhana. Bangunan gedung tidak sederhana meliputi gedung kantor bertingkat lebih dari 2 lantai yang memiliki luas di atas  $500\text{m}^2$ .

- Gedung Khusus

Bangunan gedung khusus merupakan bangunan gedung yang digunakan untuk kepentingan khusus, yang mempunyai tingkat kerahasiaan tinggi atau yang penyelenggaranya dapat membahayakan lingkungan sekitar. Bangunan gedung ini memiliki kompleksitas tertentu, oleh karena itu dalam pembangunan atau pemanfaatannya membutuhkan pengelolaan dan persyaratan khusus. Bangunan gedung khusus meliputi gedung istana negara, gedung laboratorium, bangunan gedung reaktor nuklir, instalasi pertahanan dan keamanan, dan bangunan gedung sejenisnya yang ditetapkan oleh menteri.

- Berdasarkan Fungsinya

- Gedung Rumah Tinggal

Pembuatan gedung rumah tinggal bertujuan untuk memenuhi kebutuhan manusia akan tempat tinggal. Pembuatan rumah tinggal ini harus memperhatikan faktor keamanan dan kenyamanannya. Beberapa contoh gedung rumah tinggal adalah rumah, rumah susun, asrama, mess, kontrakan dan apartemen.

- Gedung Komersial

Gedung komersial didirikan untuk mendukung aktivitas komersial meliputi jual, beli, dan sewa. Gedung komersial ditujukan untuk keperluan bisnis sehingga faktor lokasi yang strategis memegang peranan penting bagi kesuksesan bangunan tersebut. Bebe-

Beberapa contoh gedung komersial di antaranya pasar, *supermarket*, *mall*, *retail*, pertokoan, perkantoran dan komplek kios.

– Gedung Fasilitas Penginapan

Gedung penginapan tercipta dari kebiasaan manusia yang sering melakukan aktivitas dengan berpindah-pindah tempat secara mobilitas. Keberadaan gedung ini memungkinkan seseorang bisa menyewa gedung untuk sementara waktu dengan keperluan menginap. Beberapa contoh gedung penginapan yaitu *motel*, *hotel*, *cottage* dan wisma tamu.

– Gedung Fasilitas Pendidikan

Gedung pendidikan didirikan untuk mendukung proses belajar dan mendapatkan ilmu dan pengetahuan yang baru. Beberapa contoh gedung fasilitas pendidikan adalah sekolah, universitas, perpustakaan dan gedung.

– Gedung Fasilitas Kesehatan

Gedung kesehatan didirikan untuk membantu masyarakat untuk melakukan berobat agar dapat sembuh dari sakit dan dapat menjalankan aktivitasnya. Beberapa contoh dari gedung fasilitas kesehatan adalah rumah sakit, puskesmas, apotek dan pusat rehabilitasi.

– Gedung Fasilitas Peribadatan

Gedung ibadah didirikan untuk memenuhi kebutuhan rohani manusia sebagai mahluk hidup yang memiliki Tuhan. Gedung peribadatan biasanya digunakan sebagai tempat beribadah dan upacara keagamaan. Beberapa contoh gedung fasilitas peribadatan adalah vihara, gereja, kelenteng, masjid dan pura.

– Gedung Fasilitas Transportasi

Gedung transportasi didirikan untuk sebagai pusat dari alat transportasi tertentu. Misalnya terminal untuk tempat berhentinya bis, pelabuhan sebagai tempat menepinya kapal, stasiun untuk pemberhentian kereta api, dan bandara sebagai tempat mendaratnya pesawat. Di gedung fasilitas transportasi ini juga umumnya dilengkapi dengan fasilitas-fasilitas layanan yang memungkinkan alat transportasi tersebut.

– Gedung Fasilitas Budaya dan Hiburan

Gedung budaya merupakan gedung yang dipakai untuk melestarikan dan atau mempertunjukkan suatu kebudayaan. Sedangkan gedung hiburan adalah gedung yang dipakai sebagai tempat menciptakan hal-hal yang menghibur. Pada gedung, hubungan antara faktor budaya dan faktor hiburan ini saling merekat dan mendukung satu sama lain. Sebagai contoh gedung pertunjukan yang menampilkan drama sarat budaya yang dapat menghibur penonton. Begitu juga dengan bioskop dan museum.

– Gedung Fasilitas Pemerintah dan Layanan Publik

Gedung pemerintahan adalah gedung yang digunakan oleh pemerintah untuk menunaikan tugas dan kewajibannya. Di samping itu, gedung pemerintah ini juga dipakai sebagai gedung layanan publik misalnya dalam pengurusan data kependudukan, berkas-berkas resmi, surat perijinan, laporan pengaduan, dan lain-lain. Itu sebabnya, pembuatan gedung ini harus dirancang sedemikian rupa agar dapat mendukung kegiatan-kegiatan tersebut. Adapun contoh-contoh bangunan pemerintahan dan layanan publik yaitu kantor polisi, kantor perizinan, kantor dinas, dan balai pemerintahan.

### 2.1.2 Pemantauan Kesehatan Struktural [2]

Pemantauan kesehatan struktural atau biasa disebut Structural Health Monitoring (SHM) merupakan sebuah proses penerapan deteksi kerusakan dan karakterisasi untuk struktur teknik seperti jembatan dan bangunan. Kerusakan dalam hal ini diartikan sebagai perubahan pada material atau sifat-sifat geometris dari suatu sistem struktural yang secara negatif mempengaruhi kinerja sistem. Sifat geometris yang dimaksud adalah bangunan yang dibangun terdapat dilahan yang memiliki kemiringan sehingga sebuah gedung akan lebih mudah mengalami kerusakan. Proses pemantauan kesehatan struktural melibatkan pengamatan suatu sistem dari waktu ke waktu menggunakan pengukuran respons sampel secara berkala dari berbagai sensor (sering yang digunakan adalah

(accelerometer), ekstrasi fitur yang peka terhadap kerusakan dari pengukuran ini, dan melakukan analisis statistik fitur untuk menentukan keadaan saat ini kesehatan sistem.

Salah satu contoh struktural adalah gedung. Gedung harus diukur kesehatannya untuk melihat apakah suatu gedung layak digunakan atau tidak. Gedung yang kesehatannya baik menjadi syarat untuk mendapatkan dokumen IMB (Izin Mendirikan Bangunan). Dokumen IMB merupakan dokumen dimana sebuah gedung yang akan didirikan mendapatkan izin dari pemerintah. Sebuah gedung dapat diukur kesehatannya dengan melakukan pemantauan atau *monitoring* gedung. Pemantauan kesehatan gedung adalah sebuah proses pemantauan informasi kondisi dan keamanan dari sebuah gedung. Tujuan dari pemantauan kesehatan gedung ini adalah:

- Memantau secara terus-menerus kondisi kesehatan gedung
- Mengetahui sesegera mungkin gejala-gejala tidak normal yang mungkin dapat terjadi pada sebuah gedung
- Mencatat perilaku-perilaku beban yang dikirim oleh gedung
- Sebagai sumber data untuk menganalisis dalam pengambilan keputusan dalam tindakan mencegah atau perawatan pada sebuah gedung

### Faktor-Faktor Kesehatan Gedung

Kesehatan sebuah gedung dapat dilihat dari beberapa faktor. Faktor-faktor yang mempengaruhi kesehatan gedung antara lain:

- Faktor Suhu

Suhu merupakan salah satu faktor alam yang berpengaruh kepada kesehatan sebuah gedung. Suhu yang ekstrim dan terjadi secara terus-menerus menyebabkan kesehatan gedung menjadi rusak terutama dibagian luar gedung tersebut. Beberapa contoh komponen yang harus dilindungi karena pengaruh suhu adalah lapisan water proofing diatas atap plat beton, cat pada listplank kayu, serta cat eksterior yang sering terkena panas ataupun dingin secara terus menerus

- Faktor Air Hujan

Faktor air hujan menjadi salah satu faktor yang berpengaruh kepada kesehatan suatu gedung. Air hujan dapat membuat kerusakan pada gedung dan kasus yang sering terjadi akibatnya adalah kebocoran pada gedung.

- Faktor Angin

Faktor angin merupakan salah satu alam yang berpengaruh terhadap kesehatan gedung. Faktor angin dapat membuat gedung mengalami kerusakan dan salah satu komponen gedung yang sering terkena akibat angin adalah penutup atap genteng. Pada serangan angin yang kencang, menimbulkan gerakan-gerakan pada atap yang menyebabkan atap mudah bergeser satu sama lain sehingga mudah lepas jika terjadi angin kencang.

- Faktor Getaran

Salah satu faktor yang dapat digunakan untuk pemantauan kesehatan gedung adalah getaran. Getaran merupakan salah satu faktor penting dalam kesehatan gedung. Getaran dalam sebuah gedung dapat berasal dari alat-alat mesin yang sedang berjalan, adanya sebuah kerusakan dalam gedung tersebut ataupun adanya sebuah pergeseran lempengan bumi atau biasa disebut gempa bumi. Getaran-getaran yang dihasilkan gedung seringkali tidak dapat dirasakan oleh manusia sehingga dibutuhkan sensor untuk mendeteksi getaran tersebut. Gedung akan dinyatakan baik dan dapat digunakan apabila getaran yang dihasilkan oleh sebuah gedung semakin kecil.

- Faktor Petir

Faktor petir merupakan faktor kesehatan gedung yang jarang, tetapi kerusakan yang terjadi pada gedung akibat petir tidak dapat dianggap sepele karena kerusakan yang dihasilkan oleh petir dapat berakibat fatal diantaranya listrik pada gedung mati, jaringan telepon ataupun internet di gedung juga dapat mati.

- Faktor Hama

Faktor hama merupakan faktor yang dapat mempengaruhi kesehatan gedung. Bedanya untuk faktor hama, gedung yang dapat mengalami kerusakan akibat faktor hama ini adalah gedung yang terbuat dari kayu. Salah satu contoh penyebab kerusakan akibat gedung dari faktor hama adalah rayap.

### Sifat Kerusakan Pada Gedung

Sifat kerusakan yang terjadi pada gedung dapat ditinjau dari pengaruh komponen tersebut hingga akibat dari kerusakan komponen sifat tersebut. Sifat-sifat kerusakan gedung dibagi menjadi tiga bagian yaitu:

- *Emergency*

Kerusakan yang memiliki pengaruh sangat tinggi terhadap aktivitas penghuni pada gedung dan mempengaruhi komponen lain dalam gedung tersebut. Contoh kerusakannya adalah kerusakan kran air, atap bocor, instalasi listrik.

- *Urgent*

Kerusakan yang memiliki pengaruh tinggi terhadap aktivitas penghuni dan kerusakan komponen lainnya pada gedung. Contoh kerusakannya adalah kerusakan pada keramik yang sering dilalui, jalan berlubang.

- *Normal*

Kerusakan kecil yang menyebabkan fungsi kurang sempurna atau penurunan tampak pada komponen yang mempunyai pengaruh kecil pada aktivitas penghuni. Contoh kerusakannya adalah cat dinding yang udah rusak.

## 2.2 Getaran [3]

Getaran adalah suatu peristiwa gerak bolak balik secara teratur suatu benda melalui suatu titik seimbang terhadap suatu titik acuan. Keseimbangan yang dimaksud adalah suatu keadaan dimana suatu benda berada pada posisi diam jika tidak ada gaya yang bekerja pada benda tersebut. Besar kecilnya suatu getaran yang dihasilkan oleh suatu benda dipengaruhi oleh jumlah energi yang diberikan. Semakin besar energi yang diberikan maka semakin kuat getaran yang terjadi. Satu getaran sama dengan satu kali gerakan bolak balik penuh dari benda tersebut.<sup>1</sup>

### Jenis getaran

Secara umum getaran dapat diklasifikasikan menjadi beberapa jenis yaitu:

- Getaran Bebas dan Paksa

Jika sebuah sistem diberi inisial gangguan, sehingga akan terjadi getaran dengan sendirinya, maka getaran tersebut dinamakan **getaran bebas**. Tidak ada gaya eksternal bekerja pada sistem. Contoh dari getaran bebas adalah gerakan bolak-balik yang ada pada sebuah pendulum. Sedangkan jika sebuah sistem diberikan suatu gaya dari luar yang secara berkala (berulang-ulang) maka getaran yang dihasilkan pada sistem disebut dengan **getaran paksa**. Salah satu contoh getaran paksa adalah getaran yang dihasilkan oleh mesin yang sedang bekerja. Apabila frekuensi suatu gaya eksternal sama dengan frekuensi gaya sistem, maka akan menimbulkan resonansi. Resonansi ini dapat membahayakan suatu sistem dan menyebabkan kerusakan struktur dari bangunan.

- Getaran Teredam dan Tidak Teredam

**Getaran tidak teredam** adalah getaran dimana jika tidak adanya energi dalam sebuah getaran yang hilang atau terdisipasi akibat adanya getaran atau hambatan lainnya. Sedangkan yang dimaksud dengan **getaran teredam** adalah sebuah getaran dimana mengalami pengurangan energi secara bertahap.

---

<sup>1</sup><https://www.gurupendidikan.co.id/getaran/>

- Getaran Mekanis dan Nonmekanis

**Getaran mekanis** adalah getaran suatu benda yang getarannya mengalami suatu pergeseran linear atau pergeseran sudut. Contoh getaran mekanis adalah getaran senar gitar pada saat dipetik, getaran pada bandul dan getaran atom pada zat padat. Sedangkan **getaran nonmekanis** adalah suatu gerakan yang juga melibatkan adanya perubahan pada besaran-besaran fisika. Contoh dari getaran nonmekanis adalah medan listrik serta medan magnet.

- Getran Deterministik dan Acak

**Getaran deterministik** adalah sebuah getaran dimana besarnya eksitasi (gaya atau gerakan) yang bekerja pada suatu sistem getaran diketahui pada waktu tertentu. **Getaran acak** adalah getaran dimana getaran atau gaya yang bekerja pada suatu sistem dihasilkan secara acak pada waktu tertentu. Contoh dari getaran acak adalah kecepatan angin, gerakan tanah selama gempa bumi.

## 2.3 Wireless Sensor Network [4]

*Wireless Sensor Network* (WSN) merupakan jaringan nirkabel yang terdiri dari sekumpulan node sensor yang saling terhubung yang diletakan pada suatu tempat dan memiliki kemampuan untuk mengukur kondisi lingkungan sekitar (*sensing*), melakukan komputasi dan dilengkapi dengan alat komunikasi *wireless* untuk komunikasi antara node sensor. *Wireless Sensor Network* juga berguna untuk memantau hasil *sensing* kondisi suatu lingkungan yang dilakukan oleh sensor. Selain itu WSN juga dapat mengatur data yang akan dikirimkan dan dikumpulkan di *base station* atau juga dapat diteruskan ke node sensor tetangganya hingga sampai ke *base station* sebagai pusat menggunakan *radio transceiver* untuk dilakukan pengelolaan data(Firdaus, 2014). Bentuk radio transceiver untuk setiap node menggunakan antena internal ataupun antena eksternal.

### 2.3.1 Penerapan Wireless Sensor Network di berbagai bidang

*Wireless Sensor Network* pada awalnya hanya digunakan untuk melakukan ilmu komputasi saja. Semakin lama dan majunya teknologi, *Wireless Sensor Network* telah digunakan untuk melakukan pemantauan ataupun pengukuran seperti di bidang militer, *Wireless Sensor Network* digunakan untuk mendeteksi musuh yang ada dilautan ataupun darat. Kemudian pemanfaatannya dikembangkan untuk membantu berbagai bidang kegiatan manusia dalam kehidupan. Penerapan *Wireless Sensor Network* untuk kehidupan manusia dapat dilihat pada contoh ilustrasi (Gambar 2.1)<sup>2</sup>. Berikut adalah beberapa penerapan *Wireless Sensor Network*:

- Bidang Pemantauan Lingkungan

*Wireless Sensor Network* dibidang pemantauan lingkungan memiliki banyak pengaplikasian seperti untuk memantau polusi udara, mendeteksi kebakaran hutan, mendeteksi adanya bencana alam khususnya gempa, dan lain-lain.

- Bidang Kesehatan

*Wireless Sensor Network* dapat digunakan pada aplikasi kesehatan seperti mendeteksi kondisi manusia yang memiliki kekurangan seperti disabilitas, monitoring penggunaan obat, serta bisa juga untuk mendiagnosis sebuah penyakit.

- Bidang Transportasi

Pada bidang transportasi, *Wireless Sensor Network* digunakan untuk mendeteksi arus lalu lintas secara aktual yang akan dikirimkan kepada pengendara. Selain itu, juga bisa digunakan untuk mendeteksi kecepatan kendaraan yang melaju apakah melewati batas kecepatan atau tidak.

- Bidang Militer

Pemanfaatan *Wireless Sensor Network* di bidang militer adalah Wide Area Tracking System (WATS). WATS merupakan prototipe jaringan yang menggunakan teknologi *Wireless Sensor*

<sup>2</sup><http://eprints.polsri.ac.id/4497/3/File/20III.pdf>

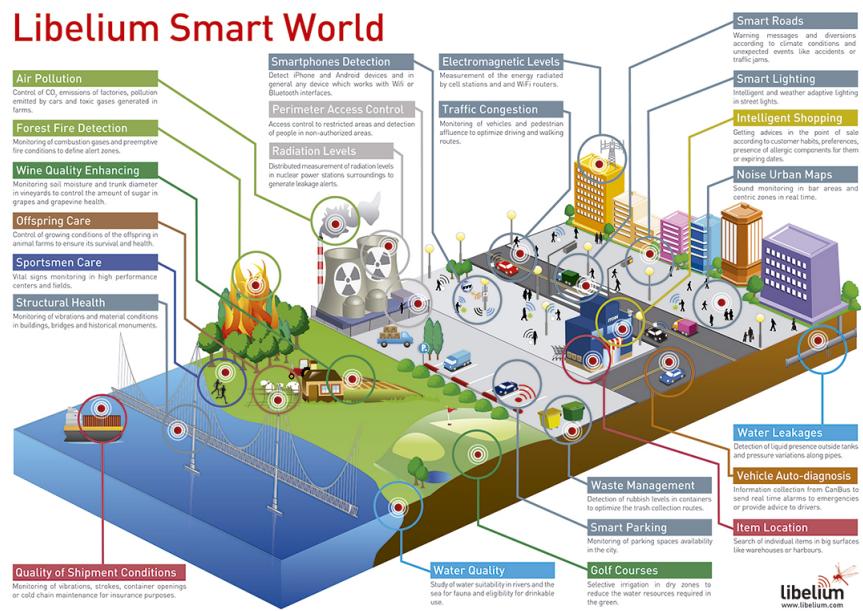
*Network* untuk mendeteksi ancaman nuklir. Pemanfaatan lainnya untuk bidang militer dapat juga digunakan untuk mendeteksi serangan dari musuh.

- Bidang Pertanian

Pada bidang pertanian, Wireless Sensor Network dapat digunakan untuk membantu pengelola pertanian untuk penggunaan air, kelembaban tanah yang digunakan, pH tanah yang digunakan untuk pertanian, serta mengelola pembuangan pertanian mereka.

- Bidang Infrastruktur

*Wireless sensor network* juga digunakan dalam pembangunan infrastruktur. Pemanfaatan *Wireless Sensor Network* biasa digunakan untuk melakukan pengamatan kondisi pembangunan infrastruktur, baik dari segi bangunan maupun geografis seperti mengukur getaran sebuah pembangunan. Penerapan WSN untuk membantu pembangunan infrastruktur dilakukan secara *daily* dan *real-time* untuk menjaga kondisi lingkungan selama proses pemabangunan selesai dilakukan.

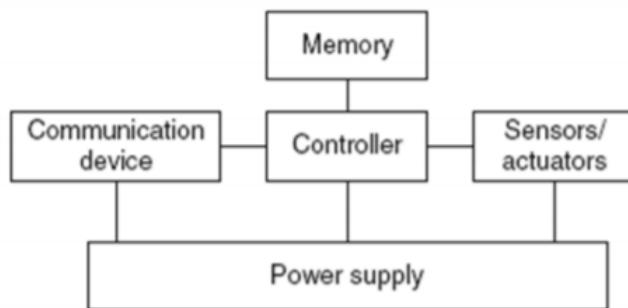


Gambar 2.1: Ilustrasi penerapan *Wireless Sensor Network*

### 2.3.2 Struktur Node Sensor

Node adalah salah satu titik sambungan, titik redistribusi, atau titik akhir komunikasi<sup>3</sup>. Komponen utama node sensor antara lain *controller*, *transceiver*, *memory*, *power source*, dan *sensor*.(Gambar 2.2).

<sup>3</sup><http://gilang777.blogspot.com/2015/06/pengertian-node-jaringan.html>



Gambar 2.2: Struktur Node Sensor

- Controller

*Controller* adalah inti utama yang ada pada node sensor. *Controller* juga bertindak sebagai pengatur fungsi dari komponen-komponen lain. *Controller* juga mengumpulkan data dari sensor lain dan memproses data. Pada *controller* terdapat *microcontroller* yang mengatur dan melakukan komputasi data. *Microcontroller* lebih sering menjadi alternatif karena mengurangi penggunaan energi dan adanya sleep states yang berarti hanya bagian dari *controller* saja yang aktif.

- Communication Device

Communication Device digunakan untuk menerima atau mengirim data antar node sensor. Communication Device membuat node sensor dapat terhubung dalam jaringan dan berkomunikasi dengan node sensor lainnya.

- Sensor / Actuator

Sensor merupakan bagian yang digunakan untuk melakukan *sensing* atau pengukuran terhadap suatu keadaan lingkungan yang diamati. *Actuator* berfungsi sebagai pengubah sinyal dari lingkungan menjadi besaran-besaran fisik.

- Memory

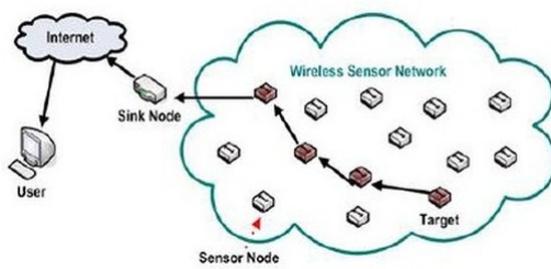
*Random Access Memory* (RAM) digunakan untuk menyimpan hasil sementara yang didapat dari *sensing* dari sensor. RAM ini memiliki sifat *volatile* yang berarti jika node sensor mati atau energi habis maka data-data yang ada pada RAM akan hilang.

- Power Supply

Power Supply digunakan sebagai sumber energi yang dibutuhkan untuk mengoperasikan fungsionalitas dari keseluruhan komponen node sensor lainnya. Penyediaan energi ini dapat memiliki dua jenis metode, yaitu *storing energy* dan *energy scavenging*. Metode *storing energy*, sensor node menggunakan baterai sebagai energi utamanya. Jenis baterai yang digunakan dapat yang diisi ulang maupun yang tidak dapat diisi ulang. Sedangkan metode *energy scavenging* digunakan saat membuat *Wireless Sensor Network* yang akan digunakan dalam waktu yang lama. Pada metode *energy scavenging*, node sensor menggunakan perubahan energi sebagai sumber energi utamanya. Energi yang didapat dengan metode *energy scavenging* didapatkan setelah mengonversi pemanfaatan energi alam seperti cahaya matahari, air atau angin untuk dijadikan energi listrik untuk menjalankan node sensor.

### 2.3.3 Arsitektur *Wireless Sensor Network*

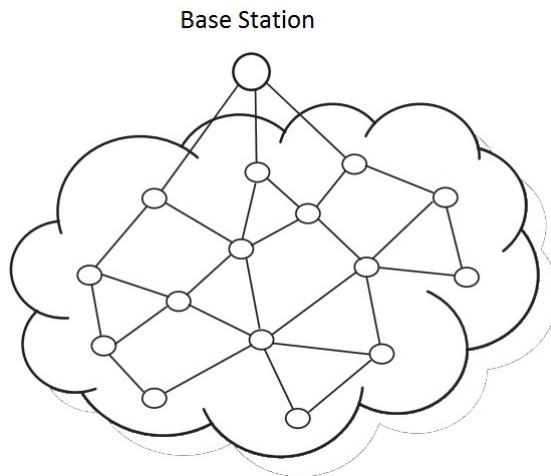
Pada *Wireless Sensor Network*, arsitektur yang biasanya dipakai adalah **arsitektur flat atau peer-to-peer** dan **arsitektur hierarki**. Perbedaan antara kedua jenis arsitektur adalah cara node sensor dalam berkomunikasi. Pada arsitektur *flat*, node yang disebar dapat langsung mengirimkan data hasil sensing ke *base station*. Sedangkan pada arsitektur *hirarkikal*, node yang disebar harus mengirimkan data ke *cluster head* terlebih dahulu, sebelum diteruskan ke *base station*.



Gambar 2.3: Ilustrasi penerapan *Wireless Sensor Network*

(i) **Arsitektur Flat / Peer-to-Peer**

Pada arsitektur *flat*, setiap node sensor yang disebar memiliki tugas dan peran yang sama dalam melakukan *sensing* dan mengirimkan hasilnya ke *base station*. Data hasil *sensing* ini langsung dikirimkan ke *base station* tanpa butuh sebuah perantara.



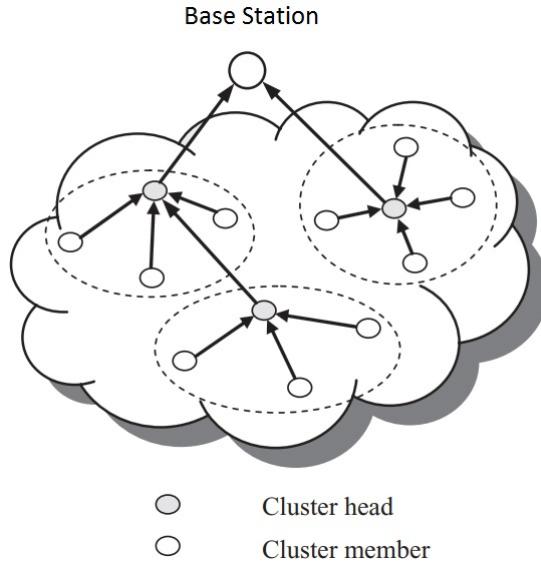
Gambar 2.4: Arsitektur *flat* pada *Wireless Sensor Network*

(ii) **Arsitektur Hierarki**

Pada arsitektur hierarki, setiap hasil *sensing* dari sensor tidak langsung dikirimkan ke *base station*. Melainkan setiap node sensor akan membentuk sebuah grup yang disebut dengan *cluster*. Tiap *cluster* terdiri dari sebuah *cluster head* dan *cluster member*. *Cluster head* bertindak sebagai penerima data hasil *sensing* dari *cluster member*, untuk diteruskan ke *base station*. Jenis arsitektur hierarki juga dapat dibedakan berdasarkan jarak antara *cluster head* dengan *cluster member*. Perbedaan dari jenis ini adalah jumlah *hop* yang dibutuhkan untuk mencapai tujuan yaitu *base station*.

- *Single hop*

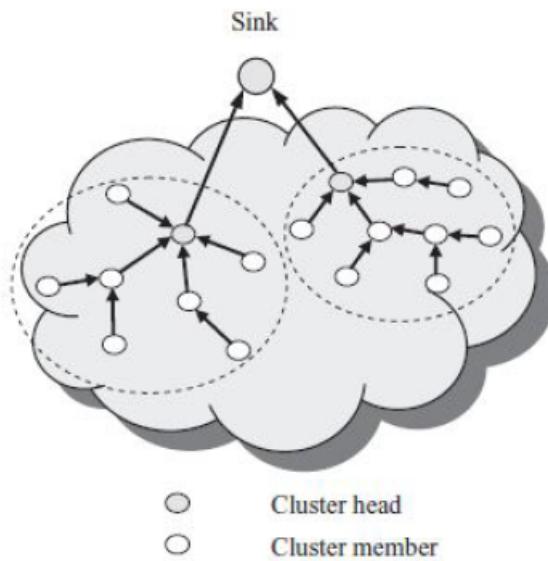
Pada arsitektur hierarki *single hop*, *cluster member* hanya membutuhkan satu lompatan untuk mencapai ke *cluster head*. Dapat dikatakan, data hasil *sensing* yang diterima oleh *cluster member* dapat langsung diterima oleh *cluster head* tanpa adanya perantara dan dapat diteruskan ke *base station*.



Gambar 2.5: Arsitektur *single hop* pada *Wireless Sensor Network*

- *Multi hop*

Pada arsitektur hierarki *multi hop*, *cluster member* membutuhkan lebih dari satu lompatan untuk mencapai *base station*. *Cluster member* akan mengirimkan data ke *cluster member* yang jaraknya lebih dekat dengan *cluster head* dan ini akan terus dilakukan sampai data yang dikirimkan sampai di *cluster head* kemudian akan diterukan ke *base station*.



Gambar 2.6: Arsitektur *multi hop* pada *Wireless Sensor Network*

#### 2.3.4 Topologi *Wireless Sensor Network*

Topologi pada *wireless sensor network* memiliki beberapa jenis. Tiap jenis topologi dibedakan berdasarkan tujuan, skala jaringan dan kondisi lingkungan. Beberapa jenis topologi pada *wireless sensor network* adalah topologi *point-to-point*, *bus*, *tree*, *star*, *ring* dan *mesh*.

- Topologi *Point-to-Point*

Topologi *point-to-point* adalah topologi yang menghubungkan dua titik (Gambar 2.7). Pada

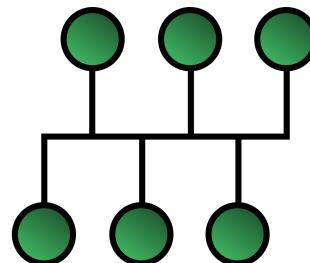
topologi ini dibagi menjadi dua yaitu *permanent point-to-point* dan *switched point-to-point*. *Permanent point-to-point* adalah koneksi antara dua titik dan bersifat tidak dapat diubah (permanen). Sedangkan *switched point-to-point* adalah koneksi point-to-point yang dapat dipindahkan antara node yang berbeda.<sup>4</sup>



Gambar 2.7: Topologi *point-to-point*

- **Topologi Bus**

Topologi bus akan terdiri dari node-node yang terhubung pada sebuah jalur. Jalur ini digunakan oleh node-node untuk saling berkomunikasi (Gambar 2.8). Sifat komunikasi pada jalur ini adalah satu arah, dimana komunikasi node dilakukan secara bergantian. Topologi ini sederhana dan mudah untuk diimplementasikan. Kekurangan dari topologi ini adalah apabila jalur mengalami kerusakan maka setiap node tidak dapat melakukan komunikasi lagi.



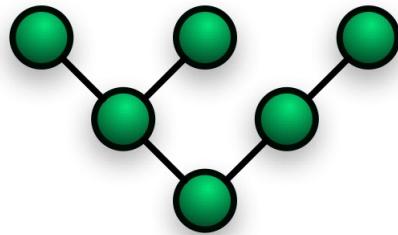
Gambar 2.8: Topologi *bus*

- **Topologi Tree**

Pada topologi *tree*, node-node akan disusun secara hierarki dengan sebuah node yang berada pada level paling atas disebut sebagai *root node*. *Root node* akan terhubung dengan satu atau lebih node yang levelnya dibawah sehingga *root node* akan bertindak sebagai komunikasi utama (Gambar 2.9). Penggunaan topologi *tree* lebih mudah untuk melakukan identifikasi dan meminimalisir kesalahan. Kelemahan dari topologi ini adalah topologi ini akan semakin sulit dikonfigurasi seiring jika ukuran *tree* yang sangat besar dan banyaknya *level tree* yang ada.

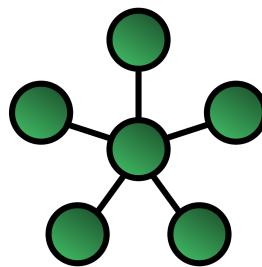
---

<sup>4</sup><http://link.springer.com/chapter/10.1007/978-1-4302-6014-14.html>

Gambar 2.9: Topologi *tree*

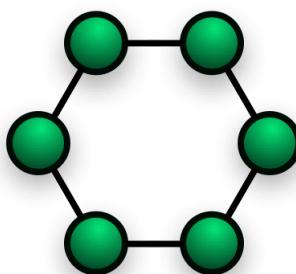
- Topologi *Star*

Topologi *star* memiliki sebuah node yang berada di tengah sebagai *hub* atau *switch* yang biasa disebut *central node* (Gambar 2.10). Setiap node-node akan terhubung dengan *central node* dan melakukan komunikasi ke node lain melalui *central node* ini. *Central node* yang telah mendapatkan pesan dari node pengirim akan meneruskan pesannya ke node tujuan. Apabila *central node* mengalami kerusakan, maka tidak akan terjadinya komunikasi antar node karena segala komunikasi harus dilakukan melalui *central node*.

Gambar 2.10: Topologi *star*

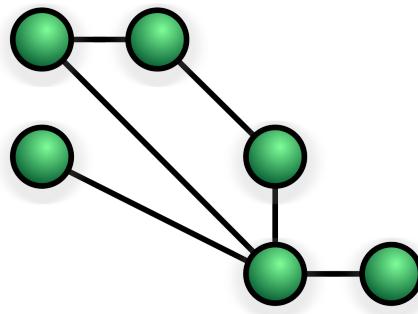
- Topologi *ring*

Jaringan topologi *ring* berbentuk rangkaian node yang saling terhubung dengan dua node terdekat lainnya sehingga akan berbentuk seperti lingkaran atau cincin (Gambar 2.11). Pada topologi *ring*, node akan mengirimkan pesan kemudian akan diteruskan ke node tetangganya sampai menemukan node tujuan yang akan dikirimkan pesan. Kekurangannya adalah salah satu node mati maka komunikasi jaringannya akan mati. Masalah ini dapat diatasi dengan membuat sebuah node tidak hanya dapat melakukan komunikasi satu arah melainkan ke arah sebaliknya.

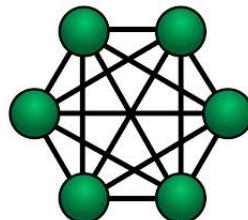
Gambar 2.11: Topologi *ring*

- Topologi *Mesh*

Topologi mesh dibagi menjadi dua jenis yaitu partially connected mesh dan fully connected mesh. Perbedaannya kedua jenis ini adalah hubungan antar node, pada partially connected mesh, node dapat terhubung dengan satu atau lebih node lainnya (Gambar 2.12). Sedangkan fully connected mesh, setiap node harus terhubung dengan semua node-node lain yang ada pada sebuah jaringan (Gambar 2.13).



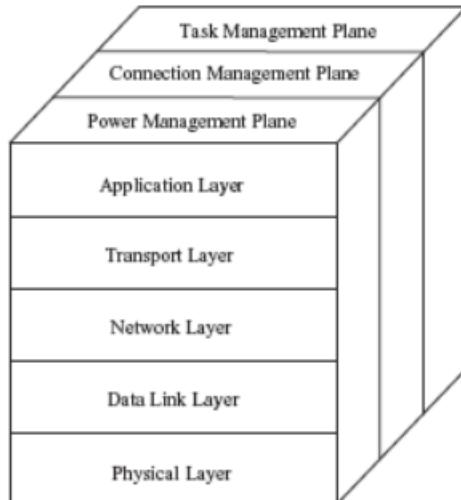
Gambar 2.12: Topologi *partially connected mesh*



Gambar 2.13: Topologi *fully connected mesh*

### 2.3.5 Protokol *Wireless Sensor Network*

*Wireless Sensor Network* memiliki lima layer protokol yaitu *physical layer*, *data link layer*, *network layer*, *transport layer* dan *application layer* (Gambar 2.14). Selain itu, protokol *Wireless Sensor Network* juga dibagi menjadi 3 grup manajemen yaitu *power management plane*, *connection management plane* dan *task management plane*. *Power management plane* bertanggung jawab untuk mengatur tingkat kekuatan sebuah sensor untuk melakukan *sensing*, memproses data dan melakukan komunikasi. *Connection management plane* bertanggung jawab untuk melakukan konfigurasi terhadap node sensor yang terkait dengan koneksi antar node sensor. *Task management plane* bertanggung jawab untuk pembagian tugas diantara node sensor dalam melakukan *sensing* agar energi yang digunakan efektif dan efisien.



Gambar 2.14: Layer pada *Wireless Sensor Network*

- *Physical layer*

*Physical layer* bertanggung jawab untuk mengubah *bit stream* dari *data link layer* menjadi sinyal agar bisa melakukan transmisi melalui media komunikasi yang tersedia. Pengubahan ini dilakukan agar transmisi dapat dilakukan melalui *transceiver*. Salah satu cara yang bisa digunakan adalah menggunakan *Radio Frequency* (RF). *Radio Frequency* sering dipakai karena biaya yang dikeluarkan murah dan ukuran perangkat yang kecil.

- *Data Link layer*

*Data Link layer* bertanggung jawab untuk melakukan *multiplexing* pada aliran data, membentuk *data frame*, mendeteksi *data frame*, *medium access*, and *error control*. Data Link layer juga memiliki proses yang penting yaitu *Medium Access Control* (MAC). Protokol MAC ini bertindak untuk melakukan kontrol terhadap akses media yang dilakukan oleh node sensor dan mencegah terjadinya adanya paket yang bertabrakan.

- *Network layer*

*Network layer* bertanggung jawab untuk melakukan *routing* dari node sensor ke *sink node*. Network layer juga merupakan lapisan yang menyediakan jalur komunikasi jaringan sehingga *network layer* bertugas untuk menentukan jenis komunikasi antar node yang digunakan apakah jenis yang digunakan *single hop* atau *multi hop*.

- *Transport layer*

*Transport layer* bertanggung jawab untuk pengiriman data yang *reliable* antar node sensor ke node sensor lainnya atau ke *sink node*. Pengiriman data pada *Wireless Sensor Network* terbagi menjadi dua jenis, yaitu **upstream** dan **downstream**. Kedua jenis ini dibedakan berdasarkan dari asal pengirim dan penerima data. **Upstream** merupakan node sensor mengirimkan data *sensing* ke *sink node*. Sedangkan **downstream**, *sink node* akan mengirimkan perintah-perintah ataupun *query* ke setiap node sensor yang ada. Setiap jenis pengiriman, kebutuhan *reliability* nya berbeda-beda. Pada *upstream*, *reliable* data dapat ditoleransi karena node sensor mengirimkan data ke sink node secara berulang-ulang sehingga data yang hilang dapat dikoreksi. Pada *downstream*, tidak dapat ditoleransi karena jika data yang dikirimkan tidak *reliable*, maka aplikasi tidak dapat dijalankan.

- *Application layer*

*Application layer* bertanggung jawab untuk manajemen lalu lintas dan menyediakan antar muka perangkat lunak untuk berbagai aplikasi yang menerjemahkan data dalam bentuk yang dapat dimengerti atau mengirim *query* untuk mendapatkan informasi tertentu.

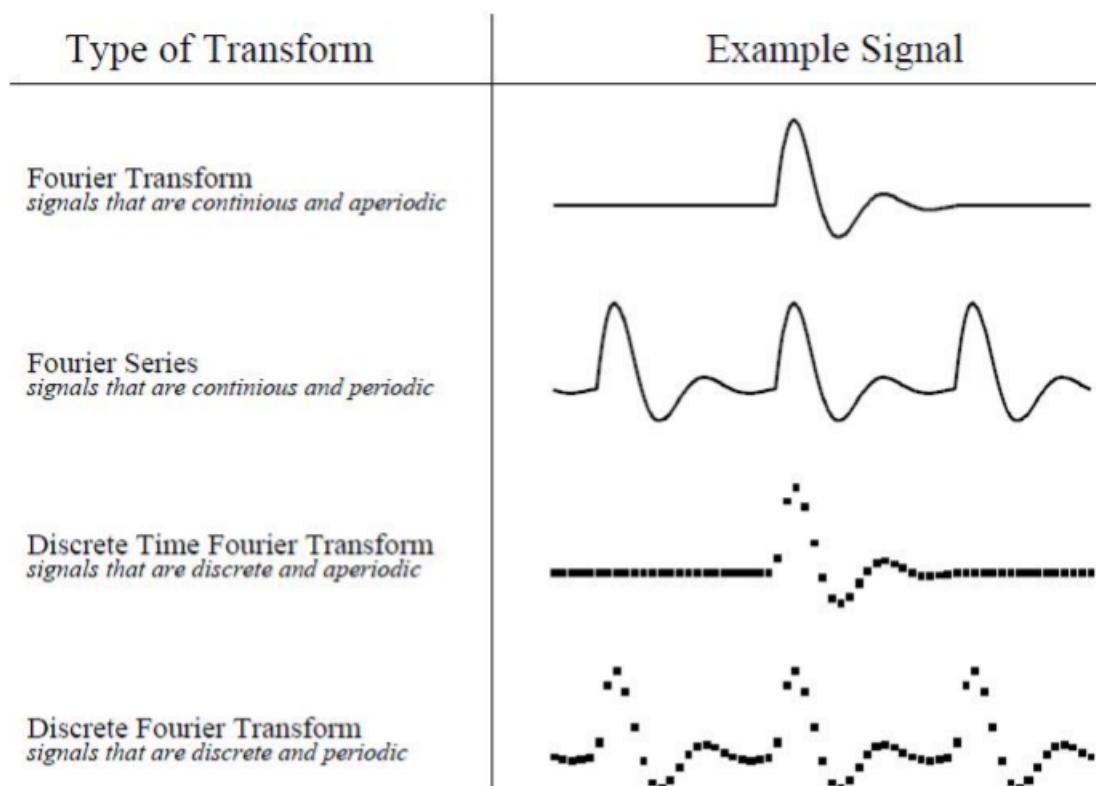
## 2.4 Accelerometer [5]

Salah satu sensor yang terdapat pada node sensor adalah sensor *accelerometer*. *accelerometer* adalah alat yang mengukur *linear acceleration*. Accelerometer memiliki kemampuan untuk mengukur akselerasi, kemiringan, dan getaran statis atau dinamis pada 1 sumbu saja  $x/y/z$  yang biasa disebut *single axis* atau pada 2 sumbu contohnya  $(x,y)$  atau  $(x,z)$  atau  $(y,z)$  yang biasa disebut dengan *two axis* ataupun 3 sumbu yaitu  $(x,y,z)$  yang disebut *three axis*. Sensor *accelerometer* dibagi menjadi 2 jenis yaitu:

- *Absolute accelerometer*  
*accelerometer* ini terpasang langsung pada objek yang akan diukur.
- *Relative accelerometer*  
*accelerometer* ini mengukur jarak antara objek yang diukur dan titik acuan yang stabil atau bergerak dengan konstan. *accelerometer* ini sering digunakan untuk mengukur getaran dari jarak tertentu.

## 2.5 Fourier Transform

Fourier Transform adalah sebuah teknik yang digunakan untuk menguraikan sinyal menjadi sinusoids. Sinyal yang diuraikan terdapat sinyal yang bersifat *continuous* atau sinyal bersifat *discrete*, *periodic* ataupun *aperiodic*. Sinyal yang bersifat *periodic* memiliki pola sinyal yang berulang-ulang sedangkan *aperiodic* memiliki pola yang tidak berulang. Berdasarkan jenis sinyal yang diurai, *Fourier Transform* dibagi menjadi 4 kategori (Gambar 2.16).



Gambar 2.15: Kategori *Fourier Transform* dan contoh bentuk sinyal

### 2.5.1 Discrete Fourier Transform (DFT)

*Discrete Fourier Transform* memiliki 2 versi yaitu menggunakan bilangan *real* dan menggunakan bilangan *complex*. *Discrete Fourier Transform* yang menggunakan bilangan complex dapat dikembangkan menjadi *Fast Fourier Transform* 5.3 dan akan memiliki persamaan seperti pada persamaan 2.1

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{-j\left(\frac{2\pi}{N}\right)nk} \quad (2.1)$$

Pada persamaan 2.1, variabel N merupakan jumlah sampel,  $x(n)$  merupakan input sinyal pada domain waktu, dan  $X(k)$  merupakan hasil keluaran sinyal pada domain frekuensi.

### 2.5.2 Fast Fourier Transform (FFT)

*Fast Fourier Transform* adalah teknik DFT yang lebih efisien. FFT memiliki cara perhitungan DFT yang lebih cepat dan memiliki hasil yang sama dengan hasil DFT. Salah satu contoh dari algoritma FFT adalah *Cooley-Tukey Algorithm*.

*Cooley-Tukey Algorithm* ini terdapat 2 jenis proses yaitu *Decimation in Time* (DIT) dan *Decimation in Frequency* (DIF). Jika masukan dari sinyal pada proses perhitungan adalah domain waktu, maka akan diproses dengan DIT sedangkan jika masukan dari sinyal pada proses perhitungan adalah domain frekuensi, maka akan diproses dengan DIF. DIT akan mengubah sinyal dari domain waktu menjadi domain frekuensi sedangkan DIF akan mengubah sinyal dari domain frekuensi menjadi domain waktu. Setiap masukan dari proses DIT maupun DIF harus diurutkan dalam *bit reverse* agar hasil keluarannya dalam urutan yang tepat. Proses DIT dapat dilihat pada 2.5.2 dan contoh visual dari DIT dengan panjang FFT 8 pada (Gambar ??)

---

#### Algorithm 1 Algoritma DIT

---

```

1: Input : masukan sinyal dalam domain waktu dan sudah dalam urutan bit-reverse
2: Output : masukan sinyal dalam domain frekuensi
3: for Setiap kelipatan 2 dari panjang FFT ( $N$ ) do
4:   for Setiap indeks semua masukan tiap bagian  $N$  do
5:     for Setiap indeks ( $k$ ) sampai  $N/2$  do
6:       masukan dengan indeks genap ditambah dengan masukan dengan indeks ganjil yang
      sudah dikalikan dengan  $W_N^k$ 
7:       masukan dengan indeks ganjil dikurang dengan masukan dengan indeks ganjil yang
      sudah dikalikan dengan  $W_N^k$ 
8:     end for
9:   end for
10: end for
11: return finalOrder

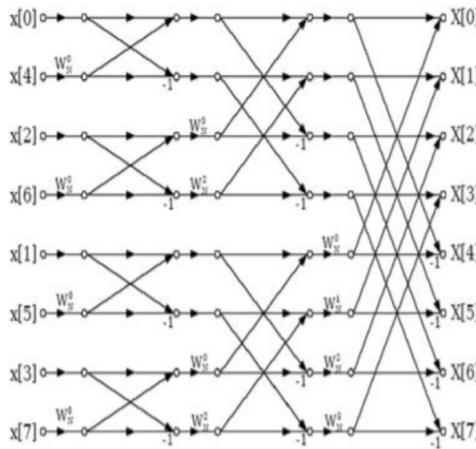
```

---

$$W_N^k = e^{-j\left(\frac{2\pi k}{N}\right)} \quad (2.2)$$

Persamaan 2.2 disebut juga dengan *twiddle factor*. Nilai k pada persamaan ini adalah indeks bernilai sampai dengan  $N/2$ .

Index	Binary	Bit-reversed Binary	Bit-reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Tabel 2.1: Tabel contoh *bit reverse* dari indeks 0 sampai 7Gambar 2.16: Contoh visual *Decimation In Time* dengan panjang 8

## 2.6 PreonVM

PreonVM adalah *virtual machine* (VM) yang berasal dari VIRTENIO untuk digunakan dalam sistem *embedded* dengan sumber daya yang sangat rendah<sup>5</sup>. *Virtual machine* PreonVM ini sudah sangat optimal dan tidak memerlukan sistem operasi tambahan dan berjalan langsung pada mikrokontroler. Dengan PreonVM, *developer* dapat membuat aplikasi sensor dengan mudah menggunakan bahasa Pemograman Java yang mengumpulkan data-data (hasil *sensing*) dari sensor. API pada PreonVM mendukung antarmuka radio sesuai dengan IEEE 802.15.4.

### 2.6.1 Fitur PreonVM

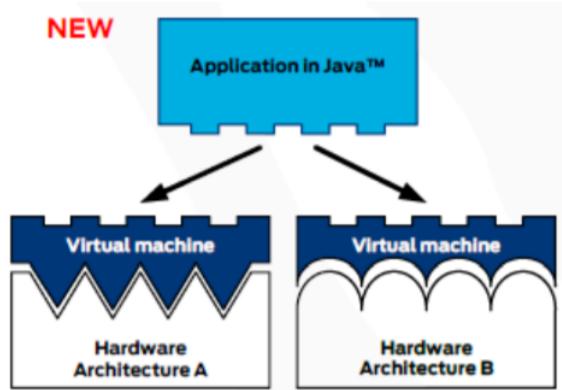
Fitur-fitur yang dimiliki oleh PreonVM sebagai berikut:

- Aplikasi dibangun dengan Bahasa Pemrograman Java
- Mendukung tipe data pada Java seperti *char*, *byte*, *int*, *long*, *float*, atau *double*
- Jumlah *thread* yang tidak terbatas
- *System properties* untuk konfigurasi aplikasi
- *Garbage collection* dengan *memory defragmentation*
- *Exception handling* (*try*, *catch*, *Exception*, atau *Runtime Exception*)

<sup>5</sup><https://www.virtenio.com/en/portfolio-items/preonvm/>

### 2.6.2 Kelebihan PreonVM

Kelebihan PreonVM adalah PreonVM menggunakan *object-oriented programming* menggunakan bahasa pemrograman Java pada *virtual machine* nya untuk *embed-system*. PreonVM juga dioptimasi agar aplikasi dapat dijalankan 8-bit sampai 32-bit *microcontroller* dengan 8KB RAM dan 128KB Flash minimum. *Virtual Machine* pada PreonVM dapat membuat aplikasi dapat berjalan secara sendiri pada arsitektur yang digunakan. Sehingga aplikasi Java yang dibuat dapat dijalankan pada arsitektur yang berbeda-beda tanpa harus dijalankan (Gambar 2.17).



Gambar 2.17: *Virtual Machine* yang membuat aplikasi dapat dijalankan secara independen

### 2.6.3 Class Library PreonVM

PreonVM memiliki 2 *package* yaitu *package* dari Virtenio itu sendiri dan *package* dari Java. *Package* yang disediakan oleh Virtenio dibagi lagi menjadi beberapa bagian diantaranya adalah *Route Packages*, *Radio Packages* dan *Other Virtenio Packages*. Sedangkan *package* yang dari Java terdiri dari *Java Related Packages*. Berikut ada tabel-tabel package yang disediakan:<sup>6</sup>

Tabel 2.2: Tabel Route Packages

Package	Deskripsi
com.virtenio.route.aodv	Paket yang berisi kelas terkait AODV (Ad Hoc On Demand Vector) Routing

Tabel 2.3: Tabel Radio Packages

Package	Deskripsi
com.virtenio.radio	Paket yang berisi kelas terkait radio
ESRT com.virtenio.radio.ieee_802_15_4	Paket yang berisi kelas terkait IEEE 802.15.4

<sup>6</sup><https://virtenio.com/assets/vm/javadoc/overview-summary.html>

Tabel 2.4: Tabel Other Virtenio Packages

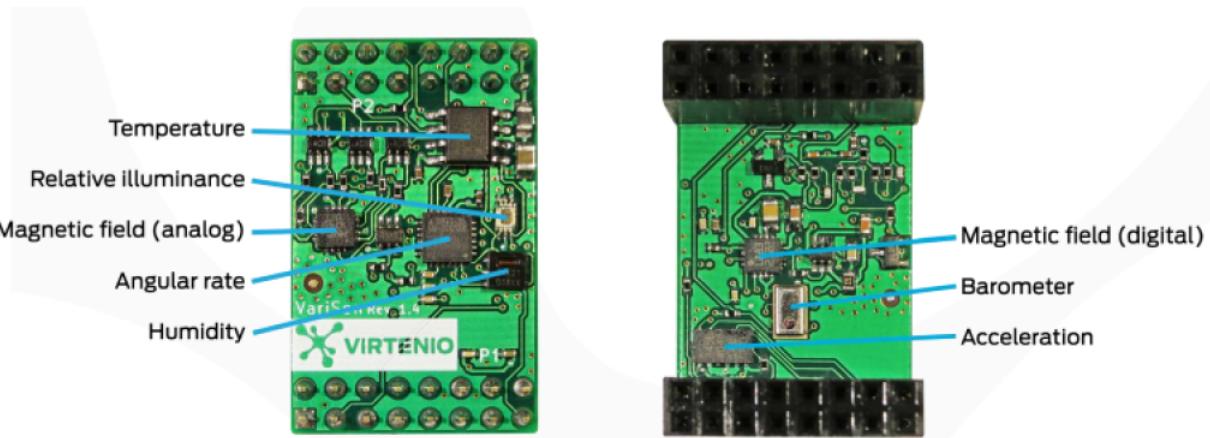
Package	Deskripsi
com.virtenio.crypt	Paket yang berisi untuk enkripsi dan dekripsi
com.virtenio.driver	Paket yang berisi drivers untuk berbagai perangkat
com.virtenio.driver.adc	Paket yang berisi kelas ADC <i>driver</i>
com.virtenio.driver.atmodem	Paket yang berisi kelas ATModem <i>driver</i>
com.virtenio.driver.button	Paket yang berisi kelas button <i>driver</i>
com.virtenio.driver.can	Paket yang berisi kelas CAN <i>driver</i>
com.virtenio.driver.cpu	Paket yang berisi kelas CPU <i>driver</i>
com.virtenio.driver.device	Paket yang berisi kelas device <i>driver</i>
com.virtenio.driver.device.at86rf212	Paket yang berisi driver untuk perangkat AT86RF212
com.virtenio.driver.device.at86rf231	Paket yang berisi driver untuk perangkat AT86RF231
com.virtenio.driver.flash	Paket yang berisi kelas Flash <i>driver</i>
com.virtenio.driver.gpio	Paket yang berisi kelas GPIO device <i>driver</i>
com.virtenio.driver.i2c	Paket yang berisi kelas I2C device <i>driver</i>
com.virtenio.driver.irq	Paket yang berisi kelas IRQ device <i>driver</i>
com.virtenio.driver.led	Paket yang berisi kelas LED device <i>driver</i>
com.virtenio.driver.lin	Paket yang berisi kelas LIN device <i>driver</i>
com.virtenio.driver.onewire	Paket yang berisi kelas OneWire device <i>driver</i>
com.virtenio.driver.pwm	Paket yang berisi kelas PWM ( <i>pulse-width modulation</i> ) device <i>driver</i>
com.virtenio.driver.ram	Paket yang berisi kelas FRAM device <i>driver</i>
com.virtenio.driver rtc	Paket yang berisi kelas untuk pengaturan jam secara real-time, dan real-counter device <i>driver</i>
com.virtenio.driver.spi	Paket yang berisi kelas SPI ( <i>Serial Peripheral Interface</i> ) device <i>driver</i>
com.virtenio.driver.sw	Paket yang berisi kelas switch device <i>driver</i>
com.virtenio.driver.timer	Paket yang berisi kelas hardware timer device <i>driver</i>
com.virtenio.driver.usart	Paket yang berisi kelas USART device <i>driver</i>
com.virtenio.driver.watchdog	Paket yang berisi WatchDog device <i>driver</i>
com.virtenio.io	Paket Virtenio VM yang berisi IO
com.virtenio.lib	Paket Virtenio VM yang berisi pengaturan classlib
com.virtenio.misc	Paket tambahan Virtenio VM
com.virtenio.net	
com.virtenio.vm	
com.virtenio.vm.event	Sistem event pada Virtenio VM untuk menangani event <i>asynchronous</i> dan <i>synchronous</i>

Tabel 2.5: Tabel Java Related Pakcages

Package	Deskripsi
java.io	Paket Java IO
java.lang	Paket Java lang
java.lang.annotation	Paket Annotation pada Java
java.lang.ref	
java.nio	
java.nio.channels	
java.text	Paket Java Text
java.util	Paket Java Utility yang berisi collection
java.util.regex	Paket Java Regular Expression

#### 2.6.4 Preon32

Preon32 merupakan salah satu sensor node buatan VIRTENIO. Preon32 menggunakan PreonVM digunakan sebagai operating software untuk sensor node ini. Pada umumnya, Preon32 memiliki 5 jenis sensor pada sebuah board. Sensor yang ada antara lain adalah sensor suhu (*temperature sensor*), sensor cahaya (*light intensity sensor*), sensor tekanan udara (*air pressure sensor*), sensor getaran (*acceleration sensor*) dan sensor kelembaban udara (*relative humidity sensor*). Preon32 juga terdapat versi tambahannya dilengkapi dengan sensor untuk mendeteksi *gyroscope* dan medan magnet (Gambar 2.18).



Gambar 2.18: Preon32 Board

### 2.6.5 Spesifikasi Sensor-Sensor Preon32

Berikut spesifikasi sensor-sensor yang terdapat di Preon32:

- Sensor suhu (*temperature* sensor)
  - Manufacture : Analog Devices
  - Model : ADT7410
  - Interface : digital, I2C
  - Resolution : 16-Bit
  - Range : -40°C sampai +105°C
  - Accuracy :  $\pm 0.5^\circ\text{C}$
- Sensor cahaya (*light intensity* sensor)
  - Manufacture : Rohm
  - Model : BH1715FVC
  - Interface : digital, I2C
  - Resolution : 16-Bit
  - Range : 1 lx to 65355 lx
- Sensor tekanan udara (*air pressure* sensor)
  - Manufacture : Freescale
  - Model : MPL115A2
  - Interface : digital, I2C
  - Resolution : 0,15 kPa
  - Range : 50 kPa sampai 115 kPa
  - Accuracy :  $\pm 1.0$  kPa
- Sensor getaran (*acceleration* sensor)
  - Manufacture : Analog Devices
  - Model : ADXL345
  - Interface : digital, SPI
  - Resolution : 13 Bit per axis
  - Range :  $\pm 16$  g, 3 axis
  - Accuracy : 3,9 mg/LSB
- Sensor kelembaban udara (*relative humidity* sensor)
  - Manufacture : Sensirion
  - Model : SHT21
  - Interface : digital, I2C
  - Resolution : 12-Bit

- Range : 0 %RH sampai 100 %RH
- Accuracy :  $\pm 2,0$  %RH (typ.)

## BAB 3

# ANALISIS

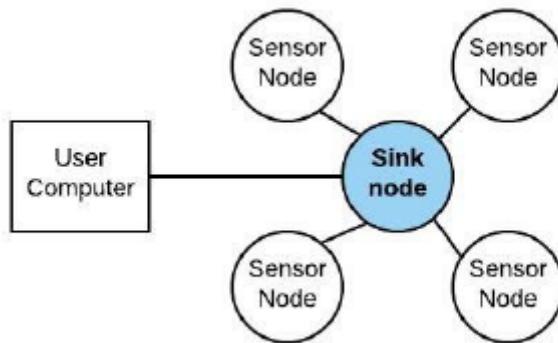
### 3.1 Analisis Aplikasi Pemantauan Getaran Gedung Berbasis *Wireless Sensor Network*

Perangkat lunak yang dibangun merupakan sebuah perangkat lunak yang berfungsi untuk memantau getaran gedung menggunakan sensor-sensor yang terhubung dalam sebuah jaringan. Sensor-sensor tersebut akan disebar di sekitar area gedung-gedung dan melakukan sensing terhadap getaran yang dihasilkan oleh gedung tersebut. Kemudian sensor-sensor tersebut mengirimkan hasil *sensing* ke *base station*, berupa besaran sensing dengan satuan gravitasi dari arah X, Y, dan Z. Base-station akan menerima semua hasil *sensing* dan disimpan di komputer dimana *base-station* berada, kemudian diproses untuk memperoleh frekuensi dari getaran pada setiap waktu dan ditampilkan ke layar komputer secara visual dan dalam waktu *near realtime*.

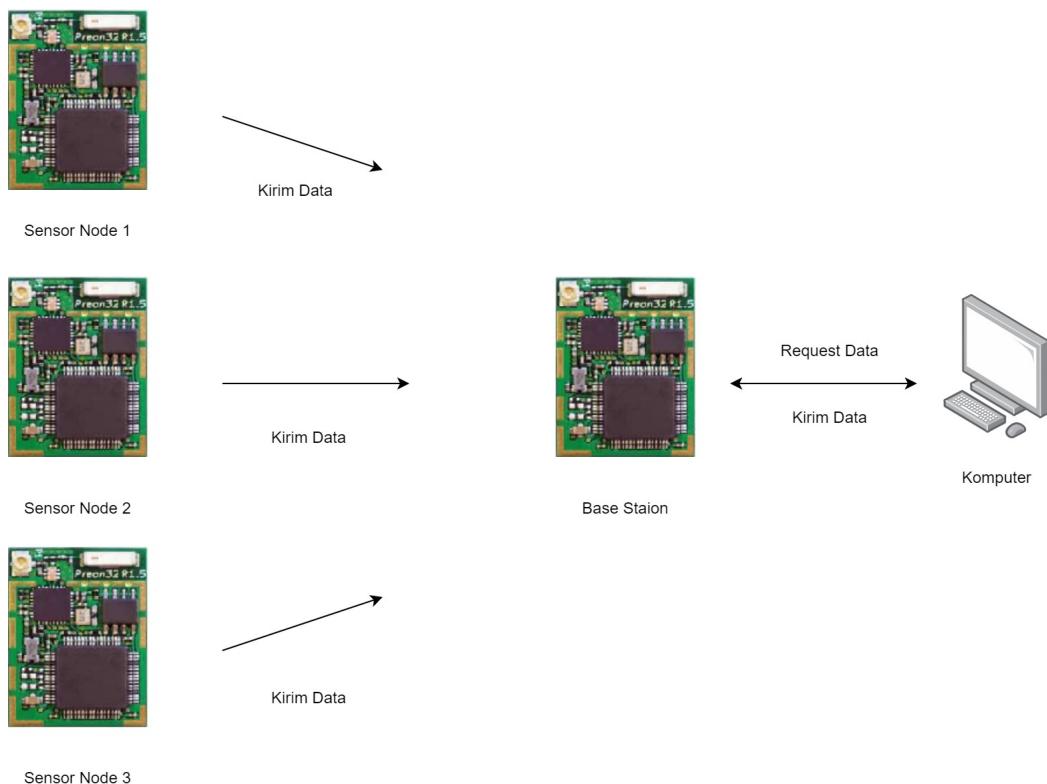
Sensor yang digunakan tidak dibatasi oleh jenis arsitektur WSN yang dapat dipilih. Jenis arsitektur Wireless Sensor Network flat ataupun hirarkikal, dapat digunakan dalam perangkat lunak yang dibangun. Jenis komunikasi *single-hop* ataupun *multi-hop* dapat digunakan pada arsitektur hirarkikal seperti yang dibahas pada subbab 2.3.3.

#### 3.1.1 Analisis Arsitektur dan Topologi *Wireless Sensor Network*

Aplikasi yang dibangun untuk melakukan *sensing* terhadap gedung yang diteliti dan melakukan komunikasi agar data yang ada dapat dikirimkan. Berdasarkan pembahasan tentang arsitektur *Wireless Sensor Network* yang dibahas pada subbab 2.3.3. Aplikasi ini akan menggunakan arsitektur bertipe flat untuk implementasi. Pemilihan arsitektur flat karena wilayah pengujian gedung yang tidak terlalu luas dan tugas dari setiap node yang disebar memiliki tugas yang sama. Sedangkan berdasarkan pembahasan tentang topologi pada subbab 2.3.4, topologi yang diuji untuk membangun aplikasi ini adalah topologi *star* ataupun *linear*. Topologi *linear* dipilih untuk dilakukan pengujian karena topologi *linear* adalah dasar dari komunikasi antar node. Pada topologi *star*, central node akan berfungsi sebagai *sink node* atau dapat dikatakan sebagai *base station* sedangkan node lainnya akan memiliki tugas untuk melakukan *sensing* (Gambar 3.1).



Gambar 3.1: *Wireless Sensor Network* dengan Topologi *Star*



Gambar 3.2: Arsitektur dan Topologi WSN

### 3.1.2 Analisis Accelerometer

*Accelerometer* yang akan digunakan dalam penelitian ini adalah *accelerometer* jenis ADXL345 dari sensor node yang berasal dari Preon32 yang ada pada subbab 2.6. Jenis *accelerometer* dari Preon32 adalah *three-axis* dan *absolute accelerometer* yang telah dibahas. Hasil *sensing* dari sensor *accelerometer* adalah percepatan dimana dan terdiri dari 3 sumbu yaitu x, y, dan z. Rentang nilai dari percepatan dari sensor ini adalah -255 sampai 255. Nilai percepatan akan dikonversi ke dalam satuan gravitasi (g) dan proses konversi sudah ditangani oleh *class library* yang dimiliki oleh PreonVM yang dimana daftar *class library*nya dapat dilihat pada subbab 2.6.4.

### 3.1.3 Analisis Fungsi Aplikasi

Aplikasi yang akan dibangun memiliki fungsi-fungsi utama sebagai berikut:

1. Memeriksa status pada setiap node sensor

Fungsi memeriksa status pada setiap node sensor bertujuan untuk memeriksa apakah setiap node aktif atau tidak sebelum melakukan *sensing*. Jika ada sensor node yang statusnya tidak aktif, maka aplikasi akan memberikan informasi berupa adanya node yang tidak aktif dan pengguna dapat melakukan perbaikan atau *maintenance*.

2. Melakukan sinkronisasi waktu pada setiap node sensor

Fungsi melakukan sinkronisasi waktu pada setiap node sensor diperlukan agar setiap node mengirimkan data hasil *sensing* secara bersamaan atau pada waktu yang sama. Selain itu, *memory* yang ada di setiap node sensor memiliki sifat sementara atau *volatile* (subbab 2.3.2). Hal ini menyebabkan adanya kebutuhan untuk menyamakan waktu di setiap node sensor. Karena aplikasi mencatat waktu kapan sensor node itu mati sehingga perlu adanya penyamaan waktu.

3. Mengirimkan perintah yang bertujuan untuk melakukan *sensing*(pengukuran) pada setiap node

Fungsi mengirimkan perintah yang bertujuan untuk melakukan *sensing* pada setiap node digunakan untuk memulai proses *sensing* getaran pada setiap node dengan sensor *accelerometer*. Setiap hasil pengukuran tersebut akan langsung diekstrasi dan dikirimkan ke *base station*.

4. Mengirimkan perintah yang bertujuan untuk memberhentikan *sensing*(pengukuran) pada setiap node

Fungsi mengirimkan perintah yang bertujuan untuk memberhentikan *sensing* pada setiap node digunakan untuk memberhentikan proses *sensing* getaran pada sensor node.

5. Mengubah data hasil *sensing* dari sinyal analog menjadi digital

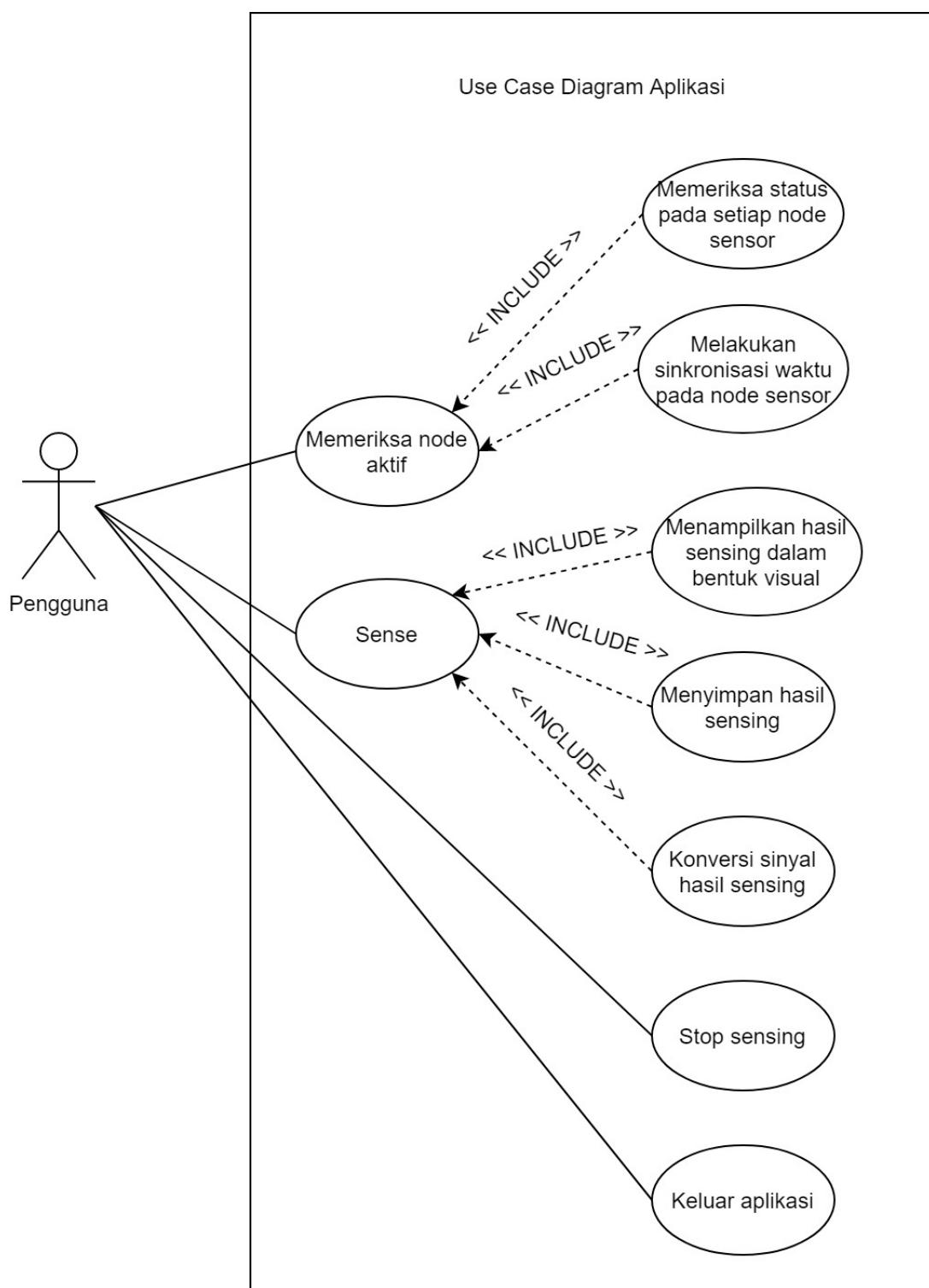
Fungsi mengubah data hasil sensing dari analog ke digital digunakan untuk mengubah sinyal hasil sensing yang berupa analog menjadi digital (dapat dilihat dengan bentuk angka). Hasil konversi ini pun akan dikirimkan langsung ke *base station* dan akan disimpan ke server atau localhost. Kemudian *base station* akan meneruskan hasilnya ke komputer.

6. Mengirimkan data hasil *sensing* ke *base station*

7. Menyimpan data hasil *sensing* yang diterima oleh *base station*

8. Menampilkan data hasil *sensing* yang disimpan oleh *base station*

Fungsi yang terakhir merupakan fungsi yang bertujuan untuk menampilkan hasil *sensing* nya. Fungsi ini dapat dikatakan sebagai fungsi antarmuka dan antarmuka pada aplikasi ini akan menampilkan data hasil *sensing* pada layar komputer pengguna, selain itu juga, antarmuka digunakan untuk melakukan komunikasi antara *base station* dan *node sensor*



Gambar 3.3: Diagram Use Case Aplikasi

Tabel 3.1: Tabel Skenario Memeriksa node aktif

Nama	Memeriksa node aktif
Deskripsi	Memeriksa status node sensor aktif atau tidak dan melakukan penyeamaan waktu sensor node dengan komputer pengguna
Aktor	Pengguna
Pre-kondisi	Aplikasi baru dibuka oleh pengguna
Alur Skenario Utama	<ol style="list-style-type: none"> <li>1. Sistem menjalankan aplikasi.</li> <li>2. Pengguna memasukan banyak node yang dipakai.</li> <li>3. Sistem menampilkan pilihan fungsi yang dapat digunakan.</li> <li>4. Pengguna memilih fungsi "Memeriksa node aktif".</li> <li>5. Sistem akan menampilkan <i>sensor node</i> dan <i>base station</i> yang memiliki status online dan waktu pada setiap <i>sensor node</i>.</li> </ol>
Pos-kondisi	Aplikasi menampilkan sensor-sensor node yang aktif

Tabel 3.2: Tabel Skenario Memberi perintah sense ke sensor node

Nama	Memberikan perintah sense ke sensor node / Sense
Deskripsi	Memberikan perintah sense ke setiap node sensor dan hasil dari setiap node sensor yang aktif kemudian akan disimpan dan ditampilkan dalam bentuk digital
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah berjalan dan fungsi "Memeriksa node aktif" sudah dijalankan
Alur Skenario Utama	<ol style="list-style-type: none"> <li>1. Sistem menampilkan pilihan-pilihan fungsi yang dapat dipilih oleh pengguna.</li> <li>2. Pengguna memilih fungsi "Sense".</li> <li>3. Sistem akan menampilkan hasil pengukuran (sense) dari setiap node sensor dan menyimpan hasilnya dalam hasil konversi menjadi digital dari setiap node.</li> </ol>
Pos-kondisi	Aplikasi melakukan sensing dan menampilkan grafik hasil sensing

Tabel 3.3: Tabel Skenario Stop Sensing

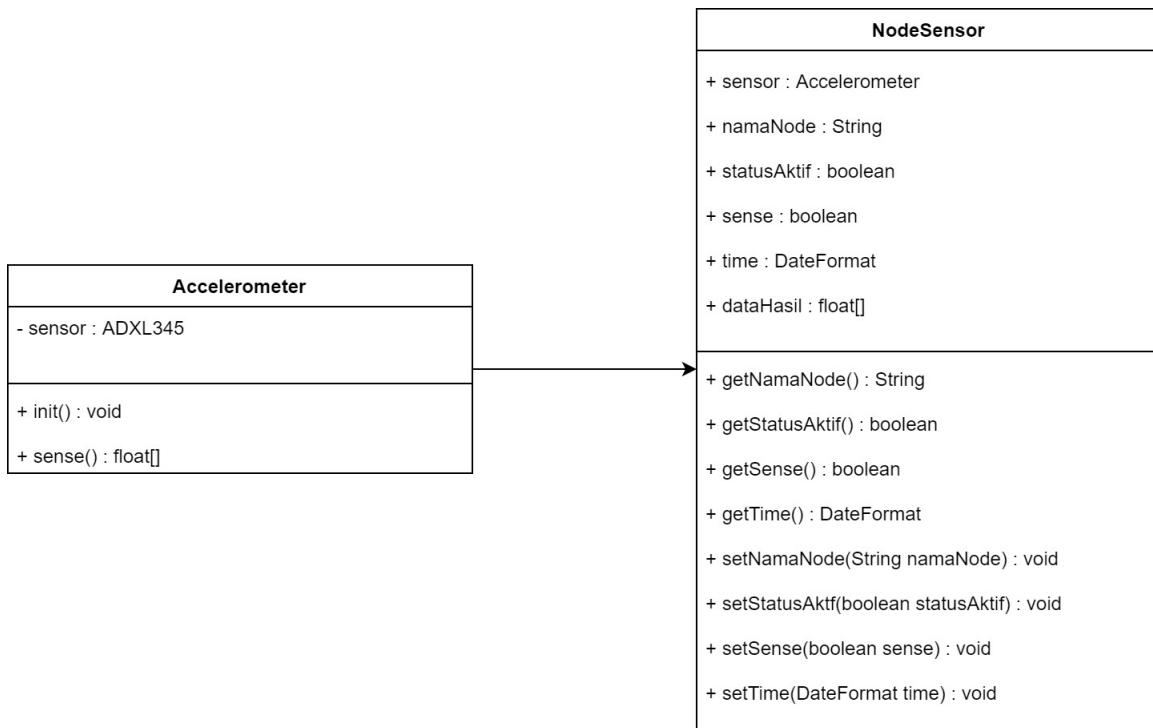
Nama	Memberikan perintah untuk berhenti melakukan sensing (Stop Sensing)
Deskripsi	Memberikan perintah berhenti sensing ke setiap node sensor
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah berjalan dan fungsi aplikasi "Sense" dijalankan
Alur Skenario Utama	<ol style="list-style-type: none"> <li>1. Sistem menampilkan pilihan-pilihan fungsi yang dapat dipilih oleh pengguna.</li> <li>2. Pengguna memilih fungsi "Stop Sensing".</li> <li>3. Sistem akan menjalankan fungsi dan menampilkan pesan tunggu sampai fungsi selesai dijalankan.</li> <li>4. Semua sensor node akan berhenti melakukan sensing dan tampilan hasil sense akan tertutup dan menampilkan pesan "Sense has stopped".</li> </ol>
Pos-kondisi	Aplikasi berhenti melakukan sensing dan grafik sensing tidak diupdate

Tabel 3.4: Tabel Skenario Keluar aplikasi

Nama	Keluar aplikasi
Deskripsi	Memberikan perintah keluar aplikasi untuk semua node sensor dan base station
Aktor	Pengguna
Pre-kondisi	Aplikasi sedang berjalan
Alur Skenario Utama	<ol style="list-style-type: none"> <li>1. Sistem menampilkan pilihan-pilihan fungsi yang dapat dipilih oleh pengguna.</li> <li>2. Pengguna memilih fungsi "Keluar aplikasi".</li> <li>3. Sistem menampilkan pesan "aplikasi telah dihentikan".</li> <li>4. Aplikasi akan berhenti.</li> </ol>
Pos-kondisi	Aplikasi akan tertutup

### 3.1.4 Analisis Kelas

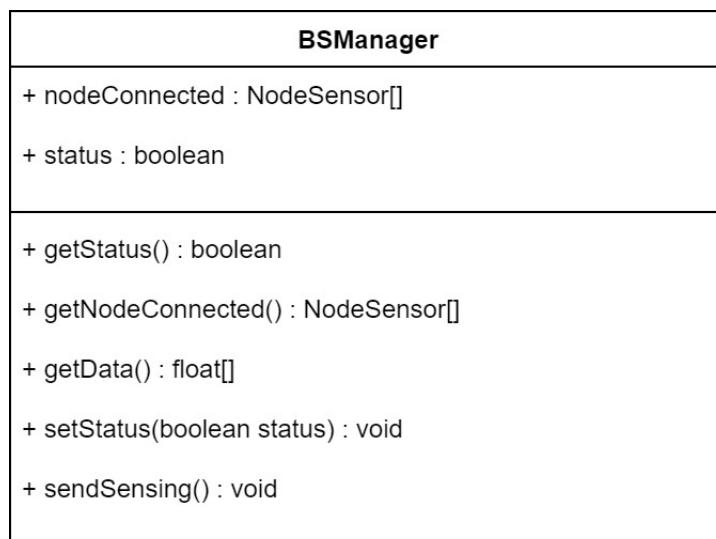
kelas) Pembuatan aplikasi pemantauan getaran gedung menggunakan IDE yaitu Eclipse dan SandBox untuk sensor node Preon32 yaitu salah satu jenis PreonVM yang berasal dari Virtenio. Berdasarkan subbab 3.1.3 yang membahas tentang analisis fungsi aplikasi, maka dibuat diagram kelas untuk pembuatan aplikasi ini. Berikut diagram kelas yang dibutuhkan untuk aplikasi ini.



Gambar 3.4: Diagram Kelas NodeSensor

Penjelasan Kelas Accelerometer dan NodeSensor:

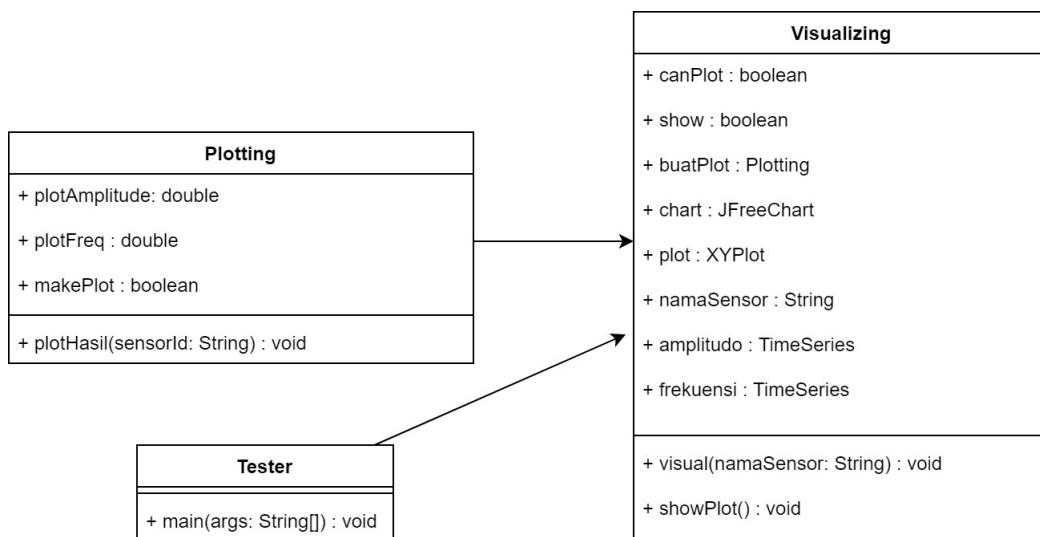
- Kelas Accelerometer Pada kelas Accelerometer terdapat sebuah atribut yaitu sensor dengan tipe data ADXL345 dimana fungsi atribut ini adalah mempresentasikan jenis sensor yang akan digunakan. Sedangkan pada methodnya terdapat dua method yaitu init() dan sense(). Method init() digunakan untuk menginisialisasi kelas Accelerometer dan method sense() digunakan untuk melakukan sensing (pengukuran).
- Kelas NodeSensor Pada Kelas NodeSensor terdapat beberapa atribut berupa sensor dimana atribut ini akan merujuk ke kelas Accelerometer, kemudian atribut namaNode untuk memberikan identitas terhadap node, statusAktif digunakan untuk memberikan kondisi node sensor sedang aktif atau tidak, atribut sense ada untuk memberikan kondisi apakah node sensor sedang melakukan *sensing*(pengukuran) atau tidak. Atribut time ada untuk mencatat waktu agar node sensor dapat sama ketika melakukan *sensing*. Atribut dataHasil digunakan untuk mencatat hasil pengukuran yang telah dilakukan oleh node sensor. Terdapat juga beberapa method-method untuk kelas NodeSensor berupa *setter* dan *getter* untuk atribut-atribut yang ada pada kelas NodeSensor.



Gambar 3.5: Diagram Kelas BaseStation

Penjelasan Kelas BSManager:

- Kelas BSManager Pada Kelas BaseStation, terdapat dua atribut yaitu nodeConnected dimana ada untuk mengetahui Base Station terhubung ke node yang mana saja kemudian ada atribut status dimana ada untuk mengetahui kondisi Base Station apakah aktif atau tidak. Terdapat beberapa method juga pada kelas BSManager yaitu *setter* dan *getter* untuk atributnya kemudian ada method getData() dimana ada untuk mengambil data hasil *sensing* dari node-node yang terhubung. Method sendSensing() digunakan untuk mengirimkan hasil dari method getData() untuk dikirimkan ke komputer pengguna.



Gambar 3.6: Diagram Kelas Tester

Penjelasan Kelas Plotting, Visualizing dan Tester:

- Kelas Plotting Pada Kelas Plotting, terdapat tiga atribut yaitu plotAmplitude dimana ada untuk mencatat nilai amplitude dari plot, kemudian plotFreq ada untuk mencatat nilai frequency dari plot sedangkan atribut makePlot ada untuk penanda pembuatan plot. Untuk method terdapat 1 method yaitu plotHasil(sensorId) dimana method digunakan untuk membuat plot pada visualisasi yang sedang berjalan
- Kelas Visualizing Pada Kelas Visualizing, terdapat beberapa atribut yang ada antaranya

canPlot digunakan untuk penanda bahwa dapat dilakukan plot atau tidak, kemudian atribut show digunakan untuk penanda apakah sudah ditampilkan atau belum, atribut buatPlot digunakan untuk membuat objek Plotting, atribut chart digunakan untuk membuat objek JFreeChart, atribut plot digunakan untuk mengatur plot yang akan divisualisasikan, atribut namaSensor dibuat untuk menyimpan nama sensor, amplitudo digunakan untuk membuat plot time series dari amplitudo dan atribut frekuensi digunakan untuk membuat plot time series dari frekuensi. Method yang ada pada kelas Visualizing adalah visual(namaSensor), method ini digunakan untuk membuat konstruktor dari kelas Visualizing dan method showPlot() digunakan untuk menampilkan plot pada aplikasi.

- Kelas Tester Pada kelas Tester, terdapat hanya 1 method yaitu main(args: String[]) dimana method ini digunakan untuk menjalankan aplikasi secara keseluruhan.

### 3.1.5 Analisis Paket/Pesan

Hal pertama yang akan dilakukan adalah sensor node akan disebarluaskan di sekitar bagian gedung, dan melakukan konfigurasi waktu pada setiap node agar waktu mulai sensing dan pengiriman data juga dilakukan bersama. Setelah itu, sensor node akan dinyalakan dan akan dilakukan pengecekan node yang aktif. Cara pengecekan node yang aktif dengan cara mengirimkan pesan yang dikirimkan oleh *base station* ke setiap node. Informasi yang dikirim adalah bahwa node telah aktif. Sistem akan menampilkan pesan bahwa status node sensor ke sekian yang sebelumnya belum aktif, sekarang menjadi aktif. Lalu akan diketahui tujuan data hasil sensing yang akan dikirimkan.

Ketika pengguna memberikan perintah sensing, maka sensor node akan melakukan pengukuran (sensing) dan melakukan pengambilan data. *Base Station* akan mengirimkan pesan berupa perintah untuk melakukan sensing ke sensor node yang ada. Data-data tersebut merupakan data getaran-getaran yang terjadi pada gedung tersebut, kemudian akan dikirimkan ke *base station*. *Base station* akan menerima data hasil sensing dari sensor node secara terus-menerus kemudian akan mengirimkan hasilnya ke aplikasi dan akan ditampilkan oleh aplikasi tersebut.

Format pesan yang digunakan untuk melakukan komunikasi antara pengguna, *basestation*, dan *sensor node* dengan menambahkan tanda "@" pada awal pesan. Fungsi dari tanda "@" ini adalah untuk menandakan bahwa pesan berasal dari *basestation* dan akan dikirimkan ke *sensor node*. Contohnya jika pengguna ingin menjalankan fitur "Check Online Node Status", maka pengguna akan memasukkan angka "1" ke *command line*. Kemudian pesan "1" akan dikirimkan ke *basestation*. Setelah *basestation* menerima kode pesan dari *command line*, *basestation* akan mengirimkan kode pesan "@1" ke setiap sensor node yang terhubung. Kemudian pesan diterima oleh *sensor node*, maka *sensor node* akan menjalankan fungsinya dan mengirimkan pesan hasil fungsi ke *basestation* dengan format ("1 «sensorId» online «waktu sekarang»").



## BAB 4

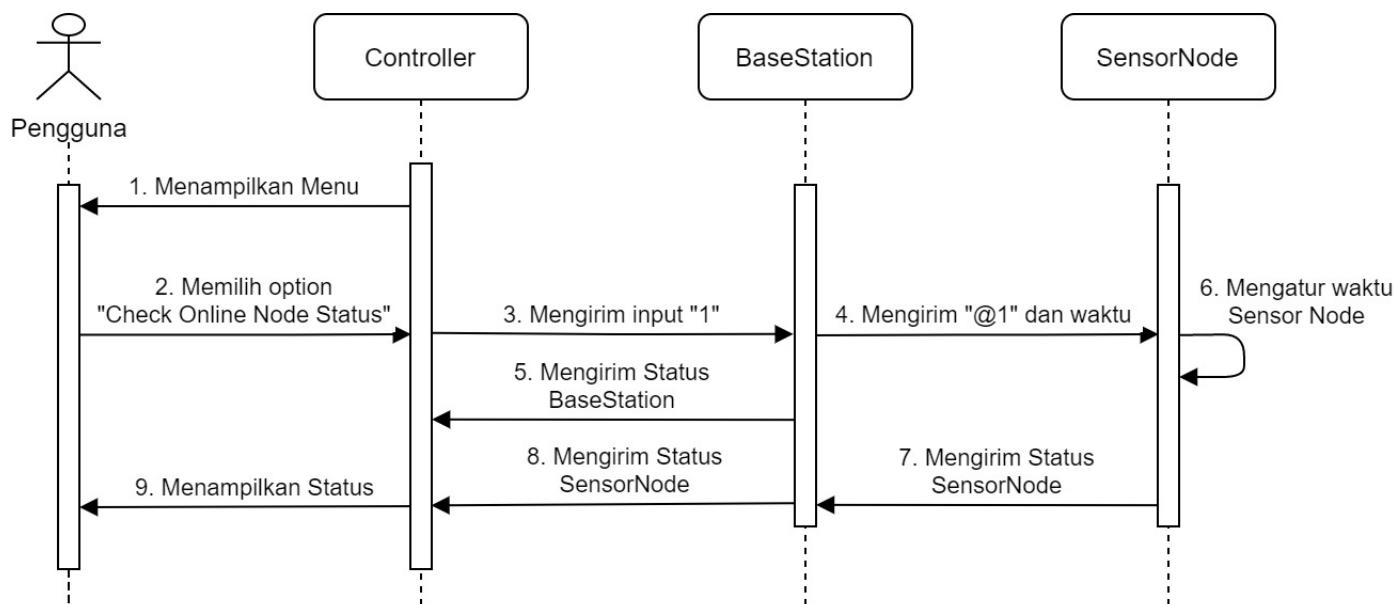
# PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan aplikasi yang dibangun meliputi perancangan interaksi antar node, perancangan aplikasi, format pesan, dan perancangan masukan dan keluaran.

### 4.1 Perancangan Interaksi Antar Node

Berikut penjelasan mengenai fungsi-fungsi aplikasi yang sudah dijelaskan pada analisis subbab 3.1.3

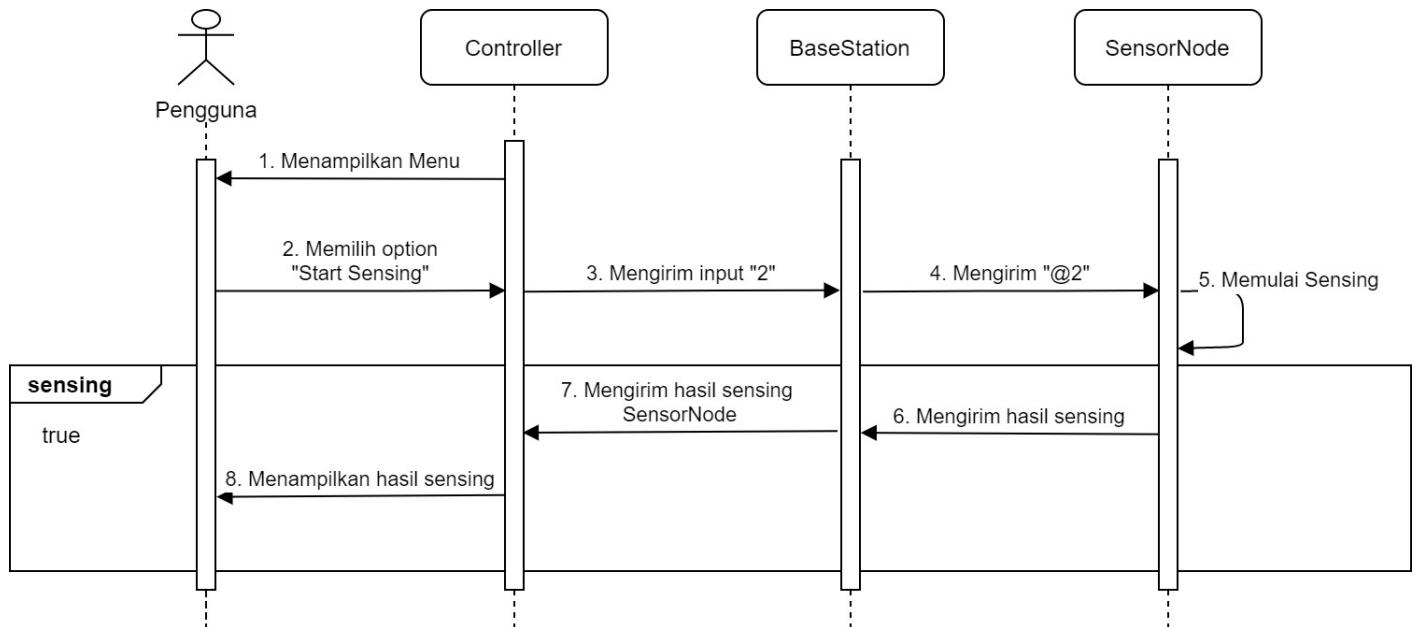
#### 4.1.1 Diagram Sequence "Check Online Node Status"



Gambar 4.1: Diagram Sequence "Check Online Node"

Pertama *Controller* akan menampilkan menu fungsi-fungsi yang ada pada aplikasi. Kemudian pengguna akan memilih option menu "Check Online Node Status". Fungsi "Check Online Node Status" digunakan untuk menyamakan waktu dari setiap *SensorNode* dan *BaseStation* dengan komputer pengguna dan akan menampilkan status online dari *SensorNode* dan *BaseStation*. Pada saat fungsi "Check Online Node Status" berjalan, pengguna akan memasukan kode perintah "1" dan akan dikirimkan ke *BaseStation*. Setelah *BaseStation* menerima kode perintah "1", *BaseStation* akan mengirim kode perintah "@1" ke alamat *SensorNode* yang tersimpan. Setiap *SensorNode* yang menerima kode perintah "@1" akan melakukan pengaturan waktu dan mengirim status *online* ke *BaseStation* secara langsung. Setelah *BaseStation* menerima status *online* dari setiap *SensorNode* maka *BaseStation* akan meneruskan status tersebut ke *Controller* dan pada *Controller* akan menampilkan status *online* dari *SensorNode* tersebut.

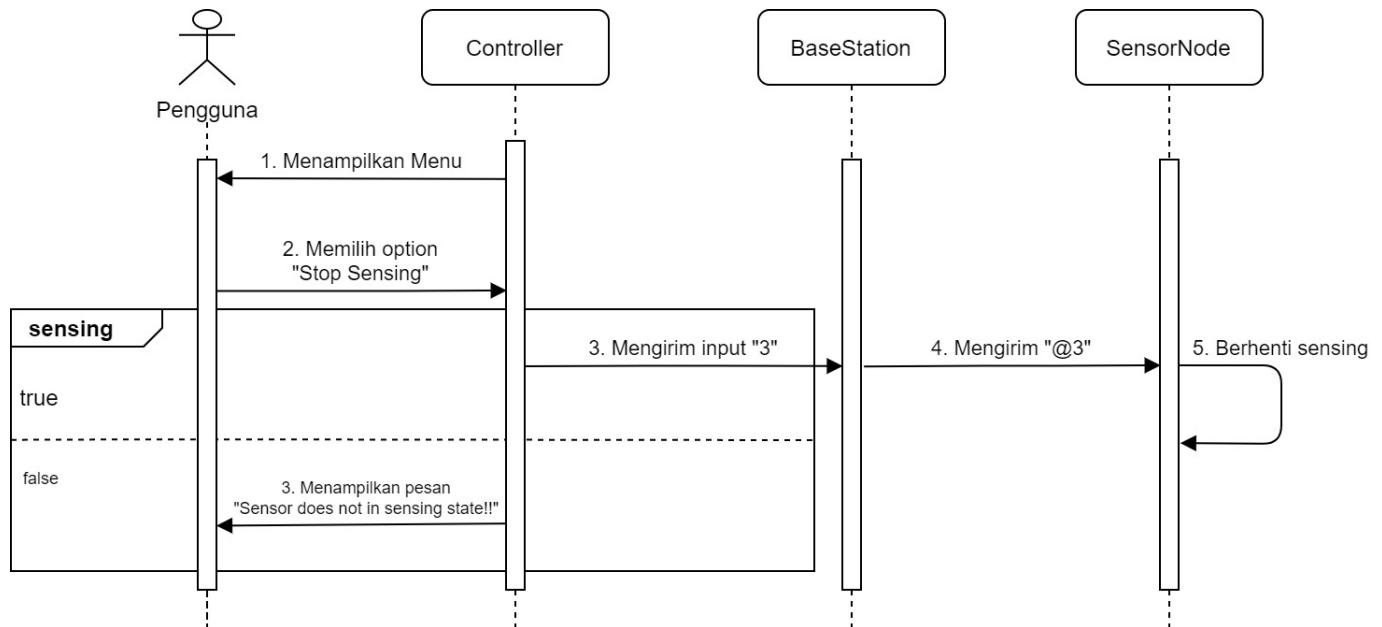
#### 4.1.2 Diagram Sequence "Start Sensing"



Gambar 4.2: Diagram Sequence "Sense"

Pertama *Controller* akan menampilkan menu fungsi-fungsi yang ada pada aplikasi. Setelah pengguna memilih option menu "Check Online Node Status", kemudian pengguna akan memilih option menu "Start Sensing" dengan cara memasukan kode perintah "2". Saat fungsi "Start Sensing" berjalan, *Controller* akan mengirimkan kode perintah "2" ke *BaseStation*. Setelah kode perintah diterima oleh *BaseStation*, *BaseStation* akan mengirimkan kode perintah "@2" ke *SensorNode*. Setiap *SensorNode* yang menerima kode perintah dari *BaseStation* akan melakukan *sensing* dan hasilnya akan dikirimkan ke *BaseStation*. Hasil sensing yang telah diterima oleh *BaseStation* akan diteruskan ke *Controller* dan akan ditampilkan dalam bentuk grafik.

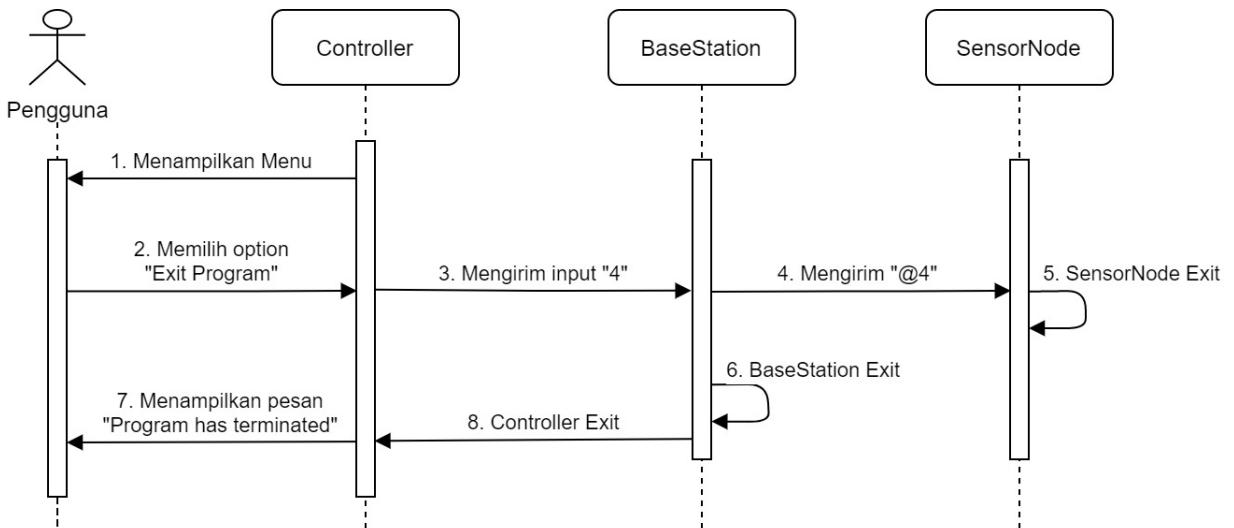
### 4.1.3 Diagram Sequence "Stop Sensing"



Gambar 4.3: Diagram Sequence "Stop Sensing"

Pertama *Controller* akan menampilkan menu fungsi-fungsi yang ada pada aplikasi. Setelah pengguna memilih option menu "Stop Sensing". Jika *SensorNode* tidak sedang dalam keadaan "Sensing" maka *UserApp* akan menampilkan pesan "Sensor does not in sensing state!!". Jika *SensorNode* dalam keadaan "Sensing", maka *Controller* akan mengirimkan kode perintah "3" ke *BaseStation*. Saat *BaseStation* menerima kode perintah, *BaseStation* akan mengirimkan kode perintah "@3" ke *SensorNode*. Setiap *SensorNode* yang menerima kode perintah dari *BaseStation* akan berhenti melakukan "Sensing". Saat semua *SensorNode* berhenti melakukan "Sensing", *Controller* akan menutup grafik hasil sense dan menampilkan pesan "Stop Sensing Done".

### 4.1.4 Diagram Sequence "Exit Program"



Gambar 4.4: Diagram Sequence "Exit"

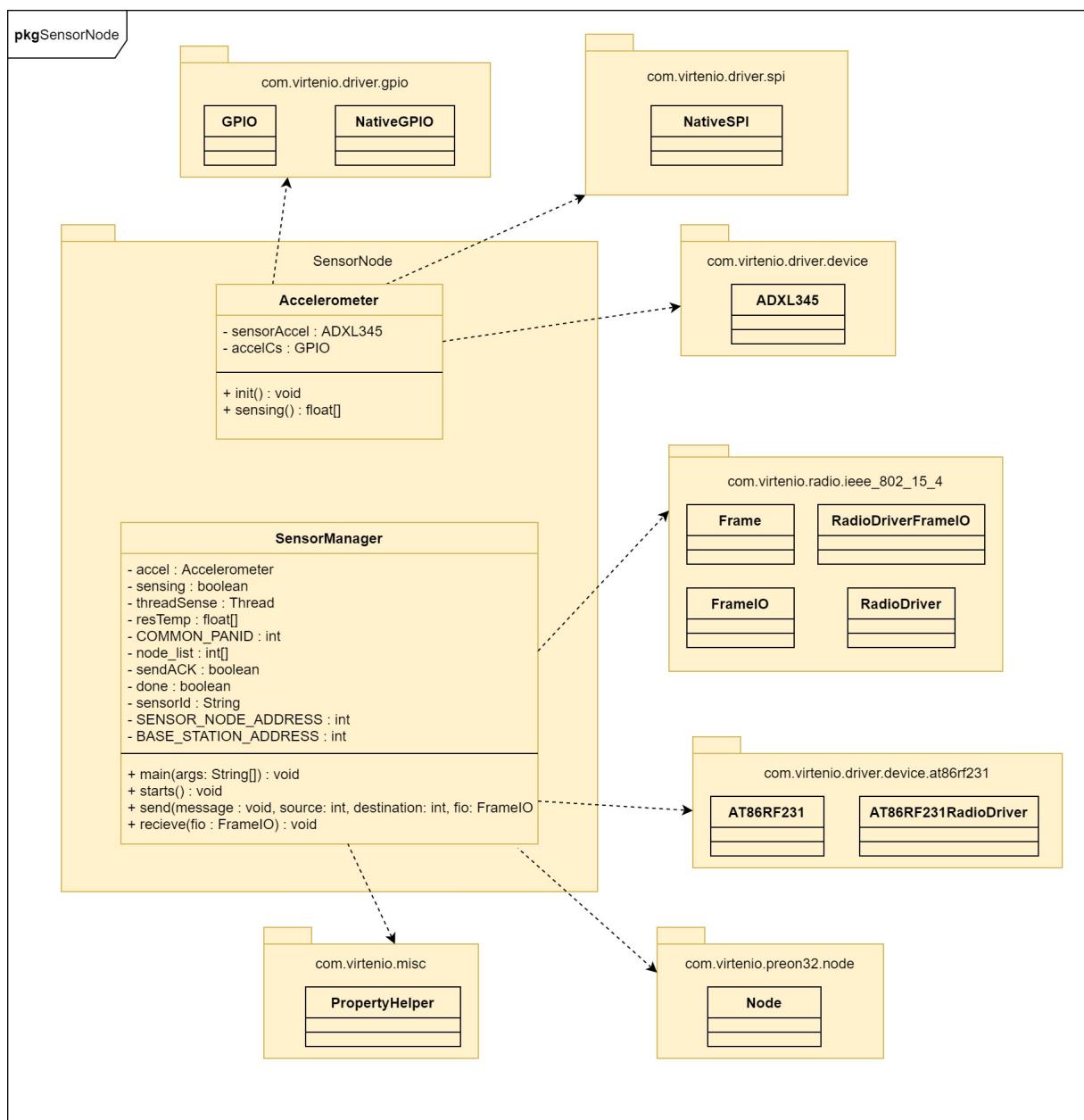
Pertama *Controller* akan menampilkan menu fungsi-fungsi yang ada pada aplikasi. Setelah pengguna memilih option menu "Exit Program". Saat fungsi "Exit Program" berjalan, *Controller* akan mengirimkan kode perintah "4" ke *BaseStation* untuk mematikan prgoram pada *BaseStation*. Setelah *BaseStation* menerima kode perintah, *BaseStation* akan mengirimkan kode perintah "@4" ke *SensorNode*. *SensorNode* yang menerima kode perintah "@4" akan memberhentikan program.

## 4.2 Perancangan Kelas Aplikasi

Pada subbab 3.1.4 kelas) telah dijelaskan kelas diagram sederhana untuk fungsi-fungsi pada aplikasi. Berikut versi lengkap kelas diagram dan detail dari package *SensorNode*, *BaseStation*, dan *Controller*

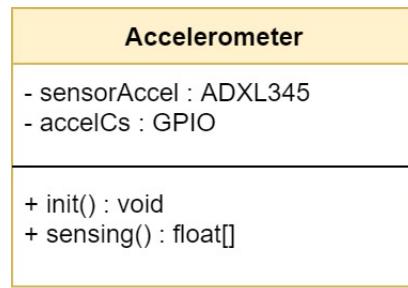
### 4.2.1 Package SensorNode

Pada package ini terdapat kelas *Accelerometer* dan *SensorManager*.



Gambar 4.5: Kelas Diagram lengkap package SensorNode

## Kelas Accelerometer



Gambar 4.6: Perancangan Kelas Accelerometer

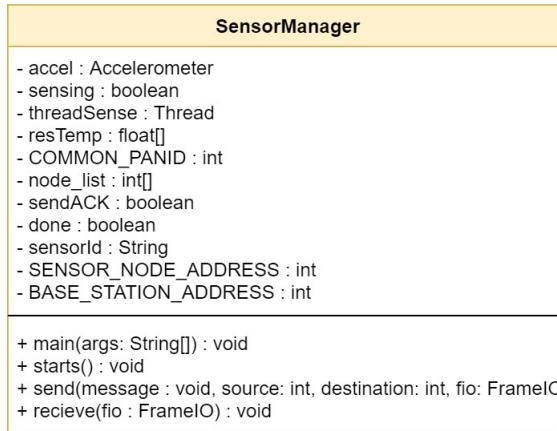
Kelas ini digunakan untuk mengatur dan menginisialisasi sensor *accelerometer* yang terdapat pada sensor node. Kelas ini memiliki atribut-atribut sebagai berikut:

- private ADXL345 sensorAccel;  
Atribut ini digunakan sebagai objek driver untuk *accelerometer* ADXL345.
- private GPIO accelCs;  
Atribut ini digunakan sebagai objek GPIO.

Metode-Metode yang terdapat pada kelas ini adalah sebagai berikut:

- public void init();  
Metode ini digunakan untuk melakukan inisialisasi sensor *accelerometer* ADXL345.
- public float[] sensing;  
Metode ini digunakan untuk mendapatkan data pengukuran *accelerometer* dari ADXL345.

## Kelas SensorManager



Gambar 4.7: Perancangan Kelas SensorManager

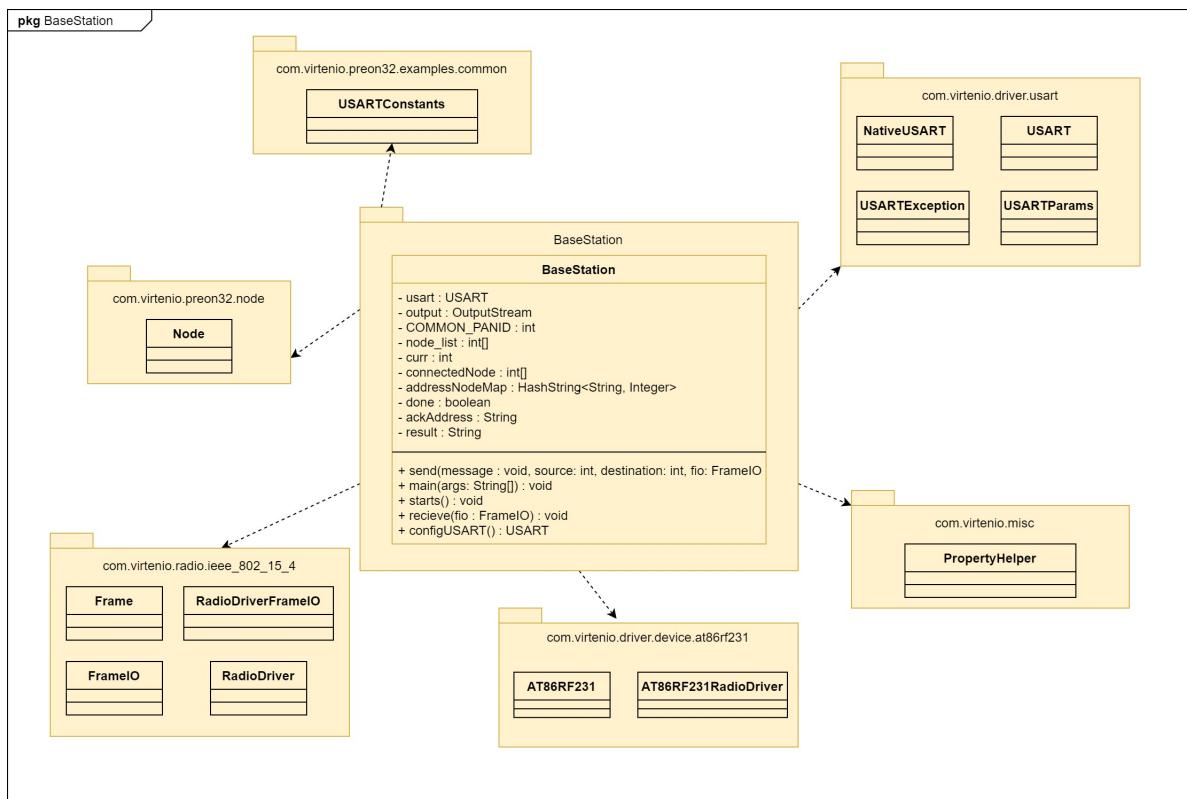
Kelas ini digunakan sebagai main class dari package ini. Kelas ini memiliki fungsi menerima pesan dari BaseStation dan mengirimkan hasil pengukuran ke BaseStation. Kelas ini memiliki atribut-atribut sebagai berikut:

- public static final Accelerometer accel;  
Atribut ini digunakan untuk menyimpan objek dari kelas Accelerometer.
- public static boolean sensing;  
Atribut ini digunakan untuk mengetahui kondisi sensing dari SensorNode, jika bernilai *true* maka SensorNode sedang melakukan sensing.

- public Thread threadSense;  
Atribut ini digunakan untuk membuat *thread* baru untuk melakukan sensing.
  - public static float[] resTemp;  
Atribut ini digunakan untuk menyimpan hasil sementara dari sensing.
  - public static int COMMON\_PANID;  
Atribut ini digunakan untuk menyimpan PANID.
  - public static int[] node\_list;  
Atribut ini digunakan untuk menyimpan alamat-alamat dari sensor node.
  - private static boolean sendACK;  
Atribut ini digunakan sebagai penanda bahwa SensorNode dapat mengirim data ke BaseStation. Jika bernilai true, maka SensorNode dapat mengirimkan data ke BaseStation.
  - private static boolean done;  
Atribut ini digunakan sebagai penanda bahwa SensorNode telah mengirimkan data ke BaseStation. Jika bernilai true, maka SensorNode telah mengirimkan data ke BaseStation.
  - private static final String sensorId;  
Atribut ini digunakan untuk menyimpan Id dari sensor node.
  - private static int SENSOR\_NODE\_ADDRESS;  
Atribut ini digunakan untuk menyimpan alamat dari sensor node.
  - private static int BASE\_STATION\_ADDRESS;  
Atribut ini digunakan untuk menyimpan alamat dari BaseStation.
- Metode-Metode yang terdapat di kelas ini adalah sebagai berikut:
- public static void main(String[] args)  
Metode ini digunakan sebagai metode main dari kelas SensorManager
  - public static void starts();  
Metode ini digunakan sebagai inisialisasi radio dan memulai thread baru untuk menerima pesan dari BaseStation.
  - public static void send(String message, int source, int destination, FrameIO fio);  
Metode ini digunakan untuk mengirim pesan ke BaseStation.
  - public static void recieve(final FrameIO fio);  
Metode ini digunakan untuk menerima pesan dari BaseStation dan mengolah perintah yang diberikan.

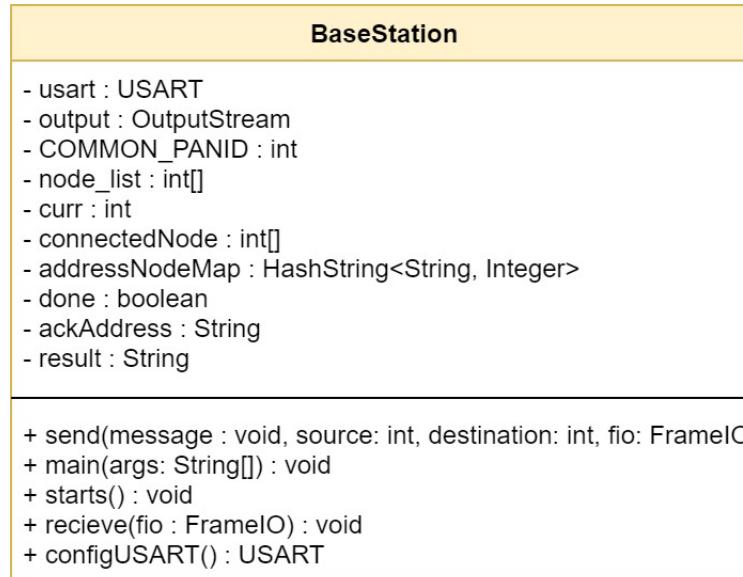
#### 4.2.2 Package BaseStation

Pada package ini terdapat kelas BaseStation



Gambar 4.8: Kelas Diagram lengkap package BaseStation

### Kelas BaseStation



Gambar 4.9: Perancangan Kelas BaseStation

Kelas ini digunakan sebagai main class dari package `BaseStation`. Kelas ini berfungsi untuk menerima perintah dari Controller, meneruskan perintah ke `SensorNode`, menerima pesan dari `SensorNode` dan meneruskan pesan ke Controller. Kelas ini memiliki atribut-atribut sebagai berikut:

- private static USART `uart`;
- Atribut ini digunakan mengatur `uart` pada `BaseStation`.

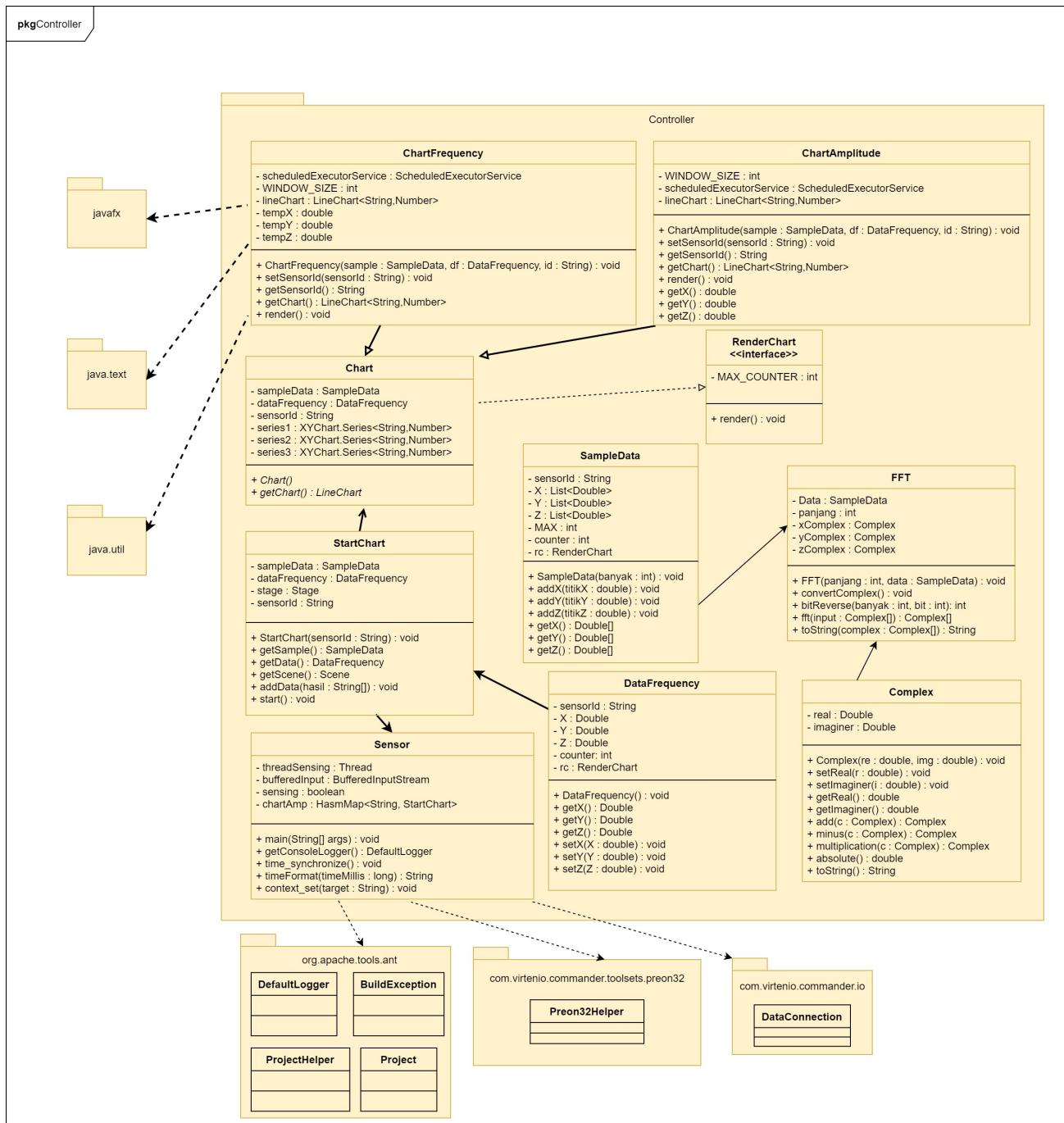
- private static OutputStream output;  
Atribut ini digunakan untuk mengatur OutputStream ke Controller.
- public static COMMON\_PANID;  
Atribut ini digunakan untuk menyimpan PANID.
- public static int[] node\_list;  
Atribut ini digunakan untuk menyimpan alamat-alamat dari sensor node.
- private static int curr;  
Atribut ini digunakan untuk menyimpan alamat dari BaseStation.
- private static int[] connectedNode;  
Atribut ini digunakan untuk menyimpan alamat sensor node yang terhubung dengan BaseStation.
- public static HashMap<String, Integer> addressNodeMap;  
Atribut ini digunakan untuk menyimpan alamat sensor node yang terhubung dengan BaseStation ke HashMap.
- private static boolean done;  
Atribut ini digunakan untuk menunjukkan apakah SensorNode telah mengirim pesan ke BaseStation.
- private static String ackAddress;  
Atribut ini digunakan untuk menyimpan alamat sensor node yang sedang mengirim data.
- private static String result;  
Atribut ini digunakan untuk menyimpan data yang terkirim oleh sensor node.

Metode-Metode yang terdapat pada kelas ini adalah sebagai berikut:

- public static void send(String message, int source, int destination, FrameIO fio);  
Metode ini digunakan untuk mengirim pesan ke SensorNode.
- public static void main(String[] args);  
Metode ini digunakan untuk metode main dari BaseStation
- public static void starts();  
Metode ini digunakan untuk inisialisasi radio dan memulai thread baru untuk menerima dan mengirim pesan.
- public static void recieve(final FrameIO fio);  
Metode ini digunakan untuk menerima pesan yang dikirim oleh SensorNode.

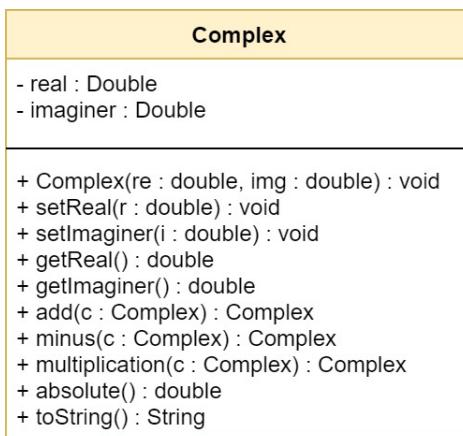
#### 4.2.3 Package Controller

Pada package ini terdapat kelas yang dapat dilihat pada gambar dibawah:



Gambar 4.10: Kelas Diagram lengkap package BaseStation

## Kelas Complex



Gambar 4.11: Perancangan Kelas Complex

Kelas ini digunakan sebagai objek yang mengatur dan menginisialisasi bilangan kompleks. Kelas ini memiliki atribut-atribut sebagai berikut:

- public double real;

Atribut ini digunakan untuk menyimpan bilangan real.

- public double imaginer;

Atribut ini digunakan untuk menyimpan bilangan imajiner.

Metode-Metode yang terdapat pada kelas ini adalah sebagai berikut:

- public Complex(double re, double img);

Metode ini digunakan sebagai konstruktor dari kelas Complex.

- public void setReal(double r);

Metode ini digunakan sebagai setter dari atribut real.

- public void setImaginer(double i);

Metode ini digunakan sebagai setter dari atribut imaginer.

- public double getReal();

Metode ini digunakan sebagai getter dari atribut real.

- public double getImaginer();

Metode ini digunakan sebagai getter dari atribut imaginer.

- public Complex add(Complex c);

Metode ini digunakan untuk melakukan penjumlahan antar bilangan kompleks.

- public Complex minus(Complex c);

Metode ini digunakan untuk melakukan pengurangan antar bilangan kompleks.

- public Complex multiplication(Complex c);

Metode ini digunakan untuk melakukan perkalian antar bilangan kompleks.

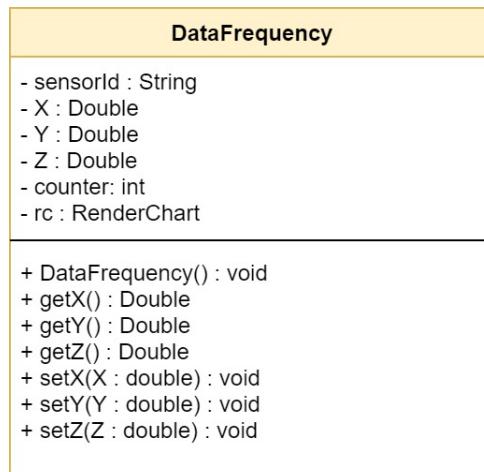
- public double absolute();

Metode ini digunakan untuk mendapatkan nilai amplitude dari bilangan kompleks.

- public String toString();

Metode ini digunakan untuk menjadikan bilangan kompleks menjadi *string*.

#### 4.2.4 Kelas DataFrequency



Gambar 4.12: Perancangan Kelas DataFrequency

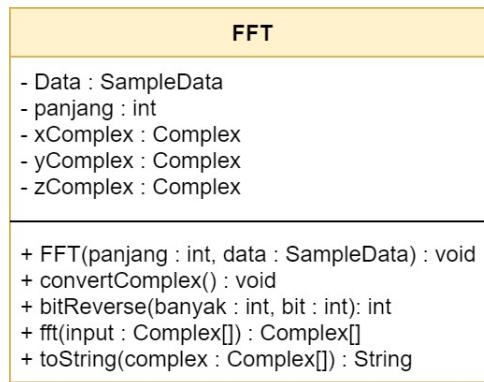
Kelas ini digunakan untuk menyimpan hasil perhitungan FFT. Kelas ini memiliki atribut-atribut sebagai berikut:

- public String sensorId;  
Atribut ini digunakan untuk menyimpan nama dari sensor yang digunakan.
- public Double X;  
Atribut ini digunakan untuk menyimpan hasil perhitungan FFT pada sumbu X.
- public Double Y;  
Atribut ini digunakan untuk menyimpan hasil perhitungan FFT pada sumbu Y.
- public Double Z;  
Atribut ini digunakan untuk menyimpan hasil perhitungan FFT pada sumbu Z.
- private int counter=0;  
Atribut ini digunakan untuk menyimpan banyaknya nilai FFT yang telah disimpan dalam atribut.
- private RenderChart rc;  
Atribut ini digunakan untuk menyimpan objek dari kelas RenderChart.

Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:

- public DataFrequency();  
Metode ini merupakan konstruktor dari kelas DataFrequency.
- public void setRender(RenderChart rc);  
Metode ini merupakan *setter* dari atribut rc.
- public Double getX();  
Metode ini merupakan *getter* dari atribut X.
- public Double getY();  
Metode ini merupakan *getter* dari atribut Y.
- public Double getZ();  
Metode ini merupakan *getter* dari atribut Z.
- public void setX();  
Metode ini merupakan *setter* dari atribut X.
- public void setY();  
Metode ini merupakan *setter* dari atribut Y.
- public void setY();  
Metode ini merupakan *setter* dari atribut Z.

## Kelas FFT



Gambar 4.13: Perancangan Kelas FFT

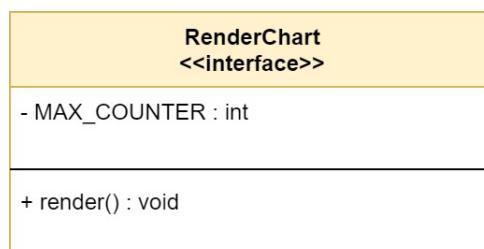
Kelas ini digunakan untuk melakukan komputasi FFT dari input bilangan kompleks. Kelas ini memiliki atribut-atribut sebagai berikut:

- public SampleData data;  
Atribut ini digunakan untuk menyimpan sampel data.
- public int panjang;  
Atribut ini digunakan untuk menyimpan panjang dari algoritma FFT.
- public Complex xComplex;  
Atribut ini digunakan untuk menyimpan nilai bilangan kompleks pada sumbu x.
- public Complex yComplex;  
Atribut ini digunakan untuk menyimpan nilai bilangan kompleks pada sumbu y.
- public Complex zComplex;  
Atribut ini digunakan untuk menyimpan nilai bilangan kompleks pada sumbu z.

Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:

- public FFT(int panjang, SampleData data);  
Metode ini digunakan sebagai konstruktor dari kelas FFT.
- public void convertComplex();  
Metode ini digunakan untuk menyimpan hasil sensing ke dalam bilangan kompleks.
- public int bitReverse(int banyak, int bit);  
Metode ini digunakan untuk melakukan *bit reverse* pada bilangan masukan.
- public Complex[] fft(Complex[] input);  
Metode ini digunakan untuk melakukan komputasi algoritma FFT.
- public String toString(Complex[] complex);  
Metode ini digunakan untuk mengubah bilangan kompleks menjadi *String*.

### 4.2.5 Kelas RenderChart

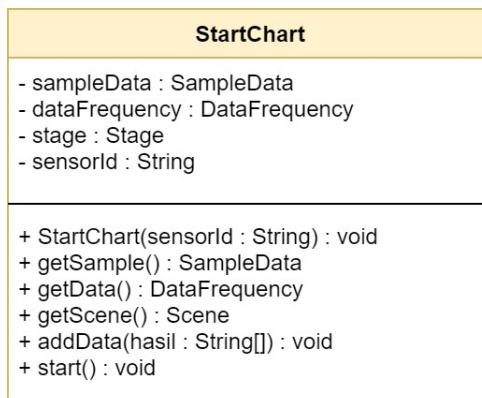


Gambar 4.14: Perancangan Kelas RenderChart

Kelas ini merupakan *interface class* yang akan digunakan oleh kelas StartChart<sup>4.15</sup> untuk melakukan visualisasi grafik. Pada kelas ini, terdapat atribut beserta metode yang akan digunakan oleh kelas lain adalah sebagai berikut:

- public static final int MAX\_COUNTER=64;  
Atribut digunakan untuk membatasi limitasi banyak data yang akan digunakan untuk melakukan perhitungan.
- public void render();  
Metode ini digunakan untuk melakukan proses visualisasi dalam grafik.

#### 4.2.6 Kelas StartChart



Gambar 4.15: Perancangan Kelas StartChart

Kelas ini digunakan untuk melakukan visualisasi grafik. Kelas ini memiliki atribut-atribut sebagai berikut:

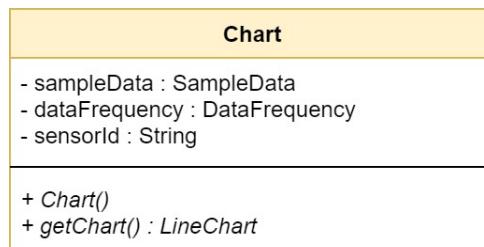
- public ArrayList<Chart> charts;  
Atribut ini digunakan untuk menyimpan grafik yang akan ditampilkan secara visual.
- private SampleData data;  
Atribut ini digunakan untuk menyimpan sampel data.
- private DataFrequency dataFrequency;  
Atribut ini digunakan untuk menyimpan nilai frekuensi hasil FFT.
- private Stage stage;  
Atribut ini digunakan untuk membuat objek Stage yang digunakan untuk melakukan visualisasi grafik.
- private String sensorId;  
Atribut ini digunakan untuk menyimpan id dari sensor.

Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:

- public StartChart(String sensorId);  
Metode ini digunakan sebagai konstruktor dari kelas StartChart.
- public SampleData getSample();  
Metode ini digunakan untuk mendapatkan objek dari kelas SampleData.
- public DataFrequency getData();  
Metode ini digunakan untuk mendapatkan objek dari kelas DataFrequency.
- public Scene getScene();  
Metode ini digunakan mengatur bagaimana grafik yang akan ditampilkan akan terlihat.
- public void addData();  
Metode ini digunakan untuk menyimpan nilai amplitudo serta nilai FFT yang telah didapatkan setelah melakukan perhitungan.
- public void start();

Metode ini digunakan untuk menjalankan kelas ini sehingga grafik yang akan divisualisasikan berhasil terlihat.

#### 4.2.7 Kelas Chart

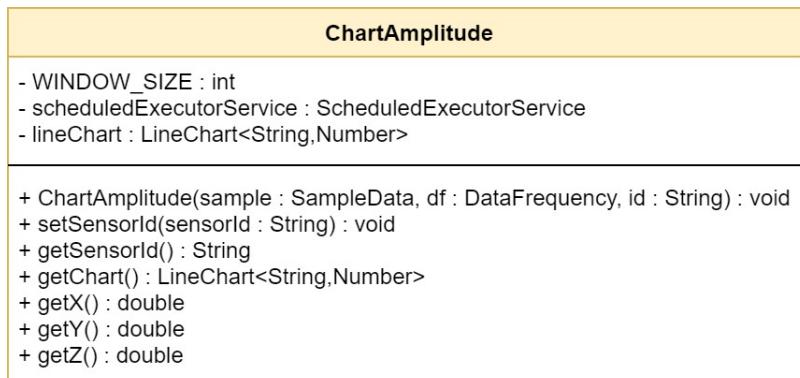


Gambar 4.16: Perancangan Kelas Chart

Kelas ini merupakan *abstract class* yang merupakan *implements* dari kelas RenderChart. Kelas ini akan digunakan oleh kelas lain untuk melakukan visualisasi. Kelas ini memiliki atribut-atribut sebagai berikut:

- public SampleData sampleData;  
Atribut ini digunakan sebagai objek dari kelas SampleData.
  - public DataFrequency dataFrequency;  
Atribut ini digunakan sebagai objek dari kelas DataFrequency.
  - public String sensorId;  
Atribut ini digunakan untuk menyimpan nama dari sensor yang digunakan.
  - XYChart.Series<String,Number> series1 = new XYChart.Series<>();  
Atribut ini digunakan untuk menginisialisasi kelas XYChart yang akan digunakan untuk membuat sumbu X pada grafik yang akan divisualisasikan.
  - XYChart.Series<String,Number> series2 = new XYChart.Series<>();  
Atribut ini digunakan untuk menginisialisasi kelas XYChart yang akan digunakan untuk membuat sumbu Y pada grafik yang akan divisualisasikan.
  - XYChart.Series<String,Number> series3 = new XYChart.Series<>();  
Atribut ini digunakan untuk menginisialisasi kelas XYChart yang akan digunakan untuk membuat sumbu Z pada grafik yang akan divisualisasikan.
  - final SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");  
Atribut ini digunakan untuk menyimpan format waktu yang akan ditampilkan pada grafik.
- Metode yang terdapat pada kelas ini adalah sebagai berikut:
- public abstract LineChart getChart();  
Metode ini merupakan metode *abstract* yang akan digunakan oleh kelas lain yang berfungsi untuk mendapatkan grafik yang akan ditampilkan.

#### 4.2.8 Kelas ChartAmplitude



Gambar 4.17: Perancangan Kelas ChartAmplitude

Kelas ini merupakan kelas yang akan digunakan untuk menampilkan grafik nilai amplitudo dari sensor. Kelas ini memiliki atribut-atribut sebagai berikut:

- final int WINDOW\_SIZE=10;

Atribut ini digunakan untuk menentukan berapa banyak nilai yang akan ditampilkan dalam grafik, dalam hal ini banyak nilai yang akan ditampilkan adalah 10 nilai.

- private ScheduledExecutorService scheduledExecutorService;

Atribut ini digunakan untuk melakukan update terhadap nilai-nilai amplitudo yang akan ditampilkan ke dalam grafik.

- public LineChart<String,Number> lineChart;

Atribut ini digunakan untuk menyimpan grafik dari amplitudo ini dalam bentuk *line chart*.

Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:

- public ChartAmplitude(SampleData sample, DataFrequency df, String id);

Metode ini digunakan sebagai konstruktor pada kelas ChartAmplitude.

- public void setSensorId(String sensorId);

Metode ini merupakan *setter* dari atribut sensorId.

- public String getSensorId();

Metode ini merupakan *getter* dari atribut sensorId.

- public LineChart<String, Number> getChart();

Metode ini digunakan untuk membuat komponen-komponen yang akan digunakan untuk melakukan visualisasi nilai amplitudo.

- public void render();

Metode ini digunakan untuk memasukan nilai-nilai amplitudo yang akan ditampilkan ke dalam bentuk grafik.

- public double getX();

Metode ini digunakan untuk mendapatkan nilai rata-rata amplitudo pada sumbu X.

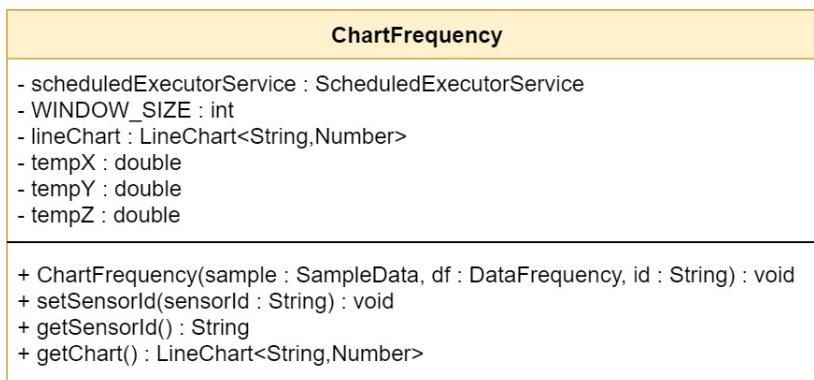
- public double getY();

Metode ini digunakan untuk mendapatkan nilai rata-rata amplitudo pada sumbu Y.

- public double getZ();

Metode ini digunakan untuk mendapatkan nilai rata-rata amplitudo pada sumbu Z.

#### 4.2.9 Kelas ChartFrequency

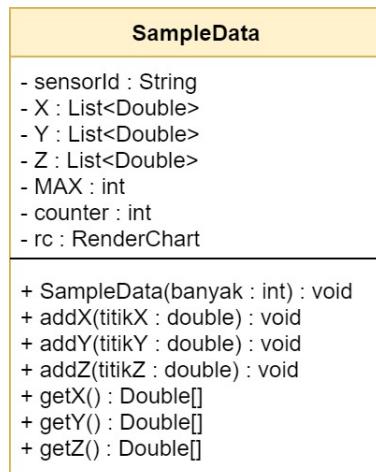


Gambar 4.18: Perancangan Kelas ChartFrequency

Kelas ini merupakan kelas yang akan digunakan untuk menampilkan grafik nilai frekuensi dari sensor. Kelas ini memiliki atribut-atribut sebagai berikut:

- private ScheduledExecutorService scheduledExecutorService;  
Atribut ini digunakan untuk melakukan update terhadap nilai-nilai frekuensi yang akan ditampilkan ke dalam grafik.
  - final int WINDOW\_SIZE=10;  
Atribut ini digunakan untuk menentukan berapa banyak nilai yang akan ditampilkan dalam grafik, dalam hal ini banyak nilai yang akan ditampilkan adalah 10 nilai.
  - public LineChart<String,Number> lineChart;  
Atribut ini digunakan untuk menyimpan grafik dari frekuensi ke dalam bentuk *line chart*.
  - public static double tempX;  
Atribut ini digunakan untuk menyimpan nilai hasil perhitungan FFT pada sumbu X.
  - public static double tempY;  
Atribut ini digunakan untuk menyimpan nilai hasil perhitungan FFT pada sumbu Y.
  - public static double tempZ;  
Atribut ini digunakan untuk menyimpan nilai hasil perhitungan FFT pada sumbu Z.
- Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:
- public ChartFrequency(SampleData sample, DataFrequency df, String id);  
Metode ini digunakan sebagai konstruktor pada kelas ChartFrequency.
  - public void setSensorId(String sensorId);  
Metode ini merupakan *setter* dari atribut sensorId.
  - public String getSensorId();  
Metode ini merupakan *getter* dari atribut sensorId.
  - public LineChart<String,Number> getChart();  
Metode ini digunakan untuk membuat komponen-komponen yang akan digunakan untuk melakukan visualisasi nilai frekuensi.
  - public void render();  
Metode ini digunakan untuk memasukkan nilai-nilai frekuensi hasil perhitungan yang akan ditampilkan dalam bentuk grafik.

## Kelas SampleData



Gambar 4.19: Perancangan Kelas SampleData

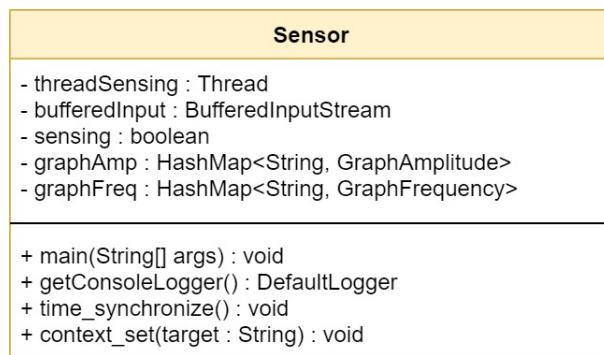
Kelas ini digunakan untuk membuat sampel data. Kelas ini memiliki atribut-atribut sebagai berikut:

- public double[] X;  
Atribut ini digunakan untuk menampung nilai x.
- public double[] Y;  
Atribut ini digunakan untuk menampung nilai y.
- public double[] Z;  
Atribut ini digunakan untuk menampung nilai z.

Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:

- public SampleData(int banyak);  
Metode ini digunakan sebagai konstruktor dari kelas SampleData.
- public void setX(double titikX);  
Metode ini digunakan sebagai setter dari atribut X.
- public void setY(double titikY);  
Metode ini digunakan sebagai setter dari atribut Y.
- public void setZ(double titikZ);  
Metode ini digunakan sebagai setter dari atribut Z.

## Kelas Sensor



Gambar 4.20: Perancangan Kelas Sensor

Kelas ini digunakan sebagai main class yang menerima input dari pengguna dan menampilkan output ke pengguna. Atribut-atribut yang terdapat pada kelas ini adalah sebagai berikut:

- private static Thread threadSensing;  
Atribut ini digunakan untuk membuat thread baru yang bertujuan untuk menerima hasil sense dari sensor node saat dalam kondisi sensing.
- private static BufferedInputStream bufferedInput;  
Atribut ini digunakan untuk membaca input dari BaseStation.
- private static boolean sensing;  
Atribut ini digunakan sebagai penanda kondisi apakah SensorNode sedang melakukan sensing atau tidak.

Metode-metode yang terdapat pada kelas ini adalah sebagai berikut:

- public static void main(String[] args);  
Metode ini digunakan sebagai metode main dari kelas ini.
- private static DefaultLogger getConsoleLogger();  
Metode ini digunakan untuk memunculkan tulisan pada console.
- private void time\_synchronize();  
Metode ini digunakan untuk mengatur dan memperbarui waktu pada BaseStation dengan waktu pada komputer pengguna.
- private void context\_set(String target);  
Metode ini digunakan untuk memilih context dari BaseStation.

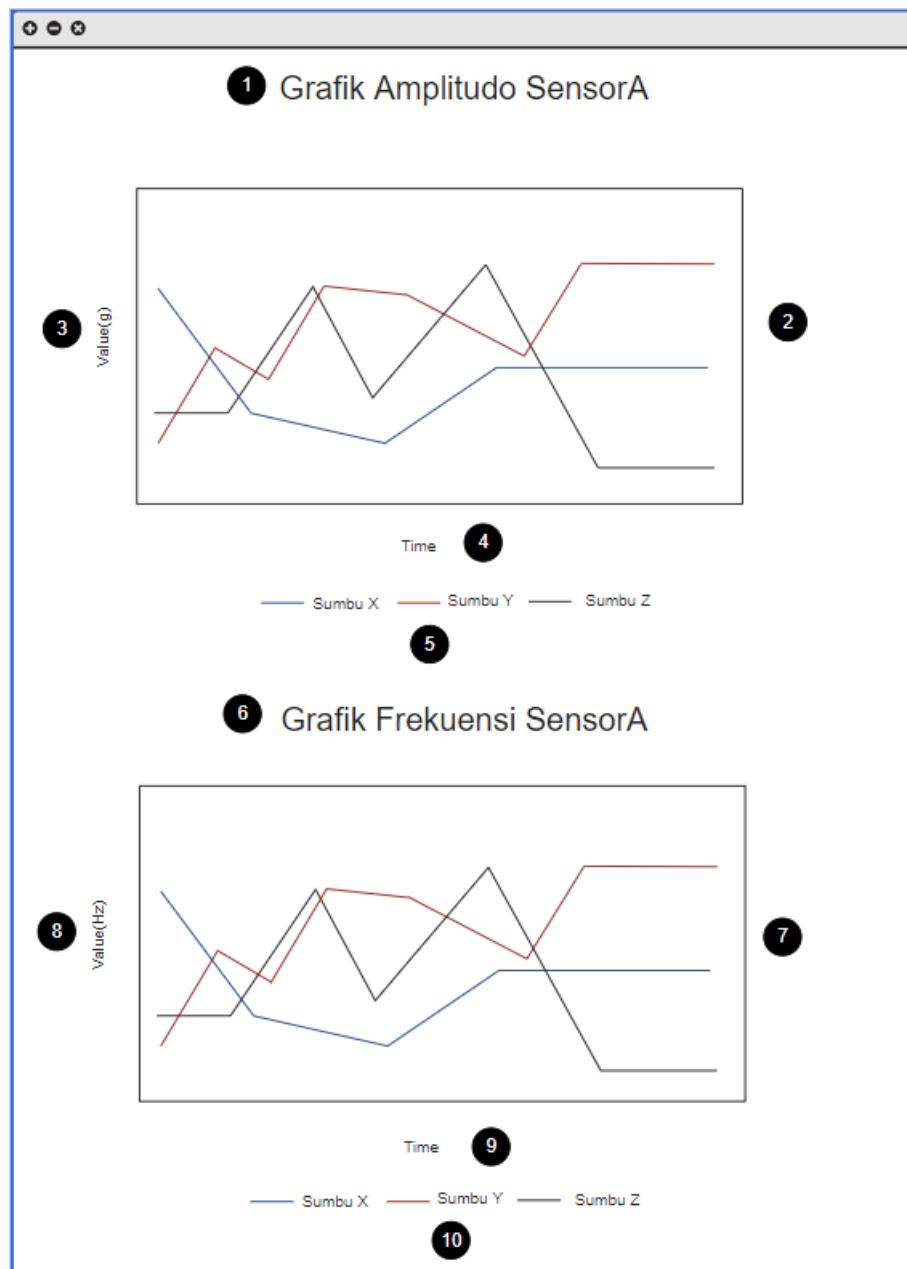
### 4.3 Perancangan Input dan Output

Aplikasi ini menggunakan interface *command line* untuk menerima input dari pengguna dan menampilkan keluaran dari aplikasi. Setiap input untuk aplikasi ini menggunakan tipe xdata *integer*. Aplikasi ini akan menampilkan 4 pilihan fungsi [4.1](#) dan pengguna dapat memasukan nilai dari 1 hingga 4 sebagai masukan untuk menjalankan fungsi tersebut. Berikut penjelasan perihal setiap masukan yang dapat dimasukkan oleh pengguna:

- Masukan dengan nilai 1 merupakan fungsi "Check Online Node Status" yang digunakan untuk mengetahui status dari setiap node sensor dan melakukan penyamaan waktu yang ada di node sensor
- Masukan dengan nilai 2 merupakan fungsi "Start Sensing" yang digunakan untuk memberikan perintah *sense* ke SensorNode dan menerima hasil sensing.
- Masukan dengan nilai 3 merupakan fungsi "Stop Sensing" yang digunakan untuk memberikan perintah berhenti sensing pada node sensor
- Masukan dengan nilai 4 merupakan fungsi "Exit Program" yang digunakan untuk memberhentikan program *Controller* dan juga program pada *SensorNode*

Jika pengguna memasukan fungsi selain "Stop Sensing" ketika *SensorNode* melakukan sensing, maka *Controller* akan menampilkan pesan "STILL SENSING.. USE OPTION "3" TO STOP IT!".

## 4.4 Perancangan Antar Muka Untuk Visualisasi



Gambar 4.21: Rancangan antar muka tampilan grafik amplitudo dan frekuensi sensor

Keterangan Gambar:

- Nomor 1 : Judul dari tampilan amplitudo sensor
- Nomor 2 : Grafik yang menampilkan hasil Amplitudo dari setiap sumbu
- Nomor 3 : Label 'Value(g)' yang dinyatakan untuk besarnya amplitudo
- Nomor 4 : Label 'Time' yang digunakan untuk menyatakan waktu
- Nomor 5 : Keterangan plot pada grafik yang ditampilkan
- Nomor 6 : Judul dari tampilan frekuensi sensor
- Nomor 7 : Grafik yang menampilkan hasil Frekuensi dari setiap sumbu
- Nomor 8 : Label 'Value(Hz)' yang dinyatakan untuk besarnya Frekuensi
- Nomor 9 : Label 'Time' yang digunakan untuk menyatakan waktu
- Nomor 10 : Keterangan plot pada grafik yang ditampilkan

## 4.5 Perancangan Pseudocode Aplikasi

### 4.5.1 Start Chart

Kelas *Start Chart* ini berfungsi untuk membuat window hasil grafik yang akan ditampilkan ke *user*. Metode yang digunakan untuk menampilkan grafik ini adalah *getScene()* dan metode yang digunakan untuk menyimpan data hasil sensing dan nilai FFT adalah *addData()*. Objek kelas dari StartChart akan digunakan ke abstract kelas yang bertujuan untuk menampilkan nilai amplitudo dan frekuensi. Berikut adalah *pseudocode* dari metode *getScene()* dan *addData()*:

- Pseudocode *getScene()*

---

#### Algorithm 1 getScene

---

```
1: Input : -
2: Output : Scene
3: function GETSCENE
4:     charts ← new ChartAmplitude(getSample(), getData(), sensorId)
5:     getSample.setRender(charts.get(0))
6:     charts ← new ChartFrequency(getSample(), getData(), sensorId)
7:     getSample.setRender(charts.get(1))
8:     ap ← AnchorPane
9:     i ← 0
10:    height ← 400
11:    width ← 600
12:    for charts in chart do
13:        lc ← LineChart
14:        lc ← charts.getChart()
15:        lc.relocate(0, height*i)
16:        i++
17:        ap.getChildren.add(LineChart)
18:        lc.setMinSize(width, height)
19:        lc.setMaxXize(width, height)
20:    end for
21:    return new Scene(ap, width, i*height);
22: end function
```

---

- Pseudocode *addData()*

---

**Algorithm 2** addData()

---

```

1: Input : String[]
2: Output : -
3: function ADDDATA
4:   if hasil[0].charAt(0) ← 2 then
5:     getSample().addX(Double.parseDouble(hasil[1]))
6:     getSample().addY(Double.parseDouble(hasil[2]))
7:     getSample().addZ(Double.parseDouble(hasil[3]))
8:     computeFFT ← FFT
9:     computeFFT.convertComplex()
10:    tempRes1 ← computeFFT.fft(computeFFT.xComplex)
11:    tempRes2 ← computeFFT.fft(computeFFT.yComplex)
12:    tempRes3 ← computeFFT.fft(computeFFT.zComplex)
13:    df ← DecimalFormat("*.****")
14:    for i to tempRes1.length do
15:      t1 ← t1 + tempRes1[i].absolute()
16:      t2 ← t2 + tempRes2[i].absolute()
17:      t3 ← t3 + tempRes3[i].absolute()
18:    end for
19:    t1 ← t1 / tempRes1.length
20:    t2 ← t2 / tempRes2.length
21:    t3 ← t3 / tempRes3.length
22:    getData().setX(Double.parseDouble(df.format(t1)))
23:    getData().setY(Double.parseDouble(df.format(t2)))
24:    getData().setZ(Double.parseDouble(df.format(t3)))
25:  end if
26: end function

```

---

#### 4.5.2 ChartAmplitude & ChartFrequency

Kelas ini berfungsi untuk menampilkan LineChart yang akan ada pada window dari kelas StartChart. Kedua kelas ini cara mengimplementasikannya hampir sama satu sama lain, metode yang digunakan untuk menampilkan LineChart pada kelas tersebut adalah *getChart()*. Terdapat juga metode *render()* yang digunakan untuk melakukan *update* nilai pada sumbu X dan Y pada *Line Chart*. Berikut adalah pseudocode dari metode *getChart()* dan *render()* dari kelas ChartAmplitude dan ChartFrequency.

- Pseudocode getChart() (ChartAmplitude)

---

**Algorithm 3** getChart()

---

```

1: Input : -
2: Output : LineChart
3: function GETCHART()
4:   final x ← CategoryAxis()
5:   final y ← NumberAxis()
6:   x.setLabel("Time")
7:   x.setAnimated(false)
8:   y.setLabel("Value(g)")
9:   y.setAnimated(false)
10:  lineChart ← LineChart<>(x,y)
11:  linechart.setAnimated(false);
12:  series1.setName("Sumbu X")
13:  series2.setName("Sumbu Y")
14:  series3.setName("Sumbu Z")
15:  lineChart.getData().add(series1)
16:  lineChart.getData().add(series2)
17:  lineChart.getData().add(series3)
18:  render()
19:  return lineChart
20: end function

```

---

- Pseudocode getChart() (ChartFrequency)

---

**Algorithm 4** getChart()

---

```

1: Input : -
2: Output : LineChart
3: function GETCHART()
4:   final x ← CategoryAxis()
5:   final y ← NumberAxis()
6:   x.setLabel("Time")
7:   x.setAnimated(false)
8:   y.setLabel("Value(Hz)")
9:   y.setAnimated(false)
10:  lineChart ← LineChart<>(x,y)
11:  linechart.setAnimated(false);
12:  series1.setName("Sumbu X")
13:  series2.setName("Sumbu Y")
14:  series3.setName("Sumbu Z")
15:  lineChart.getData().add(series1)
16:  lineChart.getData().add(series2)
17:  lineChart.getData().add(series3)
18:  render()
19:  return lineChart
20: end function

```

---

- Pseudocode render() (ChartAmplitude)

---

**Algorithm 5** render()

---

```

1: Input : -
2: Output : -
3: function RENDER()
4:   Platform.runlater(new Runnable())
5:   function RUN()
6:     now ← Date()
7:     lineChart.setTitle("Grafik Amplitudo "+sensorId)
8:     series1.getData().add(new XYChart.Data<>(SimpleDateFormat.format(now), getX()))
9:     series2.getData().add(new XYChart.Data<>(SimpleDateFormat.format(now), getY()))
10:    series3.getData().add(new XYChart.Data<>(SimpleDateFormat.format(now), getZ()))
11:    if series1.getData().size() > WINDOW_SIZE then
12:      series1.getData().remove(0)
13:      series2.getData().remove(0)
14:      series3.getData().remove(0)
15:    end if
16:  end function
17: end function

```

---

- Pseudocode render() (ChartFrequency)

---

**Algorithm 6** render()

---

```

1: Input : -
2: Output : -
3: function RENDER()
4:   Platform.runlater(new Runnable())
5:   function RUN()
6:     now ← Date()
7:     lineChart.setTitle("Grafik Frekuensi "+sensorId)
8:     series1.getData().add(new XYChart.Data<>(SimpleDateFormat.format(now), da-
taFrequency.X()))
9:     series2.getData().add(new XYChart.Data<>(SimpleDateFormat.format(now), da-
taFrequency.Y()))
10:    series3.getData().add(new XYChart.Data<>(SimpleDateFormat.format(now), da-
taFrequency.Z()))
11:    if series1.getData().size() > WINDOW_SIZE then
12:      series1.getData().remove(0)
13:      series2.getData().remove(0)
14:      series3.getData().remove(0)
15:    end if
16:  end function
17: end function

```

---

#### 4.5.3 Controller

Kelas Controller memiliki fungsi untuk menyimpan hasil sensing yang dilakukan oleh sensor dan melakukan algoritma FFT. Metode yang ada pada kelas ini adalah FFT() dan *bitReverse()*. Berikut adalah pseudocode dari metode FFT() dan *bitReverse()*

- Pseudocode *bitReverse()*

---

**Algorithm 7** bitReverse()

---

```

1: Input : n, bits
2: Output : int
3: function BITREVERSE(N, BITS)
4:   if n == 0 then
5:     return 0;
6:   end if
7:   temp ← n menjadi binaryString
8:   if panjang temp < bits then
9:     k ← bits - panjang temp
10:    while i ← do 0 to k, i ← i+1 temp ← "0" + temp
11:    end while
12:   end if
13:   res ← 0
14:   for i ← panjang temp-1 ; i>=0; i ← i-1 do
15:     if huruf ke i dari temp == '1' then res ← res + 2i
16:     end if
17:   end for
18:   return res
19: end function

```

---

- Pseudocode *FFT()*

---

**Algorithm 8** FFT()

---

```

1: Input : input
2: Output : Complex[]
3: function FFT(INPUT) bits ← Logpanjanginput/log2 orderFinal ← inisialisasi array
   Complex sebesar panjang input
4:   for i ← 0; i< panjang input;i++ do order ← bitReverse(i,bits) orderFinal[i] ←
   input[order]
5:   end for
6:   for i ← 2; j<=panjang orderFinal; i ← i x 2 do
7:     for j ← 0; j<panjang orderFinal; j← j+i do
8:       for k←0; k<i/2; k←k+1 do
9:         awal ← orderFinal[j+k]
10:        akhir ← orderFinal[j+k+(i/2)]
11:        weight ← (-2πk)/i
12:        exponential ← new Complex(cos(weight), sin(weight)).multiplication(akhir)
13:        orderFinal[j+k] ← awal tambah exponential
14:        orderFinal[j+k+(i/2)] ← awal kurang exponential
15:       end for
16:     end for
17:   end for
18:   return orderFinal
19: end function

```

---

## BAB 5

# IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan menjelaskan tentang implementasi program, pengujinya dan masalah yang dihadapi

### 5.1 Implementasi

#### 5.1.1 Lingkungan Implementasi

Berikut adalah spesifikasi *laptop* yang digunakan:

- *Processor*: Intel Core i7-6700HQ
- *Memory*: 16384 MB RAM @2300 MHz
- *Operating System*: Windows 10 Pro 64-bit

Berikut adalah spesifikasi perangkat lunak yang digunakan untuk implementasi program:

- IDE: Ecplipse verison 4.12.0
- Bahasa Pemograman: Java
- Java Library version : Java 1.8.0 191

Berikut adalah spesifikasi node sensor yang digunakan untuk implementasi program:

- Nama sensor: Preon32
- Processor: Cortex-M3
- Operating System: PreonVM
- Penyimpanan Sistem: 64 kByte SRAM
- Penyimpanan Data: 250 kByte Flash
- Pita Frekuensi: 2400.0 - 2483.5 MHz
- Jangkauan: 250 meter (luar ruangan) dan 30 meter (dalam ruangan)
- Sensor-sensor: Sensor getaran, sensor cahaya, sensor suhu, sensor kelembaban, dan sensor tekanan udara

#### 5.1.2 Hasil Implementasi Kelas

Berdasarkan pembahasan pada bab 4, terdapat beberapa kelas yang diimplementasi pada *sensor node* yaitu kelas *Accelerometer*, *SensorManager*, *BaseStation*. Kelas-kelas yang diimplementasi pada *sensor node* ini akan diatur oleh kelas *SensorManager* dan *BaseStaion* jika sensor node berperan sebagai *BaseStation*. Selain dari kelas yang telah disebut, terdapat kelas yang telah diimplementasi di komputer sebagai penghubung antara *base station* dengan komputer dan juga digunakan untuk melakukan proses perhitungan pada hasil sense.

##### Kelas Accelerometer

Kelas ini digunakan untuk mengatur dan menginisialisasi objek ADXL345 dan GPIO agar dapat mengatur sensor *accelerometer* terdapat pada sensor node. Terdapat metode *init* yang digunakan sebagai konstruktor dari kelas *Accelerometer*. Metode *sensing* yang digunakan untuk melakukan

pengukuran dan mengubah pengukuran ke dalam satuan gravitasi. Kode program kelas ini dapat dilihat pada lampiran A bagian A.1.

### Kelas SensorManager

Kelas ini digunakan sebagai main class pada sensor node. Saat sensor node dijalankan, atribut-atribut akan diinisialisasi dan metode *starts* dijalankan. Metode *starts* akan menginisialisasi *Transciever*, *RadioDriver* yang berfungsi untuk komunikasi antar sensor node dan *FrameIO* berfungsi sebagai pengiriman atau penerimaan pesan.

Kemudian metode *recieve* dijalankan, sensor node akan menerima pesan-pesan yang memiliki arti tertentu. Saat node sensor menerima pesan '@1', maka metode *recieve* akan menyamakan waktu dari pesan yang telah dikirim serta dengan mengirimkan status 'ONLINE' ke *BaseStation*. Jika sensor node menerima pesan '@2', maka sensor node akan membuat *thread* baru yang akan digunakan untuk memulai proses *sensing*. Hasil sensing kemudian akan dikirimkan dengan metode *send* menuju *BaseStation*. Saat sensor node menerima pesan '@3', sensor node akan berhenti melakukan sensing dan mengirimkan pesan ke *BaseStation* apabila *sensing* telah berhenti. Saat sensor node menerima pesan '@4', sensor node akan menghentikan program. Kode program dapat dilihat pada lampiran A bagian A.2

### Kelas BaseStation

Kelas ini digunakan sebagai main class pada sensor node yang berperan menjadi *BaseStation* dimana merupakan penghubung secara langsung ke komputer. Kelas ini akan menginisialisasi *USART* yang bertujuan untuk menerima pesan dan juga mengirim pesan dari pengguna komputer. Setelah *USART* terinisialisasi, kelas ini akan memulai *thread* baru dan metode *starts* dijalankan. Metode *starts* akan menginisialisasi *Transciever*, *RadioDriver* yang berfungsi untuk komunikasi antar sensor node dan *FrameIO* dan memulai *thread* baru untuk menerima input dari komputer. Setelah *BaseStation* menerima pesan dari komputer, *BaseStation* akan mengirimkan pesan menuju ke alamat node sensor yang terhubung yang disimpan pada atribut *connectedNode* dengan *USART*. Metode *recieve* telah diimplementasi dan dapat dilihat pada Listing 5.1.

Kode 5.1: Metode *recieve()*

```

1 public static void recieve(final FrameIO fio){
2     while(true) {
3         Frame frame = new Frame();
4         try {
5             fio.receive(frame);
6             byte[] content = frame.getPayload();
7             String str = new String(content, 0, content.length);
8             if(str.trim().charAt(0)=='1') {
9                 usart.write(str.getBytes());
10                usart.flush();
11            }
12            else if(str.trim().charAt(0)=='2') {
13                usart.write(str.getBytes());
14                usart.flush();
15            }
16            else if(str.trim().charAt(0)=='3') {
17                usart.write(str.getBytes());
18                usart.flush();
19            }
20            else if(str.trim().charAt(0)=='4') {
21                usart.write(str.getBytes());
22                usart.flush();
23            }
24        }
25        catch(USARTException e) {
26        }
27        catch(IOException e) {
28        }
29    }
30}
31}
32}

```

Kode program dari kelas ini dapat dilihat pada lampiran A bagian ??.

Selain dari kelas yang diimplementasikan di sensor node, terdapat juga kelas-kelas yang diimplementasikan pada komputer pengguna yaitu *Complex*, *FFT*, *GraphAmplitude*, *GraphFrequency*,

*SampleData, Tester, dan Visual.*

## Kelas Complex

Kelas ini digunakan untuk merepresentasikan bilangan kompleks. Pada kelas ini terdapat konstruktor, setter dan getter untuk setiap atribut dan beberapa metode yang digunakan untuk melakukan operasi bilangan kompleks. Metode *add* digunakan untuk melakukan operasi penjumlahan pada bilangan kompleks, metode *minus* digunakan untuk melakukan operasi pengurangan pada bilangan kompleks, metode *multiplication* digunakan untuk melakukan operasi perkalian pada bilangan kompleks dan metode *absolute* digunakan untuk memutlakkan bilangan kompleks. Metode *toString* digunakan untuk menjadikan bilangan kompleks ke dalam bentuk *String*. Kode program pada kelas ini dapat dilihat pada lampiran A bagian A.4.

## Kelas FFT

Pada kelas FFT terdapat metode bitReverse yang memiliki fungsi untuk melakukan *in bit reverse order* pada input dari algoritma FFT. Metode bitReverse yang telah diimplementasi dapat dilihat pada listing 5.2.

Kode 5.2: Metode bitReverse()

```

1 public int bitReverse(int banyak, int bit) {
2     if(banyak==0) {
3         return 0;
4     }
5
6     String temp = Integer.toBinaryString(banyak);
7     if(temp.length()<bit) {
8         int ct = bit - temp.length();
9         for(int i=0; i<ct; i++) {
10            temp = "0" + temp;
11        }
12    }
13
14    int hasil = 0;
15    for(int i = temp.length()-1; i>=0; i--) {
16        if(temp.charAt(i) == '1') {
17            hasil += Math.pow(2, i);
18        }
19    }
20    return hasil;
21 }
```

Terdapat juga metode FFT yang berfungsi untuk melakukan algoritma FFT. Metode fft yang telah diimplementasi dapat dilihat pada listing 5.3.

Kode 5.3: Metode fft()

```

1 public Complex[] fft(Complex[] input) {
2     int bit = (int) (Math.log(input.length) / Math.log(2));
3     Complex[] orderFinal = new Complex[input.length];
4     for(int i=0; i<input.length; i++) {
5         int order = bitReverse(i,bit);
6         orderFinal[i] = input[order];
7     }
8
9     for(int i=2; i<= orderFinal.length; i=i*2 ) {
10        for(int j=0; j<orderFinal.length; j +=i) {
11            for(int k=0; k<i/2; k++) {
12                Complex awal = orderFinal[j+k];
13                Complex akhir = orderFinal[j+k+(i/2)];
14
15                double weight = (-2 * Math.PI * k)/ (double) i;
16                Complex exponential = (new Complex(Math.cos(weight), Math.sin(weight))).multiplication(akhir);
17
18                orderFinal[j+k] = awal.add(exponential);
19                orderFinal[j+k+(i/2)] = awal.minus(exponential);
20            }
21        }
22    }
23    return orderFinal;
24 }
```

Kode program secara keseluruhan pada kelas FFT dapat dilihat pada lampiran A bagian A.6.

## Kelas StartChart

Kelas ini digunakan untuk membuat window hasil grafik yang ditampilkan ke *user*. Kelas ini terdapat beberapa metode yaitu *getScene()* dan *addData()*. Hasil implementasi metode dapat dilihat pada listing 5.4 dan 5.5.

Kode 5.4: Metode getScene()

```

1  public Scene getScene() {
2      charts.add(new ChartAmplitude(this.getSample(), this.getData(), sensorId));
3      this.getSample().setRender(charts.get(0));
4      charts.add(new ChartFrequency(this.getSample(), this.getData(), sensorId));
5      this.getData().setRender(charts.get(1));
6      AnchorPane ap = new AnchorPane();
7      int i = 0;
8      int height = 400;
9      int width = 600;
10     for (Chart chart : charts) {
11         LineChart lc = chart.getChart();
12         lc.relocate(0, height * i);
13         i++;
14         ap.getChildren().add(lc);
15         lc.setMinSize(width, height);
16         lc.setMaxSize(width, height);
17     }
18     return new Scene(ap, width, i * height);
19 }
```

Kode 5.5: Metode addData()

```

1  public void addData(String[] hasil){
2      if (hasil[0].charAt(0) == '2') {
3          // Sample Data
4          this.getSample().addX(Double.parseDouble(hasil[1]));
5          this.getSample().addY(Double.parseDouble(hasil[2]));
6          this.getSample().addZ(Double.parseDouble(hasil[3]));
7
8          // Perhitungan FFT
9          FFT computeFFT = new FFT(this.getSample());
10         computeFFT.convertComplex();
11         Complex[] tempRes1 = computeFFT.fft(computeFFT.xComplex);
12         Complex[] tempRes2 = computeFFT.fft(computeFFT.yComplex);
13         Complex[] tempRes3 = computeFFT.fft(computeFFT.zComplex);
14         double t1 = 0, t2 = 0, t3 = 0;
15         DecimalFormat df = new DecimalFormat("#.####");
16         for (int i = 0; i < tempRes1.length; i++) {
17             t1 += tempRes1[i].absolute();
18             t2 += tempRes2[i].absolute();
19             t3 += tempRes3[i].absolute();
20         }
21         t1/= tempRes1.length;
22         t2/= tempRes2.length;
23         t3/= tempRes3.length;
24         System.out.printf("%f %f %f\n", t1, t2, t3);
25         this.getData().setX(Double.parseDouble(df.format(t1)));
26         this.getData().setY(Double.parseDouble(df.format(t2)));
27         this.getData().setZ(Double.parseDouble(df.format(t3)));
28
29     }
30 }
```

Kode program secara keseluruhan pada kelas StartChart dapat dilihat pada lampiran A bagian A.10.

## Kelas ChartAmplitude dan ChartFrequency

Kelas ini berfungsi untuk menampilkan LineChart yang akan ada pada window dari kelas StartChart. Kelas ini akan menampilkan nilai amplitudo pada LineChart(ChartAmplitude) dan menampilkan nilai frekuensi pada Line Chart(ChartFrequency). Kelas ini memiliki beberapa metode yaitu *getChart()* dan *render()*. Hasil implementasi *getChart()* dan *render()* pada kelas ChartAmplitude dapat dilihat pada listing 5.6 dan 5.7. Hasil implementasi *getChart()* dan *render()* pada kelas ChartFrequency dapat dilihat pada listing 5.8 dan 5.9.

Kode 5.6: Metode getChart() (ChartAmplitude)

```

1  public LineChart<String, Number> getChart() throws IOException {
2      final CategoryAxis x = new CategoryAxis(); // buat sumbu x (waktu)
3      final NumberAxis y = new NumberAxis(); // buat sumbu y (hasil sensor)
4
5      x.setLabel("Time");
6      x.setAnimated(false);
7      y.setLabel("Value_(g)");
8      y.setAnimated(false);
```

```

9
10 // bikin line chart
11 lineChart = new LineChart<x, y>;
12 lineChart.setAnimated(false);
13
14 series1.setName("Sumbu_X");
15 series2.setName("Sumbu_Y");
16 series3.setName("Sumbu_Z");
17
18 // add series to chart
19 lineChart.getData().add(series1);
20 lineChart.getData().add(series2);
21 lineChart.getData().add(series3);
22
23 // setup scene
24
25 // setup executor to put data periodically
26 scheduledExecutorService = Executors.newSingleThreadScheduledExecutor();
27
28 render();
29 return this.lineChart;
30 }

```

Kode 5.7: Metode render() (ChartAmplitude)

```

1 public void render() throws IOException{
2
3     Platform.runLater(new Runnable() {
4
5         @Override
6         public void run() {
7
8             Date now = new Date();
9             lineChart.setTitle("Grafik_Amplitudo_" +sensorId);
10
11            try {
12                String baris = sensorId+"_"+getX()+"_"+getY()+"_"+getZ();
13                writer.write(baris);
14                writer.newLine();
15                writer.flush();
16            } catch(Exception e) {
17                e.getMessage();
18            }
19            // taro random number dengan waktu skrng
20            series1.getData().add(new XYChart.Data<x, y>(simpleDateFormat.format(now), getX()));
21            series2.getData().add(new XYChart.Data<x, y>(simpleDateFormat.format(now), getY()));
22            series3.getData().add(new XYChart.Data<x, y>(simpleDateFormat.format(now), getZ()));
23            if (series1.getData().size() > WINDOW_SIZE) {
24                series1.getData().remove(0);
25                series2.getData().remove(0);
26                series3.getData().remove(0);
27            }
28        }
29    });
30 }

```

Kode 5.8: Metode getChart() (ChartFrequency)

```

1 public LineChart<String, Number> getChart() {
2     final CategoryAxis x = new CategoryAxis(); // buat sumbu x (waktu)
3     final NumberAxis y = new NumberAxis(); // buat sumbu y (hasil sensor)
4
5     x.setLabel("Time");
6     x.setAnimated(false);
7     y.setLabel("Value_(Hz)");
8     y.setAnimated(false);
9
10    // bikin line chart
11    lineChart = new LineChart<x, y>;
12    lineChart.setAnimated(false);
13
14    // buat nampilin datanya
15    series1.setName("Sumbu_X");
16
17    series2.setName("Sumbu_Y");
18
19    series3.setName("Sumbu_Z");
20
21    // add series to chart
22    lineChart.getData().add(series1);
23    lineChart.getData().add(series2);
24    lineChart.getData().add(series3);
25
26    render();
27    return this.lineChart;
28 }

```

Kode 5.9: Metode render() (ChartFrequency)

```

1 public void render() {
2     Platform.runLater(new Runnable() {

```

```

3| 
4|     @Override
5|     public void run() {
6|         lineChart.setTitle("Grafik_FrekuenSi_" +sensorId);
7|         Date now = new Date();
8| 
9|         try {
10|             String baris = sensorId+"_"+dataFrequency.X+"_"+dataFrequency.Y+"_"+dataFrequency.Z;
11|             writer.write(baris);
12|             writer.newLine();
13|             writer.flush();
14|         }
15|         catch(Exception e) {
16|             e.getMessage();
17|         }
18|         series1.getData()
19|             .add(new XYChart.Data<>(simpleDateFormat.format(now), dataFrequency.X));
20|         series2.getData()
21|             .add(new XYChart.Data<>(simpleDateFormat.format(now), dataFrequency.Y));
22|         series3.getData()
23|             .add(new XYChart.Data<>(simpleDateFormat.format(now), dataFrequency.Z));
24|         tempX = dataFrequency.X;
25|         tempY = dataFrequency.Y;
26|         tempZ = dataFrequency.Z;
27|         if (series1.getData().size() > WINDOW_SIZE) {
28|             series1.getData().remove(0);
29|             series2.getData().remove(0);
30|             series3.getData().remove(0);
31|         }
32|     }
33| });
34|

```

Kode program secara keseluruhan pada kelas ChartAmplitude dan ChartFrequency dapat dilihat pada lampiran A bagian [A.10](#).

## Kelas RenderChart

Kelas ini merupakan *interface class* yang akan digunakan pada kelas ChartAmplitude dan ChartFrekuensi. Kode program secara keseluruhan dapat dilihat pada [A](#) bagian [A.7](#).

## SampleData

Kelas ini berfungsi untuk membuat sampel data berdasarkan hasil sensing. Pada kelas ini terdapat beberapa atribut yang digunakan untuk menyimpan hasil sensing. Kode program ini dapat dilihat pada lampiran A bagian [A.8](#).

## Sensor

Kelas ini merupakan main class dari komputer pengguna. Kelas ini digunakan sebagai penghubung komunikasi dan memberikan perintah kepada BaseStation. Sebelum kelas ini dijalankan, BaseStation harus terlebih dahulu terhubung dengan komputer pengguna melalui USB PORT.

Saat kelas ini dijalankan, komputer pengguna dan BaseStation akan diinisialisasi dengan metode *context\_set* dan BaseStation melakukan sinkronisasi waktu dengan komputer dengan metode *time\_synchronize*. Setelah itu dibuat objek Preon32Helper untuk menjalankan module pada BaseStation dan mengatur koneksi komunikasi dengan BaseStation.

Kemudian kelas ini akan menampilkan 4 pilihan yang dapat dipilih oleh pengguna yaitu "Check Online Node Status", "Start Sensing", "Stop Sensing", dan "Exit Program". Jika pengguna memasukkan angka '1', maka aplikasi akan menampilkan node-node sensor yang berstatus "ONLINE" beserta waktunya. Jika pengguna memasukkan angka '2', maka aplikasi akan menampilkan status "Sensing..." dan menampilkan grafik dari setiap node sensor. Jika pengguna memasukkan angka '3', maka aplikasi yang sedang dalam status sensing akan berhenti. Jika pengguna memasukkan angka '4', maka aplikasi akan berhenti. Kode program dapat dilihat pada lampiran A bagian [A.9](#).

### 5.1.3 Hasil Implementasi Antar Muka Visualisasi Hasil *Sense*

Berikut contoh hasil dari implementasi antar muka visualisasi hasil *sense* menggunakan *library* dari *JavaFX*:

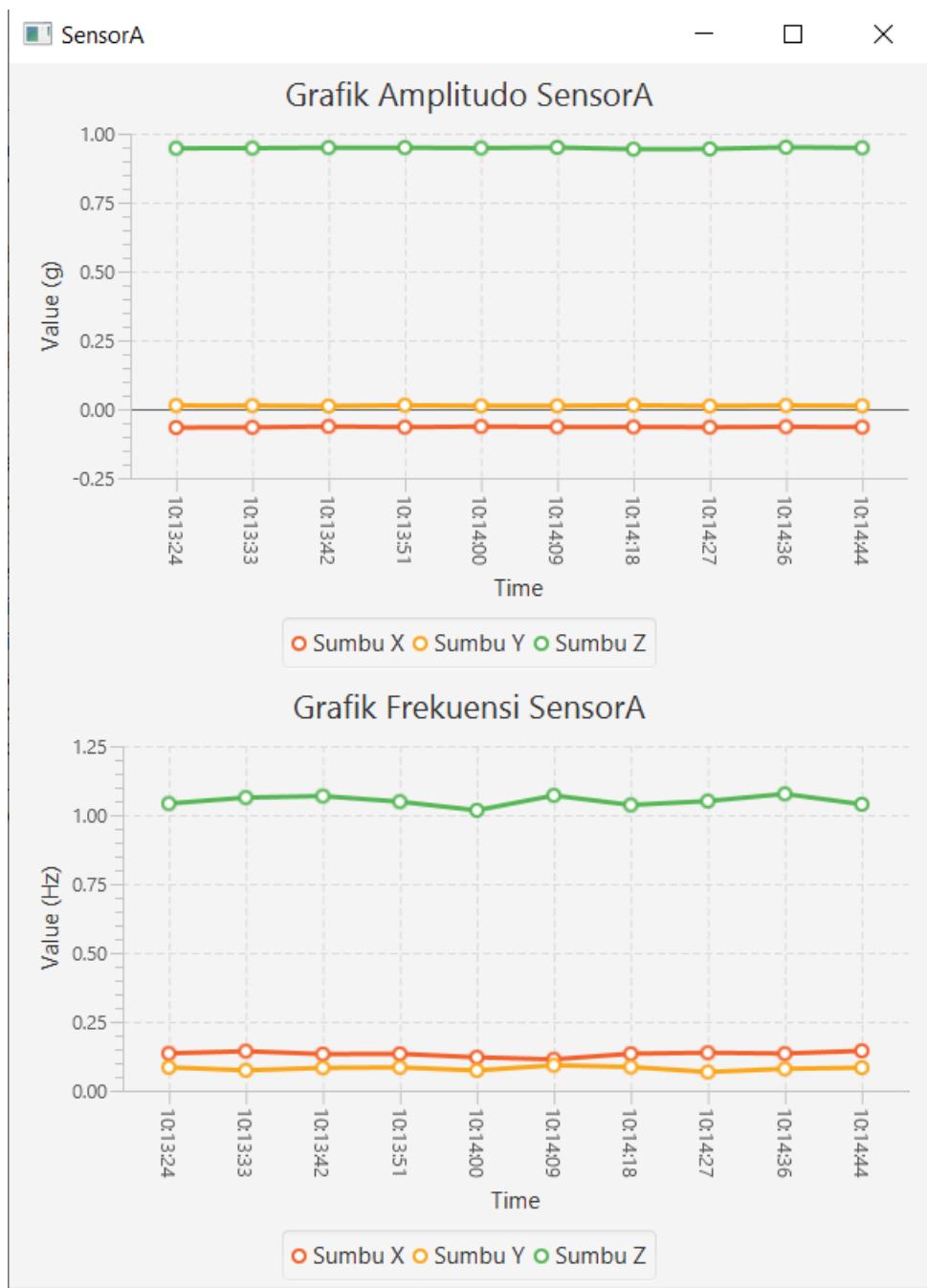
```
BUILD SUCCESSFUL
Total time: 0 seconds
[echo] Using context "config/context1.properties"

init:

cmd.time.synchronize:
[Commander.Cmd] Synchronizing time...

BUILD SUCCESSFUL
Total time: 2 seconds
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
```

Gambar 5.1: Tampilan awal aplikasi



Gambar 5.2: Grafik hasil implementasi visualisasi hasil *sense*

Pada tampilan awal (Gambar 5.1), pengguna perlu memilih *option Sense*. Tampilan hasil visualisasi hasil *sense* akan muncul setelah sensor node mengirimkan hasil *sense*. Grafik visualisasi akan menunjukkan amplitudo dan frekuensi dari sensor yang dapat dilihat pada (Gambar 5.2).

## 5.2 Pengujian

Pengujian dilakukan dengan menggunakan dua metode, yaitu pengujian fungsional dan pengujian eksperimental

### 5.2.1 Pengujian Fungsional

Pengujian dilakukan dengan menjalankan aplikasi dengan *Command Line Interface*. Tampilan utama ketika aplikasi dijalankan dapat dilihat pada Gambar 5.3

```
BUILD SUCCESSFUL
Total time: 0 seconds
[echo] Using context "config/context1.properties"

init:

cmd.time.synchronize:
[Commander.Cmd] Synchronizing time...

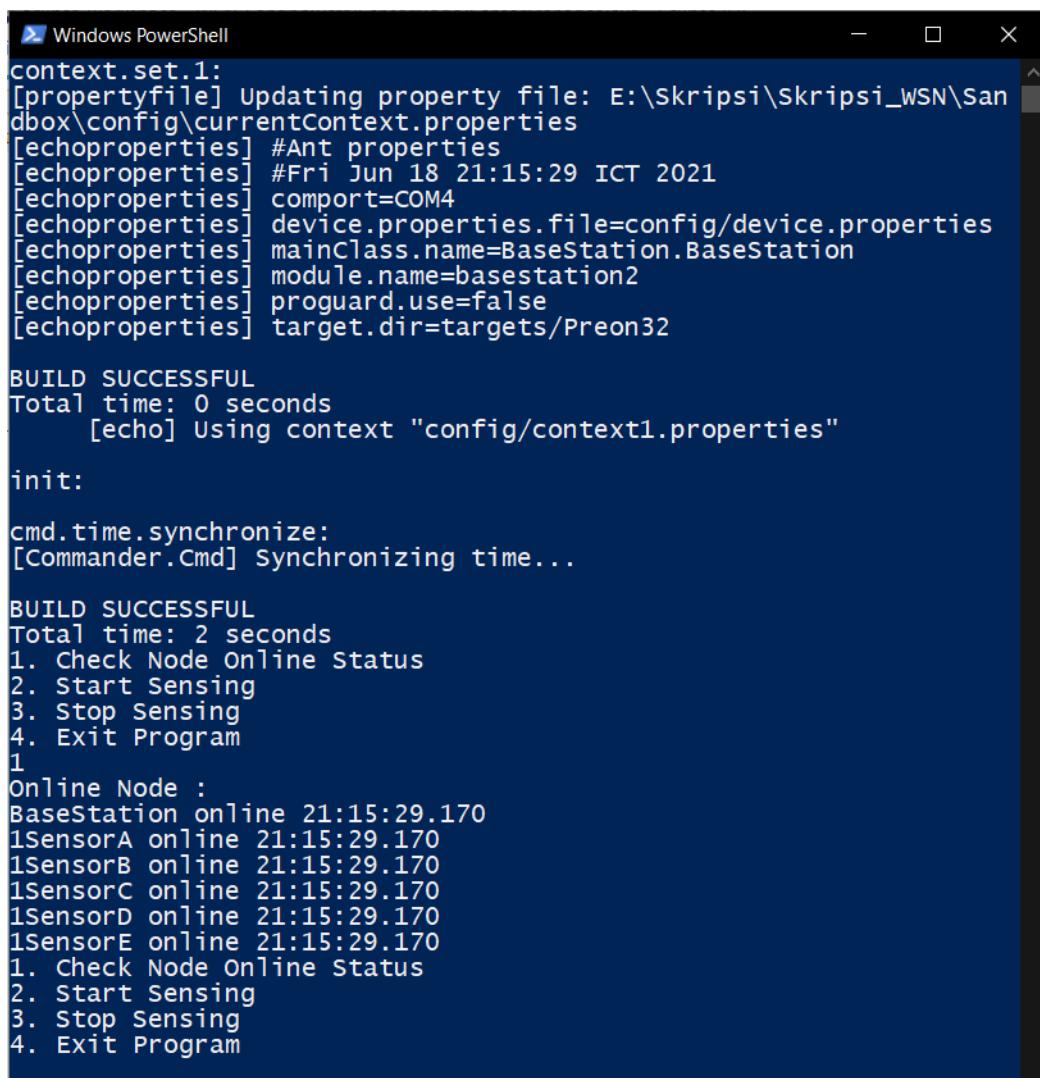
BUILD SUCCESSFUL
Total time: 2 seconds
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
```

Gambar 5.3: Tampilan awal aplikasi

Fitur-fitur yang terdapat pada aplikasi adalah sebagai berikut:

- Check Online Node Status

Fitur ini berfungsi untuk menampilkan sensor node - sensor node yang sedang menyala dan menyamakan waktu pada sensor node dengan waktu yang ada pada komputer pengguna. Untuk menjalankan fitur ini, pengguna memasukkan angka '1' dan menunggu hasil yang ditampilkan seperti pada Gambar 5.4.



```
Windows PowerShell
context.set.1:
[propertyfile] Updating property file: E:\Skripsi\Skripsi_WSN\San
dbox\config\currentContext.properties
[echoproperties] #Ant properties
[echoproperties] #Fri Jun 18 21:15:29 ICT 2021
[echoproperties] comport=COM4
[echoproperties] device.properties.file=config/device.properties
[echoproperties] mainClass.name=BaseStation.BaseStation
[echoproperties] module.name=basestation2
[echoproperties] proguard.use=false
[echoproperties] target.dir=targets/Preon32

BUILD SUCCESSFUL
Total time: 0 seconds
[echo] Using context "config/context1.properties"

init:

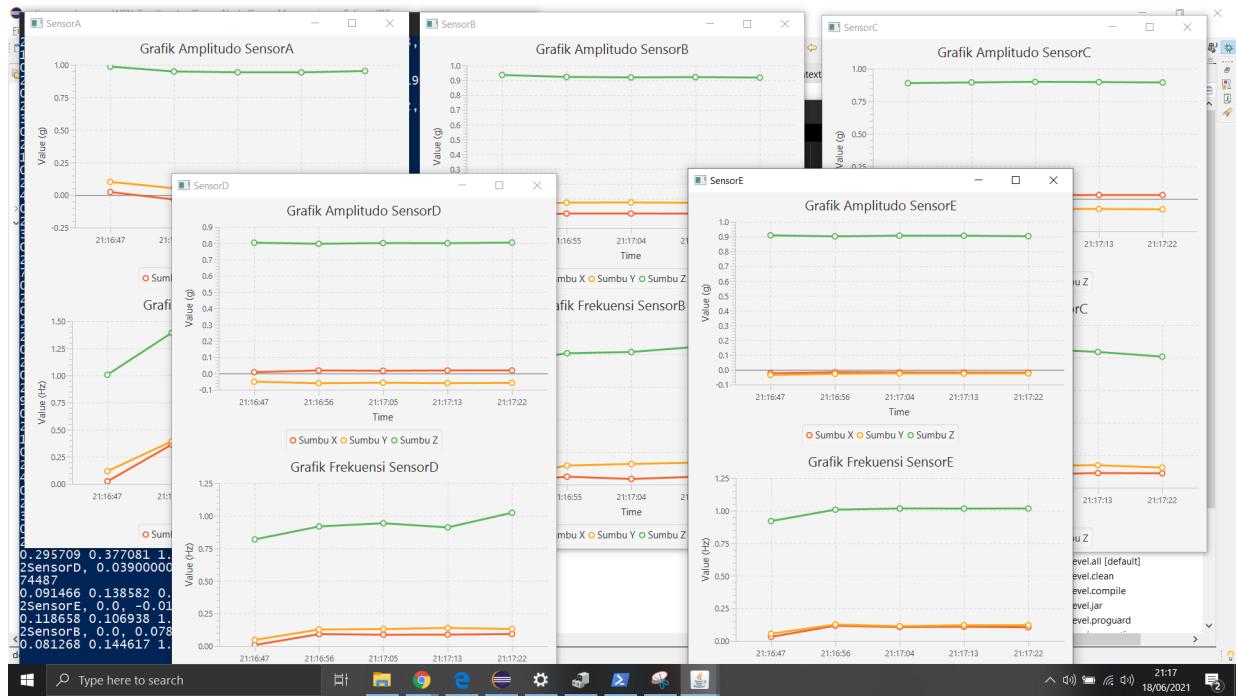
cmd.time.synchronize:
[Commander.Cmd] Synchronizing time...

BUILD SUCCESSFUL
Total time: 2 seconds
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
1
Online Node :
BaseStation online 21:15:29.170
1SensorA online 21:15:29.170
1SensorB online 21:15:29.170
1SensorC online 21:15:29.170
1SensorD online 21:15:29.170
1SensorE online 21:15:29.170
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
```

Gambar 5.4: Pengujian "Check Node Online Status"

- Start Sensing

Fitur ini berfungsi untuk memberikan perintah kepada sensor node - sensor node yang sedang menyala untuk melakukan *sensing*. Saat pengguna memasukkan angka '2', aplikasi akan menampilkan grafik hasil *sensing* dan grafik frekuensi yang akan terus diperbaharui. Berikut tampilan yang tampil setelah pengguna memasukkan angka '2' seperti pada Gambar 5.5.



Gambar 5.5: Pengujian "Start sensing"

Saat fungsi *start sensing* berjalan, aplikasi tidak akan menerima input selain masukan angka '3' untuk memberhentikan *sensing*. Tampilan jika pengguna memasukkan input selain angka '3' dapat dilihat pada Gambar 5.6.

```
BUILD SUCCESSFUL
Total time: 2 seconds
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
1
Online Node :
BaseStation online 15:24:36.410
1SensorA online 15:24:36.410
1SensorB online 15:24:36.410
1SensorC online 15:24:36.410
1SensorD online 15:24:36.410
1SensorE online 15:24:36.410
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
2
Sensing...
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
1
Still in Sensing State!!
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
2
Still in Sensing State!!
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
4
Still in Sensing State!!
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
```

Gambar 5.6: Pengujian input selain angka '3' saat "Start sensing"

- Stop sensing

Fitur ini berfungsi untuk menghentikan sensor node yang sedang melakukan *sensing*. Pengguna harus memasukkan angka '3' saat sensor node sedang *sensing* untuk menjalankan fungsi ini. Tampilan setelah fungsi ini dijalankan dapat dilihat seperti pada Gambar 5.7.

```
Stopping sense..
Sensing has been stopped!!
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
```

Gambar 5.7: Pengujian "Stop sensing"

- Exit Program

Fitur ini berfungsi untuk keluar dari aplikasi. Pengguna harus memasukkan angka '4' saat sensor node tidak dalam keadaan *sensing* untuk menjalankan fungsi ini. Tampilan hasil dari fungsi ini dapat dilihat pada Gambar 5.8.

```

Stopping sense..
Sensing has been stopped!!
1. Check Node Online Status
2. Start Sensing
3. Stop Sensing
4. Exit Program
4
Terminating program..
Program has terminated!!
PS E:\Skripsi\Skripsi_WSN>

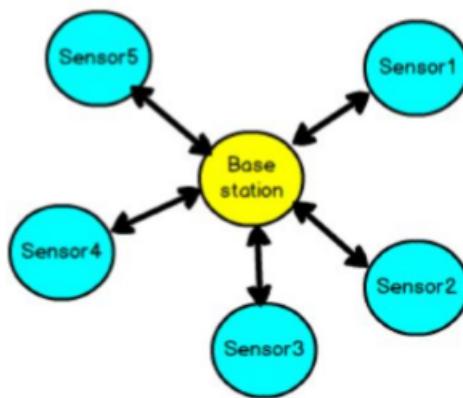
```

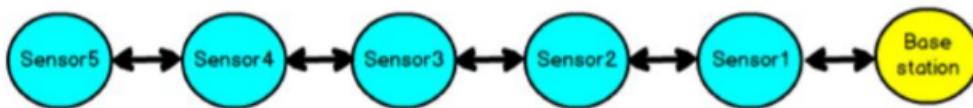
Gambar 5.8: Pengujian "Exit Program"

### 5.2.2 Pengujian Eksperimental

Pengujian Eksperimental ini dilakukan pada dua tempat, yaitu yang pertama di Mall Paskal 23 Square yang berada di jalan pasir kaliki Bandung. Lokasi spesifik pengujian dilakukan pada lantai 2 outdoor pada mall tersebut. Sedangkan tempat pengujian kedua berada di Universitas Katolik Parahyangan gedung 10 bagian *rooftop*. Sensor node yang digunakan untuk melakukan pengujian berjumlah 6 dengan 1 sensor node sebagai *base station*. Arsitektur WSN yang digunakan adalah *flat* dengan *single-hop* dan *multi-hop*. Pengujian dilakukan untuk melakukan *monitoring* sebuah gedung apakah terjadi sebuah getaran atau tidak. Pengujian ini mengukur frekuensi dari setiap sumbu *x*, *y*, dan *z* dari *accelerometer*.

Topologi yang digunakan untuk melakukan pengujian ini adalah topologi *star* (Gambar 5.9) dan topologi *tree* (Gambar 5.10). Kedua topologi ini dipilih karena topologi tersebut sudah mencakup 2 jalur komunikasi WSN (*single hop* dan *multi hop*). Topologi star merupakan komunikasi jenis *single hop* karena setiap sensor node terhubung secara langsung dengan *base station*. Topologi tree merupakan komunikasi jenis *multi hop* karena sensor node tidak terhubung secara langsung dengan *base station*, melainkan sensor node akan terhubung dengan sensor node lainnya yang jaraknya paling dekat.

Gambar 5.9: Topologi star yang digunakan (*Single hop*)



Gambar 5.10: Topologi tree yang digunakan (*Multi hop*)

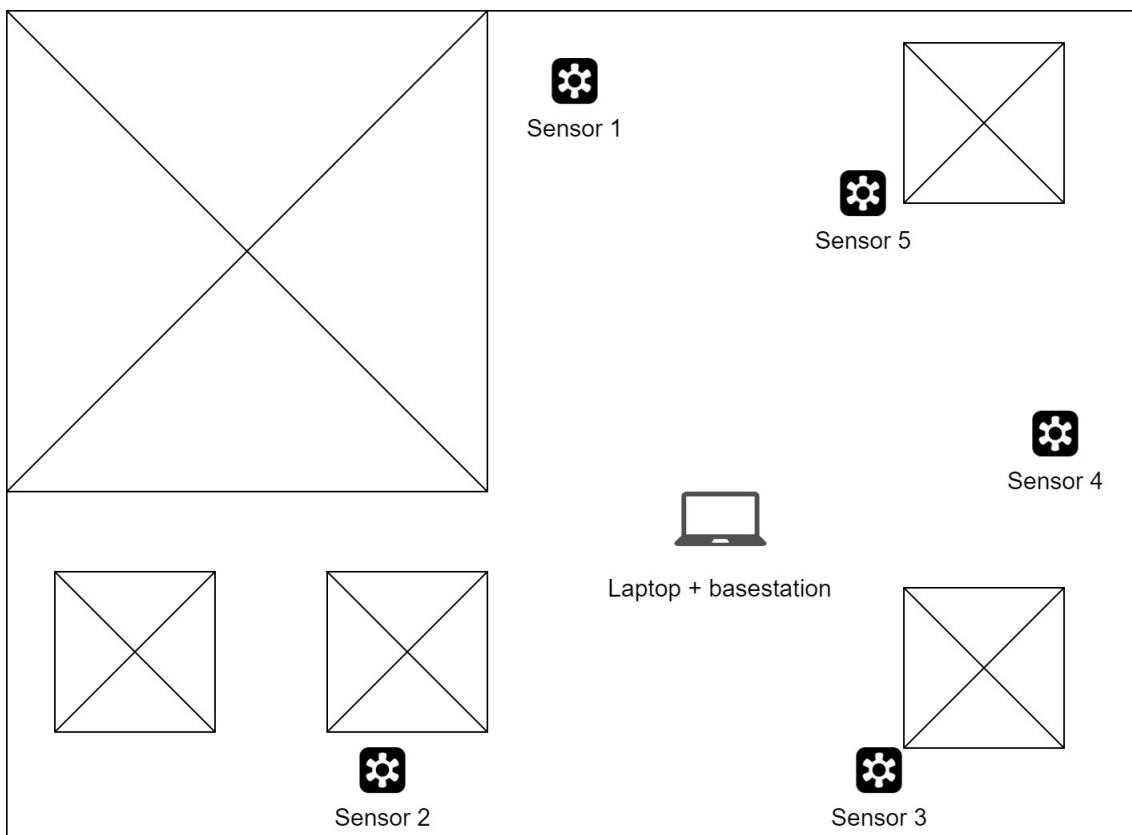
Skenario-skenario dalam pengujian ini bertujuan untuk melihat kemampuan sensor node untuk melakukan *sensing* dan monitoring getaran yang ditampilkan ke pengguna. Langkah pengujian pada setiap skenario dalam pengujian ini adalah sebagai berikut:

1. Peletakan sensor node diatur dengan topologi yang akan diujikan.
2. Sensor node diletakkan di tempat yang jaraknya cukup jauh pada setiap skenario.
3. Mengecek status online setiap sensor node melalui komputer.
4. Memberi perintah *sense* ke setiap sensor node melalui komputer.
5. Hasil tampilan monitoring getaran akan ditampilkan ke komputer pengguna.

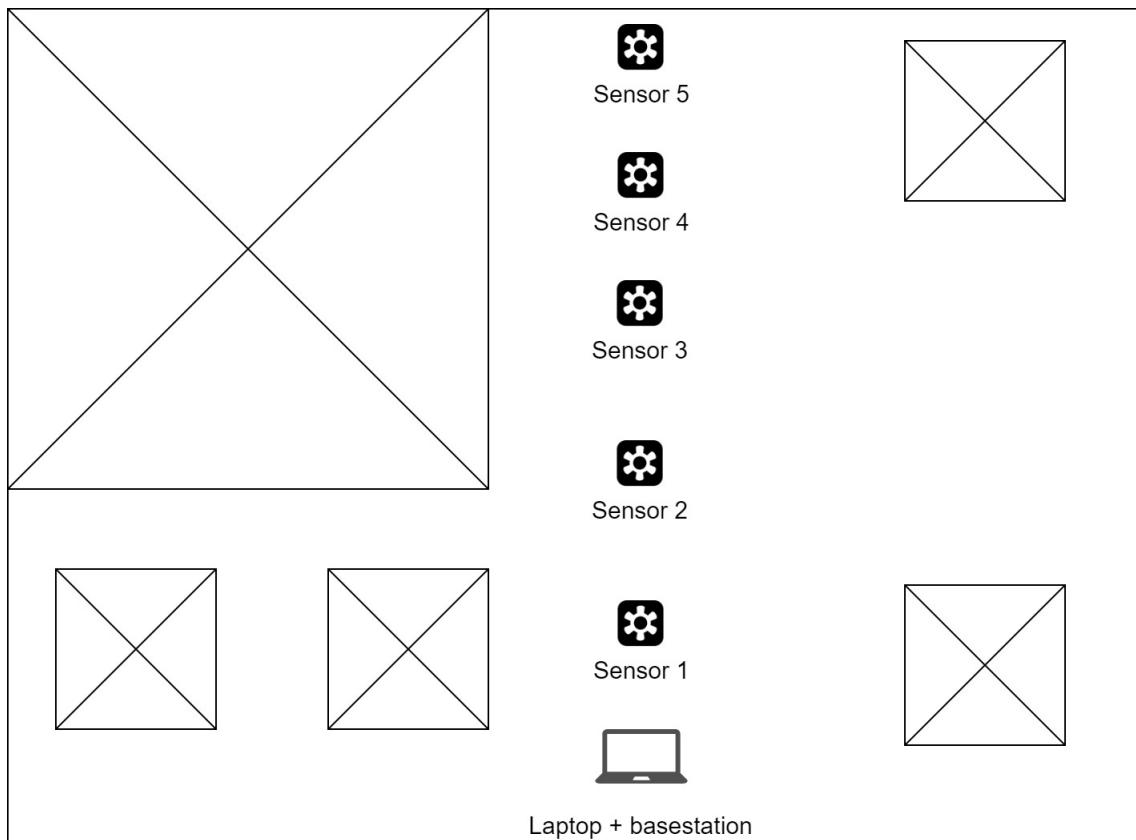
Hasil dari pengujian ini berupa grafik amplitudo akselerasi dari hasil *sensing* dan grafik frekuensi dari setiap node.

### Pengujian di Outdoor lantai 2 Mall Paskal 23 Square

Pada pengujian ini, sensor akan diletakkan sesuai topologi yang digunakan yaitu topologi star seperti pada Gambar 5.11 dan topologi tree seperti pada Gambar 5.12. Setiap sensor node diberikan jarak antar satu sama lain untuk menunjukkan komunikasi setiap sensor node secara nirkabel dan untuk melihat hasil monitoring di lokasi yang berbeda-beda.



Gambar 5.11: Denah peletakan sensor node dengan topologi star



Gambar 5.12: Denah peletakan sensor node dengan topologi tree

Pengujian dilakukan saat siang hari dengan cuaca berawan. Hasil grafik dari pengujian dapat dilihat pada lampiran B

- Topologi star

Tabel 5.1: Tabel hasil getaran yang tertangkap di Mall 23 Paskal Square dengan topologi star

Id Sensor	Waktu	Frekuensi pada sumbu x (Hz)	Frekuensi pada sumbu y (Hz)	Frekuensi pada sumbu z (Hz)
SensorA	15:07:19	1	1	2.3
SensorB	15:07:28	0.15	0.15	1
SensorC	15:07:26	0.10	0.20	1
SensorD	15:07:08	0.35	0.30	1.25
SensorE	15:07:10	0.15	0.05	1.1
SensorA	15:07:28	1.1	1.1	2.4
SensorB	15:07:37	0.15	0.15	1.01
SensorC	15:07:35	0.10	0.15	1.1
SensorD	15:07:17	0.35	0.30	1.25
SensorE	15:07:20	0.15	0.05	1
SensorA	15:07:37	1	1.1	2.3
SensorB	15:07:46	0.15	0.16	1
SensorC	15:07:44	0.10	0.15	1.1
SensorD	15:07:29	0.29	0.35	1.26
SensorE	15:07:30	0.15	0.09	1.03

- Topologi tree

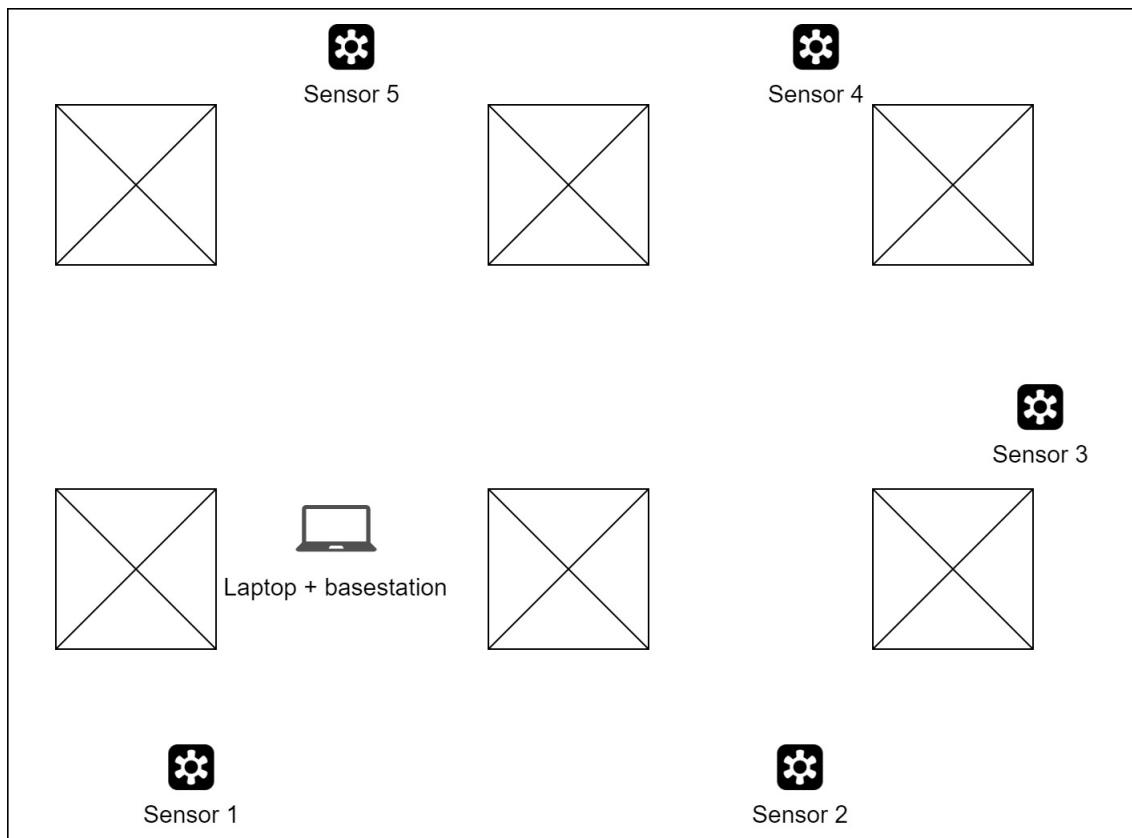
Tabel 5.2: Tabel hasil getaran yang tertangkap di Mall 23 Paskal Square dengan topologi tree

Id Sensor	Waktu	Frekuensi pada sumbu x (Hz)	Frekuensi pada sumbu y (Hz)	Frekuensi pada sumbu z (Hz)
SensorA	16:45:06	0.05	0.2	1.15
SensorB	16:48:30	0.07	0.15	1.01
SensorC	16:48:30	0.20	0.30	1.1
SensorD	16:48:27	0.10	0.15	0.97
SensorE	16:48:15	0.30	0.35	1.30
SensorA	16:45:15	1	1	2.15
SensorB	16:48:40	0.05	0.07	1
SensorC	16:48:39	0.17	0.30	1.07
SensorD	16:48:36	0.10	0.10	0.95
SensorE	16:48:27	0.27	0.29	1.35
SensorA	16:45:25	0.9	1.05	2.25
SensorB	16:48:49	0.07	0.17	1.01
SensorC	16:48:49	0.18	0.32	1.15
SensorD	16:48:46	0.10	0.15	0.99
SensorE	16:48:38	0.26	0.30	1.32

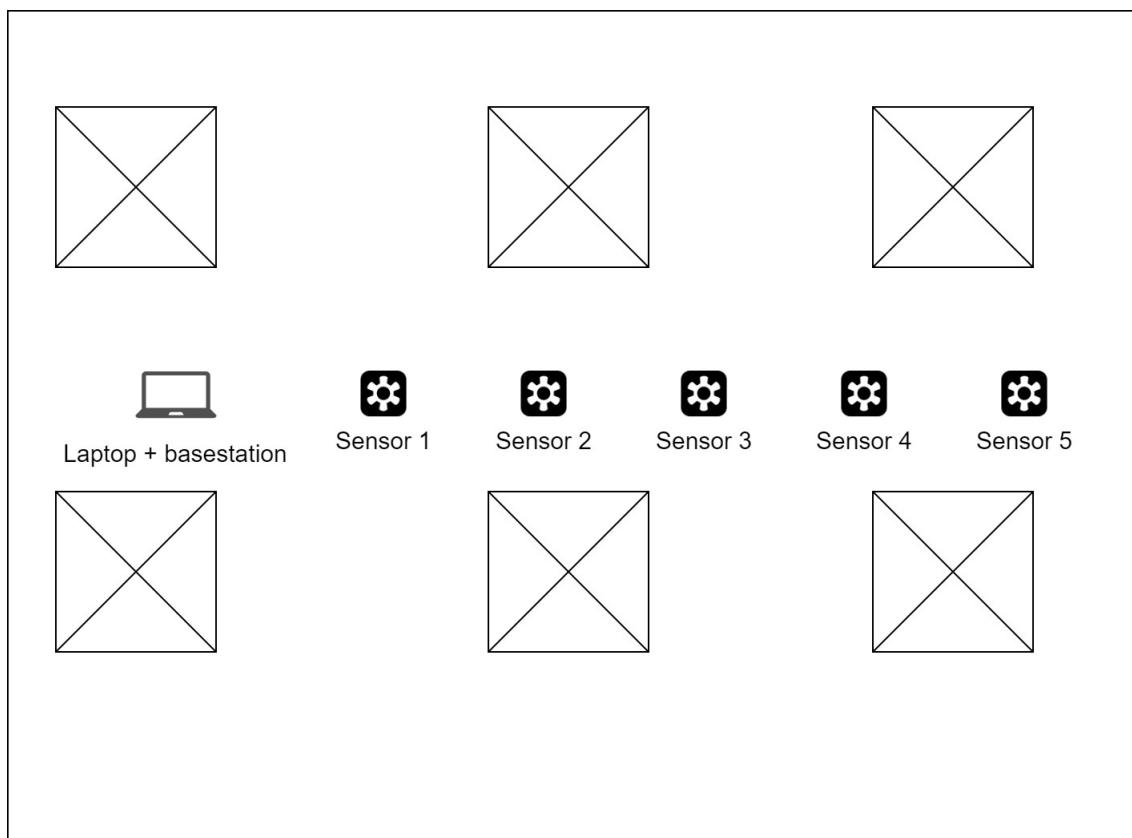
Dari tabel 5.1 dan 5.4 terlihat bahwa hasil frekuensi sensor yang berbeda-beda disebabkan karena perbedaan tempat dan waktu saat melakukan *sensing* sehingga frekuensi yang ditangkap juga nilainya berbeda-beda. Kesimpulan yang didapatkan berdasarkan hasil pengujian adalah frekuensi tertinggi pada sumbu x adalah 1.1 Hz di SensorA pada pengujian topologi star dengan waktu 15:07:28. Frekuensi tertinggi pada sumbu y adalah 1.1 Hz di SensorA pada pengujian topologi star dengan waktu 15:07:28 dan 15:07:37. Frekuensi tertinggi pada sumbu z adalah 2.4 Hz di SensorA pada pengujian topologi star dengan waktu 15:07:28.

### Pengujian di Gedung 10 bagian rooftop Universitas Katolik Parahyangan

Pada pengujian ini, sensor akan diletakkan sesuai topologi yang digunakan yaitu topologi star seperti pada Gambar 5.13 dan topologi tree seperti pada Gambar 5.14. Pengujian dilakukan pada bagian *rooftop* gedung 10 Universitas Katolik Parahyangan. Setiap sensor node diberikan jarak antar satu sama lain untuk menunjukkan komunikasi setiap sensor node secara nirkabel dan untuk melihat hasil monitoring di lokasi yang berbeda-beda.



Gambar 5.13: Denah peletakan sensor node dengan topologi tree



Gambar 5.14: Denah peletakan sensor node dengan topologi tree

Pengujian dilakukan saat siang hari dengan cuaca berawan. Hasil grafik dari pengujian dapat dilihat pada lampiran C.

- Topologi star

Tabel 5.3: Tabel hasil getaran yang tertangkap di Rooftop gedung 10 dengan topologi star

Id Sensor	Waktu	Frekuensi pada sumbu x (Hz)	Frekuensi pada sumbu y (Hz)	Frekuensi pada sumbu z (Hz)
SensorA	10:37:15	0.32	0.40	1.40
SensorB	10:36:33	0.12	0.12	1.07
SensorC	10:35:22	0.10	0.09	1
SensorD	10:35:40	0.10	0.07	1.05
SensorE	10:36:29	0.18	0.10	0.99
SensorA	10:37:19	0.30	0.40	1.35
SensorB	10:36:42	0.10	0.10	1.05
SensorC	10:35:40	0.09	0.09	1.05
SensorD	10:35:54	0.12	0.05	0.95
SensorE	10:36:38	0.18	0.10	1
SensorA	10:37:23	0.29	0.35	1.35
SensorB	10:36:50	0.12	0.10	1.05
SensorC	10:35:46	0.1	0.1	1
SensorD	10:36:08	0.13	0.06	1.02
SensorE	10:36:47	0.18	0.10	1.1

- Topologi tree

Tabel 5.4: Tabel hasil getaran yang tertangkap di Rooftop gedung 10 dengan topologi tree

Id Sensor	Waktu	Frekuensi pada sumbu x (Hz)	Frekuensi pada sumbu y (Hz)	Frekuensi pada sumbu z (Hz)
SensorA	15:27:17	0.05	0.07	1.08
SensorB	15:27:30	0.10	0.19	1.05
SensorC	15:27:47	0.13	0.12	1.13
SensorD	15:28:17	0.09	0.13	0.90
SensorE	15:28:27	0.12	0.1	1.01
SensorA	15:27:26	0.05	0.07	1.05
SensorB	15:27:41	0.11	0.17	1.3
SensorC	15:27:58	0.13	0.13	1.1
SensorD	15:28:28	0.10	0.13	0.92
SensorE	15:28:39	0.12	0.12	1
SensorA	15:27:35	0.06	0.09	1.08
SensorB	15:27:52	0.11	0.17	1
SensorC	15:28:09	0.13	0.13	1.15
SensorD	15:28:39	0.1	0.13	0.90
SensorE	15:28:50	0.1	0.1	1.02

Kesimpulan yang didapatkan dari hasil pengujian di Rooftop Universitas Katolik Parahyangan adalah frekuensi tertinggi pada sumbu x adalah 0.32 Hz di SensorA pada pengujian topologi star dengan waktu 10:37:15. Frekuensi tertinggi pada sumbu y adalah 0.40 Hz di SensorA pada pengujian topologi star dengan waktu 10:37:15 dan 10:37:19. Frekuensi tertinggi pada sumbu z adalah 1.40 di SensorA pada pengujian topologi star dengan waktu 10:37:15.

### 5.2.3 Kesimpulan Hasil Pengujian

Berdasarkan pengujian yang telah dilakukan pada dua tempat yaitu Mall Paskal 23 Square dan Gedung Rooftop Universitas Katolik Parahyangan, didapatkan kesimpulan nilai frekuensi yang ditangkap berbeda diantaranya karena keramaian dari kedua juga berbeda. Frekuensi terendah yang tercatat selama proses pengujian adalah 0.05 Hz pada pengujian di Mall Paskal 23 Square topologi star oleh SensorA dengan waktu 16:48:40. Frekuensi tertinggi yang tercatat selama proses pengujian adalah 2.4 Hz pada pengujian di Mall Paskal 23 Square topologi star oleh SensorA dengan waktu 15:07:28

### 5.2.4 Masalah dalam pengujian

Berikut masalah-masalah yang dihadapi saat melakukan pengujian:

1. Sensor node yang digunakan untuk melakukan pengujian terbatas, sehingga sensor node harus dipakai secara bergantian dengan mahasiswa lain yang mengambil topik skripsi yang berkaitan dengan sensor node.
2. Saat pengujian, sensor node dapat mati secara tiba-tiba, karena sumber power sensor node yaitu baterai dapat habis saat sensor node melakukan *sensing*
3. Cuaca hujan yang menghambat pengujian karena pengujian dilakukan di ruangan semi terbuka.
4. Pengujian dilakukan saat adanya pandemi Covid19, sehingga adanya keterbatasan untuk penggunaan sensor node secara bersama dengan mahasiswa lain yang memilih topik skripsi yang berkaitan demi mematuhi aturan protokol kesehatan.



## **BAB 6**

### **KESIMPULAN DAN SARAN**

#### **6.1 Kesimpulan**

Berdasarkan hasil penelitian yang dilakukan, diperoleh kesimpulan-kesimpulan sebagai berikut:

1. Cara kerja sensor node, sensor accelerometer dan WSN telah berhasil dipelajari
2. Aplikasi pemantauan getaran gedung menggunakan WSN telah berhasil dibangun.
3. Topologi yang digunakan yaitu *star* dan *tree* untuk melakukan pengujian tidak berpengaruh terhadap hasil pemantauan.
4. Gedung yang dilakukan pengujian masih dalam kondisi sehat berdasarkan dengan keadaan gedung, jenis gedung dan juga hasil dari nilai amplitudo ataupun frekuensi.

#### **6.2 Saran**

Berdasarkan hasil penelitian yang dilakukan, ada beberapa saran untuk pengembangan aplikasi sebagai berikut:

1. Aplikasi ini hanya dibangun untuk melakukan pemantauan getaran, lebih baik apabila aplikasi dikembangkan agar dapat mengklasifikasi jenis getaran yang dipantau.
2. Aplikasi lebih baik diaplikasikan ke lingkungan big data, supaya akurasi hasil getaran yang ditangkap lebih akurat.
3. Jumlah sensor node yang digunakan untuk pengujian dapat lebih banyak sehingga lingkungan jangkauan pengujian dapat dilakukan dengan skala yang lebih besar.



## **DAFTAR REFERENSI**

- [1] Dharmawijaya, H. (2010) Pengertian dan klasifikasi bangunan gedung, . **1**, 1–28.
- [2] P, C. E. . F. (2004) *Vibration based condition monitoring: a review*.
- [3] Putra, V. G. V. (2017) *Pengantar Fisika Dasar*.
- [4] Dargie, W. dan Poellabauer, C. (2011) *Fundamentals of Wireless Sensor Networks: Theory and Practice*.
- [5] Brigham, E. O. (1988) *The Fast Fourier Transform and Its Applications*. A Division of Simon & Schuster, Englewood Cliffs, New Jersey 07632.



## LAMPIRAN A

### KODE PROGRAM

Kode A.1: Accelerometer.java

```
1 package SensorNode;
2 import com.virtenio.driver.device.ADXL345;
3 import com.virtenio.driver.gpio.GPIO;
4 import com.virtenio.driver.gpio.NativeGPIO;
5 import com.virtenio.driver.spi.NativeSPI;
6
7
8 public class Accelerometer {
9     private ADXL345 sensorAccel;
10    private GPIO accelCs;
11
12    public void init() throws Exception{
13        accelCs = NativeGPIO.getInstance(20);
14        NativeSPI spi = NativeSPI.getInstance(0);
15
16        spi.open(ADXL345.SPI_MODE, ADXL345.SPI_BIT_ORDER, ADXL345.SPI_MAX_SPEED);
17
18        sensorAccel = new ADXL345(spi, accelCs);
19
20        sensorAccel.open();
21        sensorAccel.setDateFormat(ADXL345.DATA_FORMAT_RANGE_2G);
22        sensorAccel.setDataRate(ADXL345.DATA_RATE_3200HZ);
23        sensorAccel.setPowerControl(ADXL345.POWER_CONTROL_MEASURE);
24    }
25
26
27    public float[] sensing() throws Exception{
28        short[] resTemp = new short[3];
29        float[] res = new float[3];
30        sensorAccel.getValuesRaw(resTemp, 0);
31        sensorAccel.convertRaw(resTemp, 0, res, 0);
32        for (int i =0; i<3; i++) {
33            res[i] *= sensorAccel.getConversionScale();
34        }
35        return res;
36    }
37 }
```

Kode A.2: SensorManager.java

```
1 package SensorNode;
2 import java.io.IOException;
3
4 import com.virtenio.driver.device.at86rf231.AT86RF231;
5 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
6 import com.virtenio.misc.PropertyHelper;
7 import com.virtenio.preon32.node.Node;
8 import com.virtenio.radio.ieee_802_15_4.Frame;
9 import com.virtenio.radio.ieee_802_15_4.FrameIO;
10 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
11 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
12 import com.virtenio.vm.Time;
13
14
15 // kelas buat menghandle antar sensor
16 public class SensorManager {
17
18     public static final Accelerometer accel = new Accelerometer();
19     public static boolean sensing = true;
20     public Thread threadSense;
21     public static float[] resTemp;
22
23     //Setting address sensor
24     public static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
25     public static int[] node_list = new int[] {
26         PropertyHelper.getInt("radio.panid", 0xABFE),
27         PropertyHelper.getInt("radio.panid", 0xDAAA),
28         PropertyHelper.getInt("radio.panid", 0xDAAB),
29         PropertyHelper.getInt("radio.panid", 0xDAAC),
30         PropertyHelper.getInt("radio.panid", 0xDAAD),
31         PropertyHelper.getInt("radio.panid", 0xDAAE)
32     };
33 }
```

```

34 //Setting sensor
35
36
37 private static final String sensorId = "SensorC";
38 private static int PREVIOUS_NODE_ADDRESS = node_list[0];
39 private static int SENSOR_NODE_ADDRESS = node_list[3];
40 private static int NEXT_NODE_ADDRESS = 0;
41 private static int BASE_STATION_ADDRESS = node_list[0];
42
43
44 public static void main(String[] args) throws Exception{
45     accel.init();
46     System.out.println(sensorId + "waiting...");
47     starts();
48 }
49
50 public static void starts() {
51     try {
52         AT86RF231 trans = Node.getInstance().getTransceiver();
53         trans.open();
54         trans.setAddressFilter(COMMON_PANID, SENSOR_NODE_ADDRESS, SENSOR_NODE_ADDRESS, false);
55         final RadioDriver radioDriver = new AT86RF231RadioDriver(trans);
56         final FrameIO fio = new RadioDriverFrameIO(radioDriver);
57         recieve(fio);
58     } catch(Exception e) {
59     }
60 }
61
62
63
64 public static void send(String message, int source, int destination, FrameIO fio) {
65     int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
66             | Frame.SRC_ADDR_16;
67     final Frame sentFrame = new Frame(frameControl);
68     sentFrame.setDestPanId(COMMON_PANID);
69     sentFrame.setDestAddr(destination);
70     sentFrame.setSrcAddr(source);
71     sentFrame.setPayload(message.getBytes());
72     try {
73         fio.transmit(sentFrame);
74     } catch (Exception e) {
75     }
76 }
77
78
79
80 public static void recieve(final FrameIO fio) {
81     Thread thread = new Thread() {
82         public void run() {
83             Frame frame = new Frame();
84             while(true) {
85                 try {
86                     fio.receive(frame);
87                     byte[] content = frame.getPayload();
88                     String str = new String(content, 0, content.length);
89                     System.out.println(str);
90                     if(str.charAt(0)!= '@'){
91                         send(str, SENSOR_NODE_ADDRESS, PREVIOUS_NODE_ADDRESS, fio);
92                     }
93                     if(str.substring(0, 2).equalsIgnoreCase("@1")) {
94                         long curTime = Time.currentTimeMillis();
95                         send("@1", SENSOR_NODE_ADDRESS, NEXT_NODE_ADDRESS, fio);
96                         send("1" + sensorId + "_online_" + curTime+"\\n", SENSOR_NODE_ADDRESS, PREVIOUS_NODE_ADDRESS, fio);
97                     }
98                     else if(str.substring(0, 2).equalsIgnoreCase("@2")) {
99                         System.out.println("Sensing_start");
100                        send("@2", SENSOR_NODE_ADDRESS, NEXT_NODE_ADDRESS, fio);
101                        sensing = true;
102                        Thread sensingThread = new Thread() {
103                            public void run() {
104                                try {
105                                    while(sensing) {
106                                        resTemp = accel.sensing();
107                                        String message = "2"+sensorId+", "+resTemp[0]+", "+resTemp[1]+", "+resTemp[2]+"\n";
108                                        send(message, SENSOR_NODE_ADDRESS, PREVIOUS_NODE_ADDRESS, fio);
109                                        Thread.sleep(100);
110                                    }
111                                } catch (InterruptedException e) {
112                                    e.printStackTrace();
113                                }
114                                catch (Exception e) {
115                                    e.printStackTrace();
116                                }
117                            }
118                        };
119                        sensingThread.start();
120                    };
121                };
122            };
123        }
124        else if(str.substring(0, 2).equalsIgnoreCase("@3")) {
125            if(sensing) {
126                String message = "3Sensing_has_stopped";
127                send("@3", SENSOR_NODE_ADDRESS, NEXT_NODE_ADDRESS, fio);
128                send(message, SENSOR_NODE_ADDRESS, PREVIOUS_NODE_ADDRESS, fio);
129                sensing = false;
130            }
131        }
132        else if(str.substring(0, 2).equalsIgnoreCase("@4")) {

```

```

133                     send("04", SENSOR_NODE_ADDRESS, NEXT_NODE_ADDRESS, fio);
134                     send("4", SENSOR_NODE_ADDRESS, PREVIOUS_NODE_ADDRESS, fio);
135                     System.out.println("Exit from the program");
136                     System.exit(0);
137                 }
138             }
139         } catch (IOException e) {
140             }
141         }
142     }
143 }
144 thread.start();
145
146 }
147 }
```

Kode A.3: BaseStation.java

```

1 package BaseStation;
2
3 import java.io.IOException;
4 import java.io.OutputStream;
5 import java.util.HashMap;
6
7 import com.virtenio.driver.device.at86rf231.AT86RF231;
8 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
9 import com.virtenio.driver.usart.NativeUSART;
10 import com.virtenio.driver.usart.USART;
11 import com.virtenio.driver.usart.USARTException;
12 import com.virtenio.driver.usart.USARTParams;
13 import com.virtenio.misc.PropertyHelper;
14 import com.virtenio.preon32.examples.common.USARTConstants;
15 import com.virtenio.preon32.node.Node;
16 import com.virtenio.radio.ieee_802_15_4.Frame;
17 import com.virtenio.radio.ieee_802_15_4.FrameIO;
18 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
19 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
20 import com.virtenio.vm.Time;
21
22 public class BaseStation {
23     private static USART usart;
24     private static OutputStream output;
25
26     //Setting address sensor
27     public static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
28     public static int[] node_list = new int[] {
29         PropertyHelper.getInt("radio.panid", 0xABFE),
30         PropertyHelper.getInt("radio.panid", 0xAAAA),
31         PropertyHelper.getInt("radio.panid", 0xDAAB),
32         PropertyHelper.getInt("radio.panid", 0xDAAC),
33         PropertyHelper.getInt("radio.panid", 0xDAAD),
34         PropertyHelper.getInt("radio.panid", 0xDAAE)
35     };
36
37     private static int curr = node_list[0];
38
39     private static int[] connectedNode = new int[] {
40         node_list[1]
41     };
42
43     public static HashMap<String, Integer> addressNodeMap;
44
45     private static boolean done;
46     private static String ackAddress;
47     private static String result;
48
49     public static void send(String message, int source, int destination, FrameIO fio) {
50         int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
51             | Frame.SRC_ADDR_16;
52         final Frame sentFrame = new Frame(frameControl);
53         sentFrame.setDestPanId(COMMON_PANID);
54         sentFrame.setDestAddr(destination);
55         sentFrame.setSrcAddr(source);
56         sentFrame.setPayload(message.getBytes());
57         try {
58             fio.transmit(sentFrame);
59             Thread.sleep(100);
60         } catch (Exception e) {
61             }
62         }
63         System.out.println(message);
64     }
65
66     public static void main(String[] args) throws USARTException {
67         addressNodeMap = new HashMap<String, Integer>();
68         for (int i=1; i<connectedNode.length; i++) {
69             addressNodeMap.put("Sensor"+i, connectedNode[i-1]);
70         }
71
72         usart = configUSART();
73         output = usart.getOutputStream();
74
75         ackAddress = "";
76         result = "";
77         done = false;
78
79         Thread thread = new Thread() {
```



```

180        else if(str.trim().charAt(0)=='4') {
181            usart.write(str.getBytes());
182            usart.flush();
183        }
184    } catch(USARTException e) {
185    }
186    catch(IOException e) {
187    }
188}
189}
190}
191}
192}
193}
194}
195
196 public static USART configUSART() {
197     USARTParams params = USARTConstants.PARAMS_115200;
198     NativeUSART usart = NativeUSART.getInstance(0);
199     try {
200         usart.close();
201         usart.open(params);
202         return usart;
203     }
204     catch(Exception e) {
205         return null;
206     }
207 }
208 }
```

Kode A.4: Complex.java

```

1
2
3 public class Complex {
4     public double real;
5     public double imaginer;
6
7     public Complex(double re, double img) {
8         this.real = re;
9         this.imaginer = img;
10    }
11
12    public void setReal(double r) {
13        this.real = r;
14    }
15
16    public void setImaginer(double i) {
17        this.imaginer = i;
18    }
19
20    public double getReal() {
21        return this.real;
22    }
23
24    public double getImaginer() {
25        return this.imaginer;
26    }
27
28    public Complex add(Complex c) {
29        Complex result = new Complex(this.real+c.real, this.imaginer+c.imaginer);
30        return result;
31    }
32
33    public Complex minus(Complex c) {
34        Complex result = new Complex(this.real-c.real, this.imaginer-c.imaginer);
35        return result;
36    }
37
38    public Complex multiplication(Complex c) {
39        Complex result = new Complex(this.real*c.real - this.imaginer*c.imaginer, this.real*c.imaginer + this.imaginer*c.real);
40        return result;
41    }
42
43    public double absolute() {
44        return Math.sqrt(Math.pow(this.real, this.real) + Math.pow(this.imaginer, this.imaginer));
45    }
46
47    public String toString() {
48        return this.real+", "+this.imaginer;
49    }
50 }
```

Kode A.5: DataFrequency.java

```

1 import java.io.IOException;
2
3 public class DataFrequency {
4
5     public String sensorId;
6     public Double X;
7     public Double Y;
8     public Double Z;
9     private int counter = 0;
10    private RenderChart rc;
```

```

11
12     public DataFrequency() {
13         this.X = 0.0;
14         this.Y = 0.0;
15         this.Z = 0.0;
16     }
17
18     public void setRender(RenderChart rc) {
19         this.rc = rc;
20     }
21
22     public Double getX() {
23         return this.X;
24     }
25
26     public Double getY() {
27         return this.Y;
28     }
29
30     public Double getZ() {
31         return this.Z;
32     }
33
34     public void setX(double X) {
35         this.X = X;
36     }
37
38     public void setY(double Y) {
39         this.Y = Y;
40     }
41     public void setZ(double Z) throws IOException {
42         this.Z = Z;
43         counter++;
44         if(counter == RenderChart.MAX_COUNTER) {
45             counter = 0;
46             rc.render();
47         }
48     }
49 }
```

Kode A.6: FFT.java

```

1  public class FFT {
2     public SampleData data;
3     public int panjang;
4     public Complex[] xComplex;
5     public Complex[] yComplex;
6     public Complex[] zComplex;
7
8     public FFT(int panjang, SampleData data) {
9         this.panjang = panjang;
10        this.data = data;
11        this.xComplex = new Complex[panjang];
12        this.yComplex = new Complex[panjang];
13        this.zComplex = new Complex[panjang];
14    }
15
16    public void convertComplex() {
17        for(int i=0; i<panjang; i++) {
18            xComplex[i] = new Complex(data.X[i],0);
19            yComplex[i] = new Complex(data.Y[i],0);
20            zComplex[i] = new Complex(data.Z[i],0);
21        }
22    }
23
24    public int bitReverse(int banyak, int bit) {
25        if(banyak==0) {
26            return 0;
27        }
28
29        String temp = Integer.toBinaryString(banyak);
30        if(temp.length()<bit) {
31            int ct = bit - temp.length();
32            for(int i=0; i<ct; i++) {
33                temp = "0" + temp;
34            }
35        }
36
37        int hasil = 0;
38        for(int i = temp.length()-1; i>=0; i--) {
39            if(temp.charAt(i) == '1') {
40                hasil += Math.pow(2, i);
41            }
42        }
43        return hasil;
44    }
45
46    public Complex[] fft(Complex[] input) {
47        int bit = (int) (Math.log(input.length) / Math.log(2));
48        Complex[] orderFinal = new Complex[input.length];
49        for(int i=0; i<input.length; i++) {
50            int order = bitReverse(i,bit);
51            orderFinal[i] = input[order];
52        }
53
54        for(int i=2; i<= orderFinal.length; i=i*2 ) {
55            for(int j=0; j<orderFinal.length; j +=i) {
56
```

```

57         for(int k=0; k<i/2; k++) {
58             Complex awal = orderFinal[j+k];
59             Complex akhir = orderFinal[j+k+(i/2)];
60
61             double weight = (-2 * Math.PI * k)/ (double) i;
62             Complex exponential = (new Complex(Math.cos(weight), Math.sin(weight)).multiplication(akhir));
63
64             orderFinal[j+k] = awal.add(exponential);
65             orderFinal[j+k+(i/2)] = awal.minus(exponential);
66         }
67     }
68 }
69     return orderFinal;
70 }
71
72 public void toString(Complex[] complex) {
73     for (int i=0; i<complex.length; i++) {
74         System.out.println(complex[i].absolute());
75     }
76 }
77 }
```

Kode A.7: RenderChart.java

```

1 import java.io.IOException;
2
3 public interface RenderChart {
4     public void render() throws IOException;
5     public static final int MAX_COUNTER = 64;
6 }
```

Kode A.8: SampleData.java

```

1 import java.io.IOException;
2 import java.util.ArrayList;
3 import java.util.Collections;
4 import java.util.List;
5
6 public class SampleData {
7
8     public String sensorId;
9     public List<Double> X;
10    public List<Double> Y;
11    public List<Double> Z;
12    public final int MAX;
13    private int counter = 0;
14    private RenderChart rc;
15
16    public SampleData(int banyak) {
17        MAX = banyak;
18        this.X = Collections.synchronizedList(new ArrayList<>());
19        this.Y = Collections.synchronizedList(new ArrayList<>());
20        this.Z = Collections.synchronizedList(new ArrayList<>());
21    }
22
23    public void setRender(RenderChart rc) {
24        this.rc = rc;
25    }
26
27    public void addX(double titikX) {
28        this.X.add(titikX);
29        if (this.X.size() > MAX) {
30            this.X.remove(0);
31        }
32    }
33
34    public void addY(double titikY) {
35        this.Y.add(titikY);
36        if (this.Y.size() > MAX) {
37            this.Y.remove(0);
38        }
39    }
40
41    public void addZ(double titikZ) throws IOException {
42        this.Z.add(titikZ);
43        if (this.Z.size() > MAX) {
44            this.Z.remove(0);
45        }
46        counter++;
47        if(counter == RenderChart.MAX_COUNTER) {
48            counter = 0;
49            rc.render();
50        }
51    }
52
53    public Double[] getX() {
54        return this.X.toArray(new Double[this.X.size()]);
55    }
56
57    public Double[] getY() {
58        return this.Y.toArray(new Double[this.Y.size()]);
59    }
60
61    public Double[] getZ() {
62        return this.Z.toArray(new Double[this.Z.size()]);
63    }
64 }
```

```
63 }
64 }
```

### Kode A.9: Sensor.java

```

1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6
7 //import org.apache.tools.ant.BuildException;
8 //import org.apache.tools.ant.DefaultLogger;
9 //import org.apache.tools.ant.Project;
10 //import org.apache.tools.ant.ProjectHelper;
11 //
12 //import com.virtenio.commander.io.DataConnection;
13 //import com.virtenio.commander.toolsets.preon32.Preon32Helper;
14 import java.io.BufferedInputStream;
15 import java.io.BufferedReader;
16 import java.io.File;
17 import java.io.IOException;
18 import java.io.InputStreamReader;
19 import java.text.SimpleDateFormat;
20 import java.util.Date;
21 import java.util.HashMap;
22 import java.util.Scanner;
23 import java.util.logging.Level;
24 import java.util.logging.Logger;
25
26 import org.apache.tools.ant.BuildException;
27 import org.apache.tools.ant.DefaultLogger;
28 import org.apache.tools.ant.Project;
29 import org.apache.tools.ant.ProjectHelper;
30
31 import com.virtenio.commander.io.DataConnection;
32 import com.virtenio.commander.toolsets.preon32.Preon32Helper;
33
34 import javafx.application.Application;
35 import javafx.application.Platform;
36 import javafx.stage.Stage;
37
38 public class Sensor extends Application {
39
40     public static void main(String[] args) {
41         Sensor.launch(args);
42     }
43     private Thread threadSensing;
44     private BufferedInputStream bufferedInput;
45     private DataConnection dataCon;
46     private boolean sensing;
47     private HashMap<String, StartChart> chartAmp = new HashMap<String, StartChart>();
48
49 ;
50
51 // console build ant
52     private static DefaultLogger getConsoleLogger() {
53         DefaultLogger consoleLogger = new DefaultLogger();
54         consoleLogger.setErrorPrintStream(System.err);
55         consoleLogger.setOutputPrintStream(System.out);
56         consoleLogger.setMessageOutputLevel(Project.MSG_INFO);
57         return consoleLogger;
58     }
59
60     private static String timeFormat(long timeMillis) {
61         Date date = new Date(timeMillis);
62         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss,SSS");
63         return simpleDateFormat.format(date);
64     }
65
66 // ant build time synchronize
67     private void time_synchronize() throws Exception {
68         DefaultLogger consoleLogger = getConsoleLogger();
69         File buildFile = new File("E:\\Skripsi\\Skripsi_WSN\\Sandbox\\build.xml");
70         Project antProject = new Project();
71         antProject.setUserProperty("ant.file", buildFile.getAbsolutePath());
72         antProject.addBuildListener(consoleLogger);
73         try {
74             antProject.fireBuildStarted();
75             antProject.init();
76             ProjectHelper helper = ProjectHelper.getProjectHelper();
77             antProject.addReference("ant.ProjectHelper", helper);
78             helper.parse(antProject, buildFile);
79             String target = "cmd.time.synchronize";
80             antProject.executeTarget(target);
81             antProject.fireBuildFinished(null);
82         } catch (BuildException e) {
83     }
84     }
85
86 // set context basestation
87     private void context_set(String target) throws Exception {
88         DefaultLogger consoleLogger = getConsoleLogger();
89         File buildFile = new File("E:\\Skripsi\\Skripsi_WSN\\Sandbox\\buildUser.xml");
90         Project antProject = new Project();
91         antProject.setUserProperty("ant.file", buildFile.getAbsolutePath());
92         antProject.addBuildListener(consoleLogger);
93         try {

```

```

94    antProject.fireBuildStarted();
95    antProject.init();
96    ProjectHelper helper = ProjectHelper.getProjectHelper();
97    antProject.addReference("ant.ProjectHelper", helper);
98    helper.parse(antProject, buildFile);
99    antProject.executeTarget(target);
100   antProject.fireBuildFinished(null);
101  }
102 }

103 public StartChart getChart(String sensorId) {
104  if (!chartAmp.containsKey(sensorId)) {
105   Platform.runLater(new Runnable() {
106    @Override
107    public void run() {
108     StartChart sc = new StartChart(sensorId);
109     sc.start();
110     chartAmp.put(sensorId, sc);
111    }
112   });
113  }
114  while(!chartAmp.containsKey(sensorId)) {
115  }
116  return chartAmp.get(sensorId);
117 }
118 }

119 @Override
120 public void start(Stage primaryStage) throws Exception {
121  new Thread() {
122   @Override
123   public void run() {
124
125    Long current = System.currentTimeMillis();
126    //Input User
127    try {
128     Scanner sc = new Scanner(System.in);
129     context_set("context.set.1");
130     time_synchronize();
131
132     /*
133      * Com nya khusus basestation
134      * module nama sama
135     */
136     Preon32Helper nodeHelper = new Preon32Helper("COM4", 115200);
137     dataCon = nodeHelper.runModule("basestation2");
138     bufferedInput = new BufferedInputStream(dataCon.getInputStream());
139     dataCon.flush();
140     while (true) {
141      System.out.println("1._Check_Node_Online_Status");
142      System.out.println("2._Start_Sensing");
143      System.out.println("3._Stop_Sensing");
144      System.out.println("4._Exit_Program");
145      int masukan = sc.nextInt();
146
147      if (sensing) {
148       if (masukan == 1 || masukan == 2 || masukan == 4) {
149        System.out.println("Still_in_Sensing_State!!!");
150        masukan = 0;
151       } else {
152        dataCon.write(masukan);
153       }
154      } else {
155       dataCon.write(masukan);
156      }
157      byte[] buffer = new byte[1024];
158      if (masukan == 1) {
159       System.out.println("Online_Node_:_");
160       Thread.sleep(500);
161       while (bufferedInput.available() > 0) {
162        bufferedInput.read(buffer);
163        dataCon.flush();
164
165        String temp = new String(buffer).trim();
166        String[] lines = temp.split("\n");
167        for(String s : lines) {
168         String[] words = s.split(" ");
169         if(words[0].charAt(0) == '1') {
170
171          System.out.printf("%s_%s_%s\n",words[0],words[1],timeFormat(current));
172         }else {
173
174          System.out.printf("%s_%s_%s\n",words[0],words[1],timeFormat(current));
175         }
176        }
177      }
178    }
179  } else if (masukan == 2) {
180   System.out.println("Sensing...");
181   sensing = true;
182
183   if (threadSensing == null) {
184    threadSensing = new Thread() {
185     public void run() {
186      while (sensing) {
187       try {
188        if (bufferedInput.available() > 0) {
189         byte[] bytes = new byte[1024];
190         BufferedReader br = new BufferedReader(new InputStreamReader(dataCon.
191           getInputStream()));
192         dataCon.flush();
193
194        }
195       }
196      }
197     }
198    }
199   }
200  }
201 }

202 
```

```

192     String temp = br.readLine();
193     System.out.println(temp);
194     String[] hasil = temp.split(",");
195     if (hasil[0].charAt(0) == '2') {
196         String sensorid = hasil[0].substring(1);
197         final StartChart ct = getChart(sensorid);
198         ct.addData(hasil);
199     }
200     br.close();
201 } catch (IOException e) {
202     System.out.println(e.getMessage());
203 }
204 }
205 }
206 };
207 threadSensing.start();
208 } else if (masukan == 3) {
209     if (sensing) {
210         System.out.println("Stopping_sense..");
211         sensing = false;
212         threadSensing = null;
213         System.out.println("Sensing_has_been_stopped!!!");
214     } else if (!sensing) {
215         System.out.println("Sensor_does_not_in_sensing_state!!!");
216     }
217 } else if (masukan == 4) {
218     System.out.println("Terminating_program..");
219     sensing = false;
220     Thread.sleep(500);
221     System.out.println("Progam_has_terminated!!!");
222     System.exit(0);
223 }
224 }
225 }
226 } catch (Exception e) {
227     System.out.println("Error_occured!");
228 }
229 }
230 }.start();
231 }
232 }
233 }

```

Kode A.10: StartChart.java

```

1 import javafx.application.Platform;
2 import javafx.scene.Scene;
3 import javafx.scene.chart.CategoryAxis;
4 import javafx.scene.chart.LineChart;
5 import javafx.scene.chart.NumberAxis;
6 import javafx.scene.chart.XYChart;
7 import javafx.scene.layout.AnchorPane;
8 import javafx.stage.Stage;
9
10 import java.io.BufferedReader;
11 import java.io.FileWriter;
12 import java.io.IOException;
13 import java.text.DecimalFormat;
14 import java.text.SimpleDateFormat;
15 import java.util.ArrayList;
16 import java.util.Date;
17 import java.util.concurrent.Executors;
18 import java.util.concurrent.ScheduledExecutorService;
19 import java.util.concurrent.TimeUnit;
20
21 public class StartChart {
22
23     public ArrayList<Chart> charts = new ArrayList<>();
24     private SampleData sampleData = new SampleData(64);
25     private DataFrequency dataFrequency = new DataFrequency();
26     private Stage stage = new Stage();
27     private String sensorId;
28
29
30     public StartChart(String sensorId){
31         this.sensorId = sensorId;
32     }
33
34     public SampleData getSample() {
35         return sampleData;
36     }
37
38     public DataFrequency getData() {
39         return dataFrequency;
40     }
41
42     public Scene getScene() throws IOException {
43         charts.add(new ChartAmplitude(this.getSample(), this.getData(), sensorId));
44         this.getSample().setRender(charts.get(0));
45         charts.add(new ChartFrequency(this.getSample(), this.getData(), sensorId));
46         this.getData().setRender(charts.get(1));
47         AnchorPane ap = new AnchorPane();
48         int i = 0;
49         int height = 400;
50         int width = 600;
51         for (Chart chart : charts) {
52             LineChart lc = chart.getChart();
53             lc.relocate(0, height * i);

```

```

54         i++;
55         ap.getChildren().add(lc);
56         lc.setMinSize(width, height);
57         lc.setMaxSize(width, height);
58     }
59     return new Scene(ap, width, i * height);
60 }
61
62 public void addData(String[] hasil) throws IOException {
63     if (hasil[0].charAt(0) == '2') {
64         // Sample Data
65         this.getSample().addX(Double.parseDouble(hasil[1]));
66         this.getSample().addY(Double.parseDouble(hasil[2]));
67         this.getSample().addZ(Double.parseDouble(hasil[3]));
68
69         // Perhitungan FFT
70         FFT computeFFT = new FFT(this.getSample());
71         computeFFT.convertComplex();
72         Complex[] tempRes1 = computeFFT.fft(computeFFT.xComplex);
73         Complex[] tempRes2 = computeFFT.fft(computeFFT.yComplex);
74         Complex[] tempRes3 = computeFFT.fft(computeFFT.zComplex);
75         double t1 = 0, t2 = 0, t3 = 0;
76         DecimalFormat df = new DecimalFormat("#.####");
77         for (int i = 0; i < tempRes1.length; i++) {
78             t1 += tempRes1[i].absolute();
79             t2 += tempRes2[i].absolute();
80             t3 += tempRes3[i].absolute();
81         }
82         t1/= tempRes1.length;
83         t2/= tempRes2.length;
84         t3/= tempRes3.length;
85         System.out.printf("%f %f %f\n",t1,t2,t3);
86         this.getData().setX(Double.parseDouble(df.format(t1)));
87         this.getData().setY(Double.parseDouble(df.format(t2)));
88         this.getData().setZ(Double.parseDouble(df.format(t3)));
89
90     }
91 }
92
93 public void start(){
94     Platform.runLater(new Runnable() {
95         @Override
96         public void run() {
97             stage.setTitle(sensorId);
98             stage.show();
99             try {
100                 stage.setScene(getScene());
101             } catch (IOException e) {
102                 // TODO Auto-generated catch block
103                 e.printStackTrace();
104             }
105         }
106     });
107 }
108 }
109 abstract class Chart implements RenderChart {
110
111     public abstract LineChart getChart() throws IOException;
112
113     public SampleData sampleData;
114     public DataFrequency dataFrequency;
115     public String sensorId;
116     public BufferedWriter writer;
117
118     // buat nampilin datanya
119     XYChart.Series<String, Number> series1 = new XYChart.Series<>();
120
121     XYChart.Series<String, Number> series2 = new XYChart.Series<>();
122
123     XYChart.Series<String, Number> series3 = new XYChart.Series<>();
124
125     final SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
126 }
127
128
129 class ChartAmplitude extends Chart {
130
131     final int WINDOW_SIZE = 10;
132     private ScheduledExecutorService scheduledExecutorService;
133     public LineChart<String, Number> lineChart;
134     public ChartAmplitude(SampleData sample, DataFrequency df, String id) throws IOException {
135         this.sampleData = sample;
136         this.sensorId = id;
137         this.dataFrequency = df;
138         this.writer = new BufferedWriter(new FileWriter("hasilXYZ.txt"));
139     }
140
141     public void setSensorId(String sensorId) {
142         this.sensorId = sensorId;
143     }
144
145     public String getSensorId() {
146         return this.sensorId;
147     }
148
149     public LineChart<String, Number> getChart() throws IOException {
150         final CategoryAxis x = new CategoryAxis(); // buat sumbu x (waktu)
151         final NumberAxis y = new NumberAxis(); // buat sumbu y (hasil sensor)
152

```

```

153     x.setLabel("Time");
154     x.setAnimated(false);
155     y.setLabel("Value_(g)");
156     y.setAnimated(false);
157
158     // bikin line chart
159     lineChart = new LineChart<x, y>;
160     lineChart.setAnimated(false);
161
162     series1.setName("Sumbu_X");
163     series2.setName("Sumbu_Y");
164     series3.setName("Sumbu_Z");
165
166     // add series to chart
167     lineChart.getData().add(series1);
168     lineChart.getData().add(series2);
169     lineChart.getData().add(series3);
170
171     // setup scene
172
173     // setup executor to put data periodically
174     scheduledExecutorService = Executors.newSingleThreadScheduledExecutor();
175
176     render();
177     return this.lineChart;
178 }
179
180
181 public void render() throws IOException{
182
183     Platform.runLater(new Runnable() {
184
185         @Override
186         public void run() {
187
188             Date now = new Date();
189             lineChart.setTitle("Grafik_Amplitudo_" +sensorId);
190
191             try {
192                 String baris = sensorId+", "+getX()+" , "+getY()+" , "+getZ();
193                 writer.write(baris);
194                 writer.newLine();
195                 writer.flush();
196             }
197             catch(Exception e) {
198                 e.getMessage();
199             }
200             // taro random number dengan waktu skrg
201             series1.getData().add(new XYChart.Data<>(simpleDateFormat.format(now), getX()));
202             series2.getData().add(new XYChart.Data<>(simpleDateFormat.format(now), getY()));
203             series3.getData().add(new XYChart.Data<>(simpleDateFormat.format(now), getZ()));
204             if (series1.getData().size() > WINDOW_SIZE) {
205                 series1.getData().remove(0);
206                 series2.getData().remove(0);
207                 series3.getData().remove(0);
208             }
209
210         }
211     });
212 }
213
214 public double getX() {
215     double hasil = 0;
216     for (double b : sampleData.X) {
217         hasil += b;
218     }
219     return hasil/sampleData.X.size();
220 }
221
222 public double getY() {
223     double hasil = 0;
224     for (double b : sampleData.Y) {
225         hasil += b;
226     }
227     return hasil/sampleData.Y.size();
228 }
229
230 public double getZ() {
231     double hasil = 0;
232     for (double b : sampleData.Z) {
233         hasil += b;
234     }
235     return hasil/sampleData.Z.size();
236 }
237 }
238
239 class ChartFrequency extends Chart {
240
241     private ScheduledExecutorService scheduledExecutorService;
242     final int WINDOW_SIZE = 10;
243     public LineChart<String, Number> lineChart;
244     public static double tempX = Double.MAX_VALUE;
245     public static double tempY = Double.MAX_VALUE;
246     public static double tempZ = Double.MAX_VALUE;
247
248     public ChartFrequency(SampleData sample, DataFrequency df, String id) throws IOException{
249         this.sampleData = sample;
250         this.sensorId = id;
251         this.dataFrequency = df;

```

```

252     this.writer = new BufferedWriter(new FileWriter("hasilFFT.txt"));
253 }
254
255 public void setSensorId(String sensorId) {
256     this.sensorId = sensorId;
257 }
258
259 public String getSensorId() {
260     return this.sensorId;
261 }
262
263 public LineChart<String, Number> getChart() {
264     final CategoryAxis x = new CategoryAxis(); // buat sumbu x (waktu)
265     final NumberAxis y = new NumberAxis(); // buat sumbu y (hasil sensor)
266
267     x.setLabel("Time");
268     x.setAnimated(false);
269     y.setLabel("Value_(Hz)");
270     y.setAnimated(false);
271
272     // bikin line chart
273     lineChart = new LineChart<>(x, y);
274     lineChart.setAnimated(false);
275
276     // buat nampolin datanya
277     series1.setName("Sumbu_X");
278
279     series2.setName("Sumbu_Y");
280
281     series3.setName("Sumbu_Z");
282
283     // add series to chart
284     lineChart.getData().add(series1);
285     lineChart.getData().add(series2);
286     lineChart.getData().add(series3);
287
288     render();
289     return this.lineChart;
290 }
291
292 public void render() {
293     Platform.runLater(new Runnable() {
294
295         @Override
296         public void run() {
297             lineChart.setTitle("Grafik_Frekuensi_" +sensorId);
298             Date now = new Date();
299
300             try {
301                 String baris = sensorId+",,"+dataFrequency.X+",,"+dataFrequency.Y+",,"+dataFrequency.Z;
302                 writer.write(baris);
303                 writer.newLine();
304                 writer.flush();
305             }
306             catch(Exception e) {
307                 e.getMessage();
308             }
309             series1.getData()
310                 .add(new XYChart.Data<>(simpleDateFormat.format(now), dataFrequency.X));
311             series2.getData()
312                 .add(new XYChart.Data<>(simpleDateFormat.format(now), dataFrequency.Y));
313             series3.getData()
314                 .add(new XYChart.Data<>(simpleDateFormat.format(now), dataFrequency.Z));
315             tempX = dataFrequency.X;
316             tempY = dataFrequency.Y;
317             tempZ = dataFrequency.Z;
318             if (series1.getData().size() > WINDOW_SIZE) {
319                 series1.getData().remove(0);
320                 series2.getData().remove(0);
321                 series3.getData().remove(0);
322             }
323
324         }
325     });
326 }
327
328 }
329

```



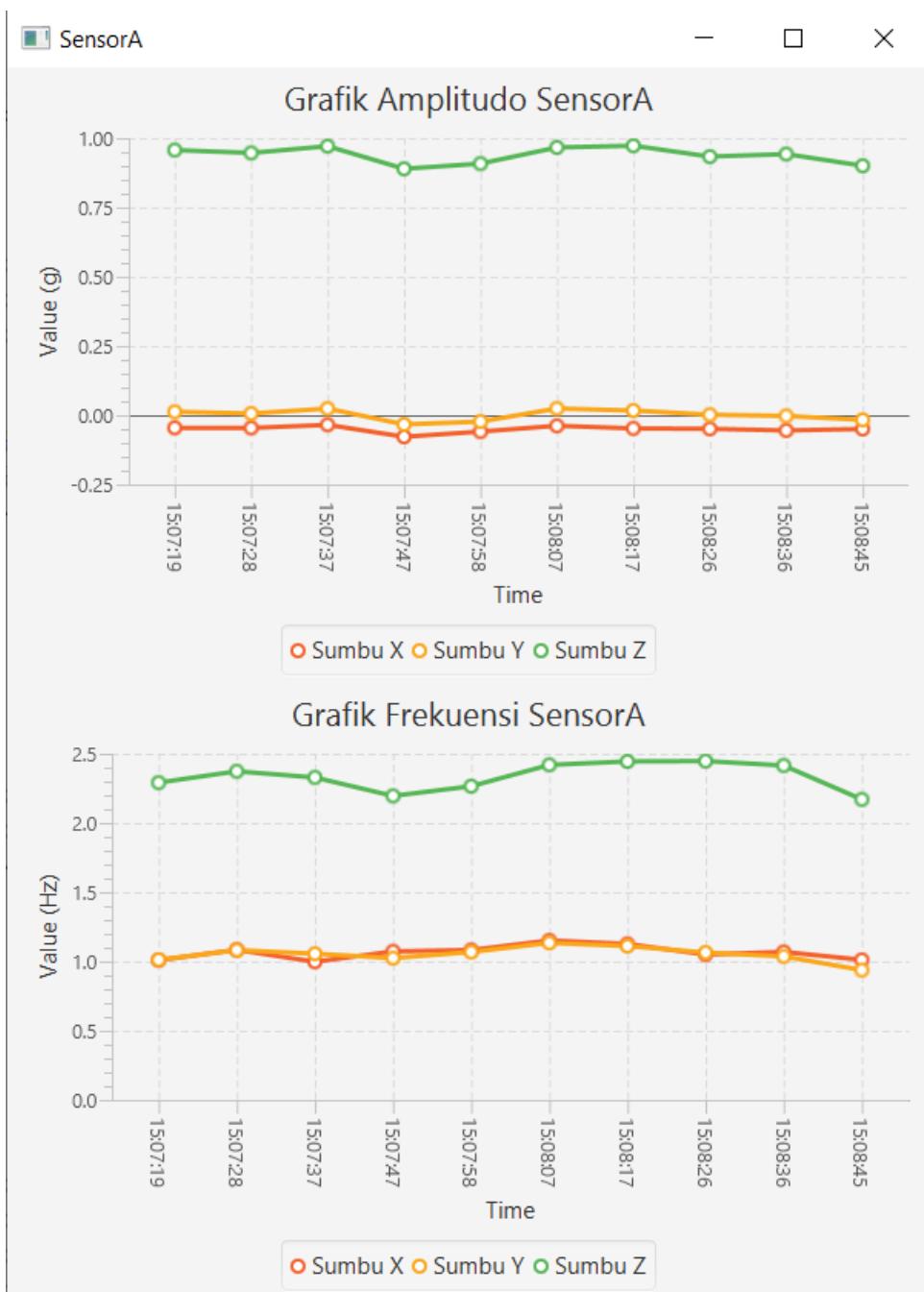


## LAMPIRAN B

### HASIL EKSPERIMENT DI MALL PASKAL 23 SQUARE

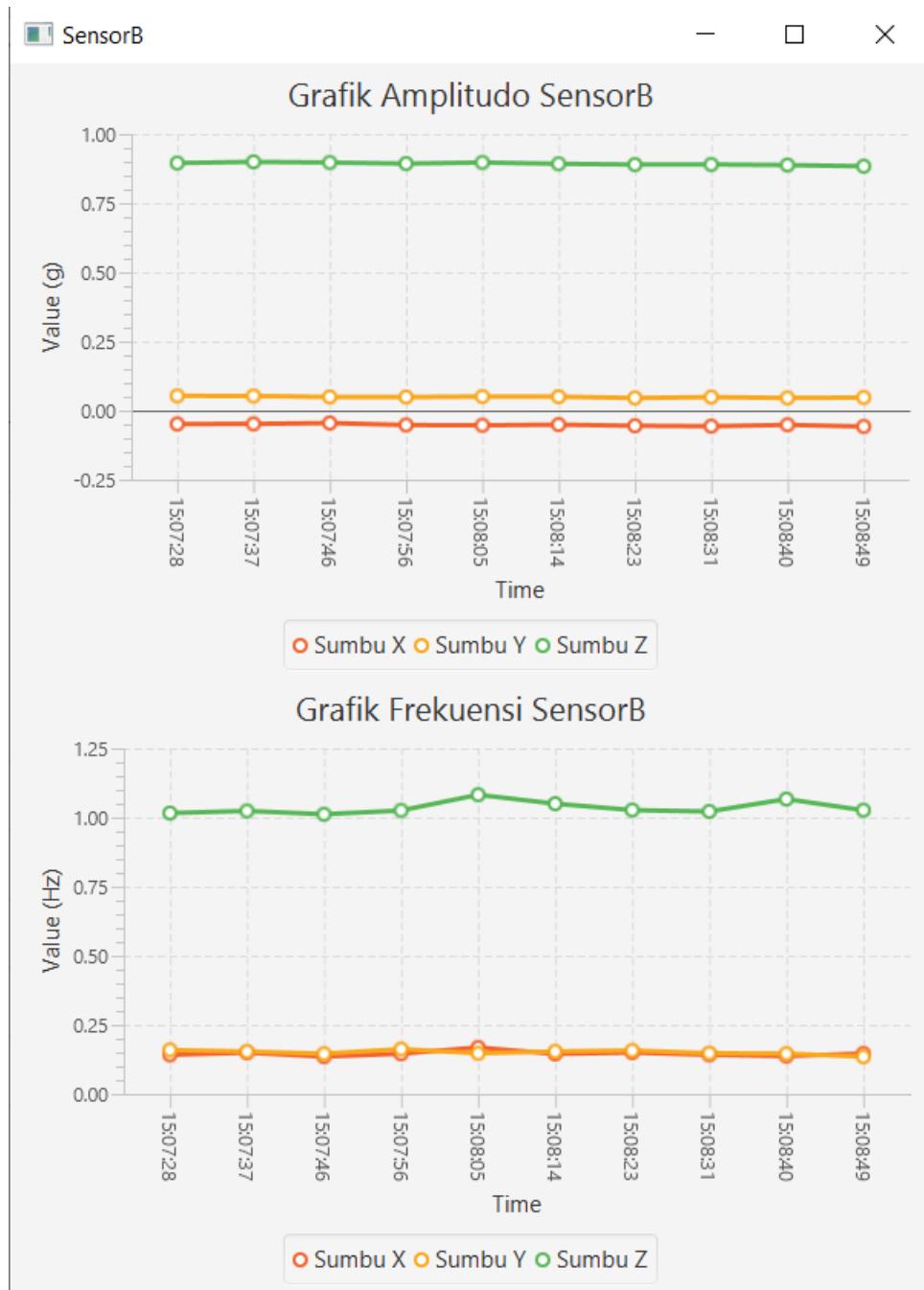
#### B.1 Topologi Star

##### B.1.1 SensorA



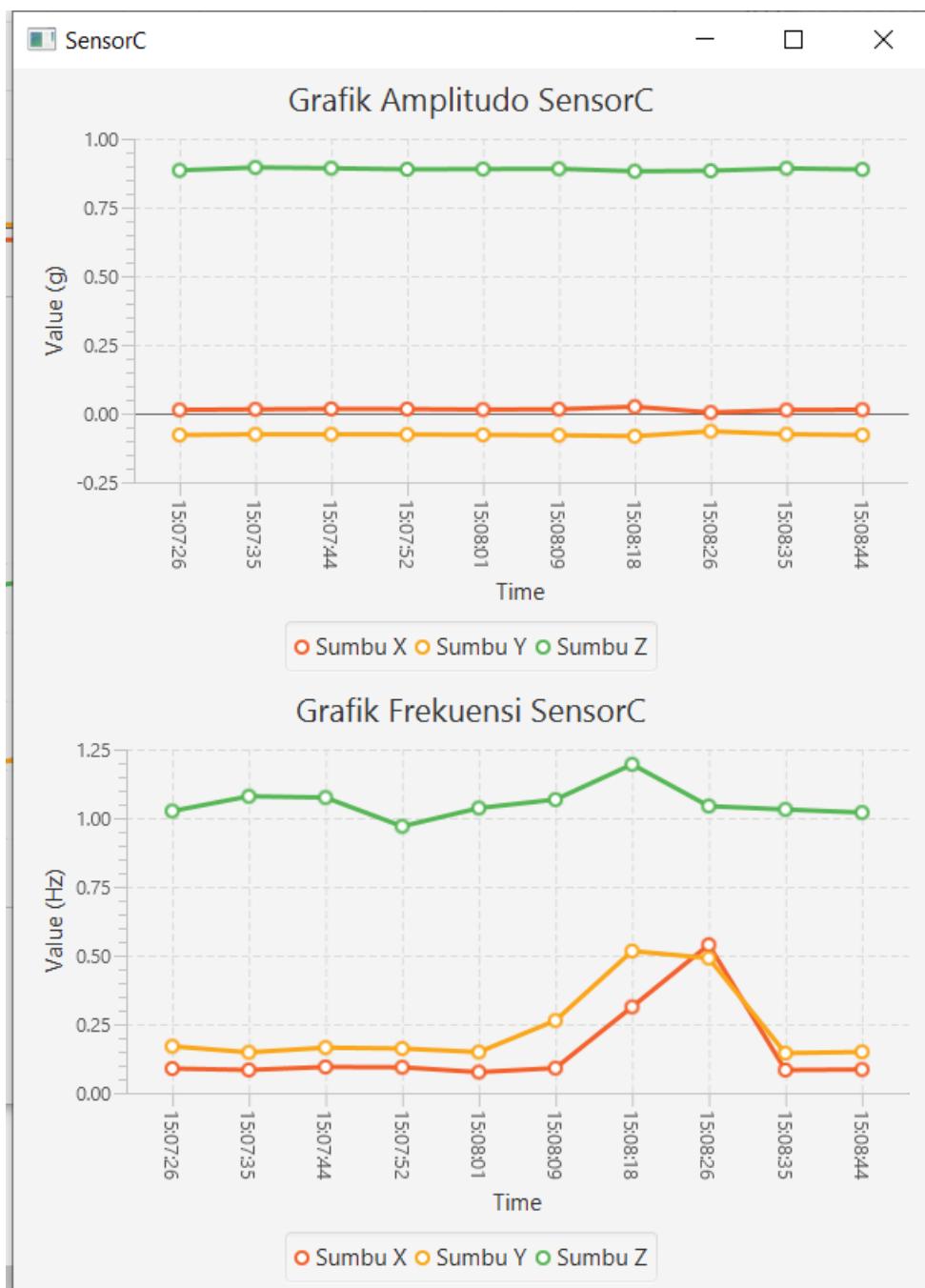
Gambar B.1: Grafik Monitoring getaran SensorA dengan topologi star

### B.1.2 SensorB



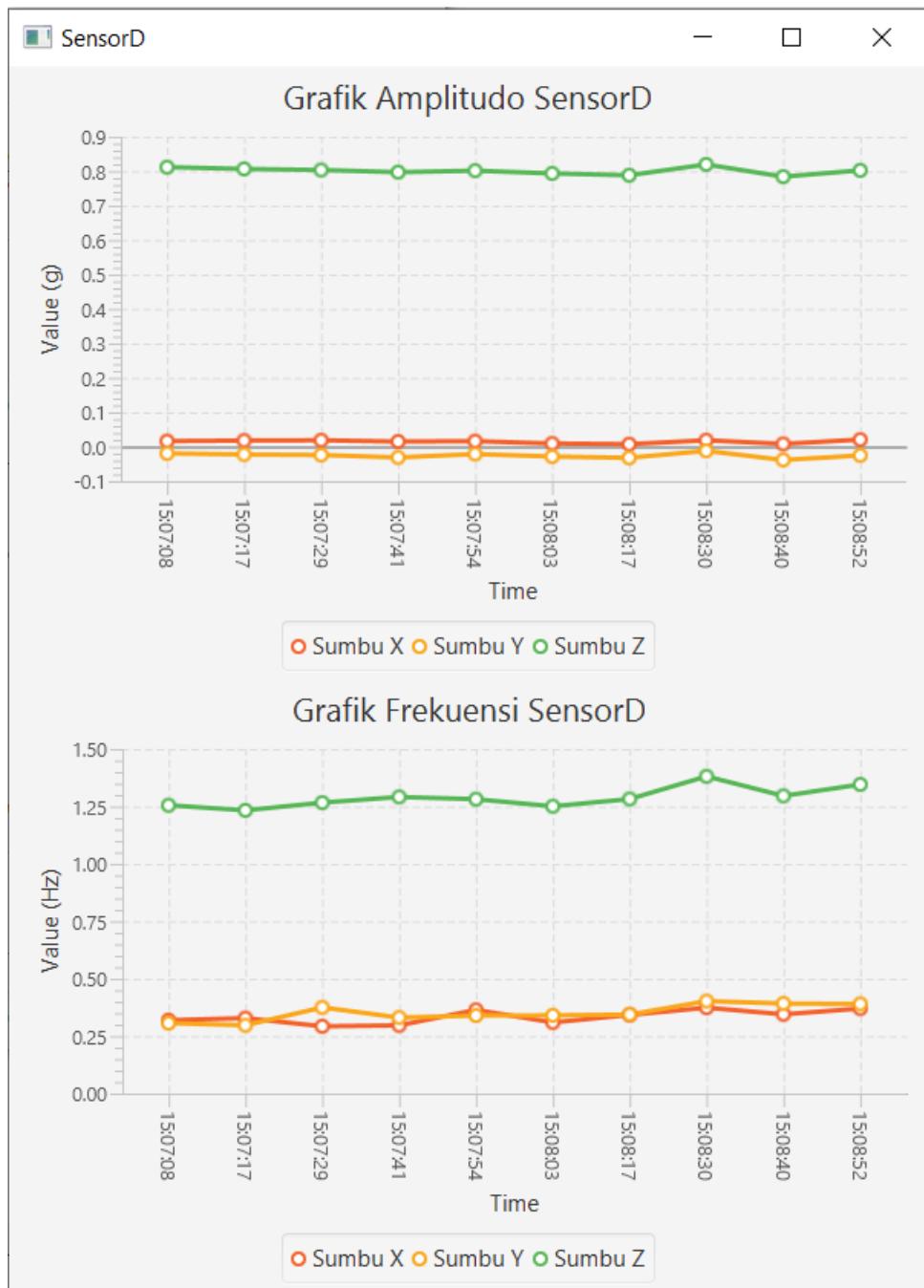
Gambar B.2: Grafik Monitoring getaran SensorB dengan topologi star

### B.1.3 SensorC



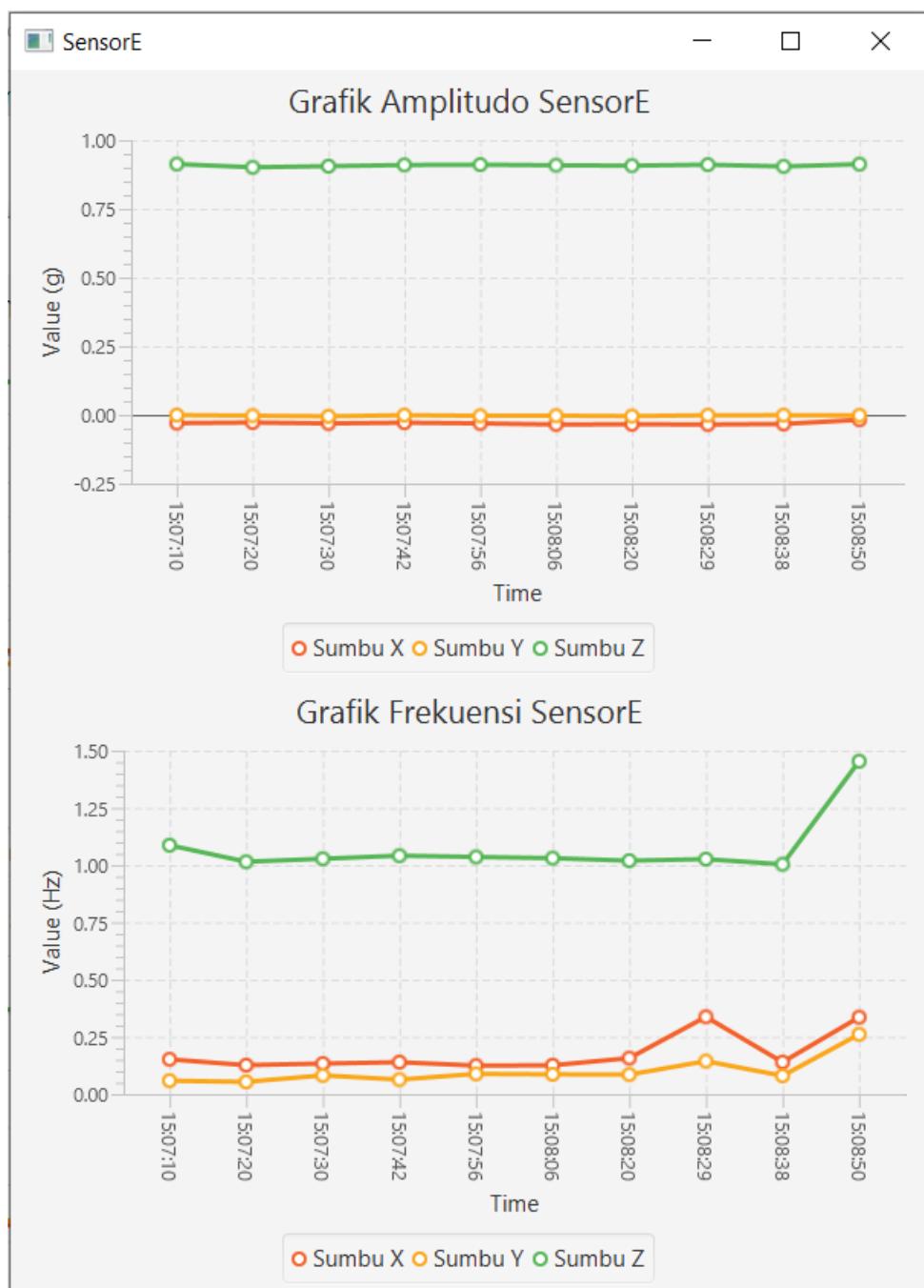
Gambar B.3: Grafik Monitoring getaran SensorC dengan topologi star

#### B.1.4 SensorD



Gambar B.4: Grafik Monitoring getaran SensorD dengan topologi star

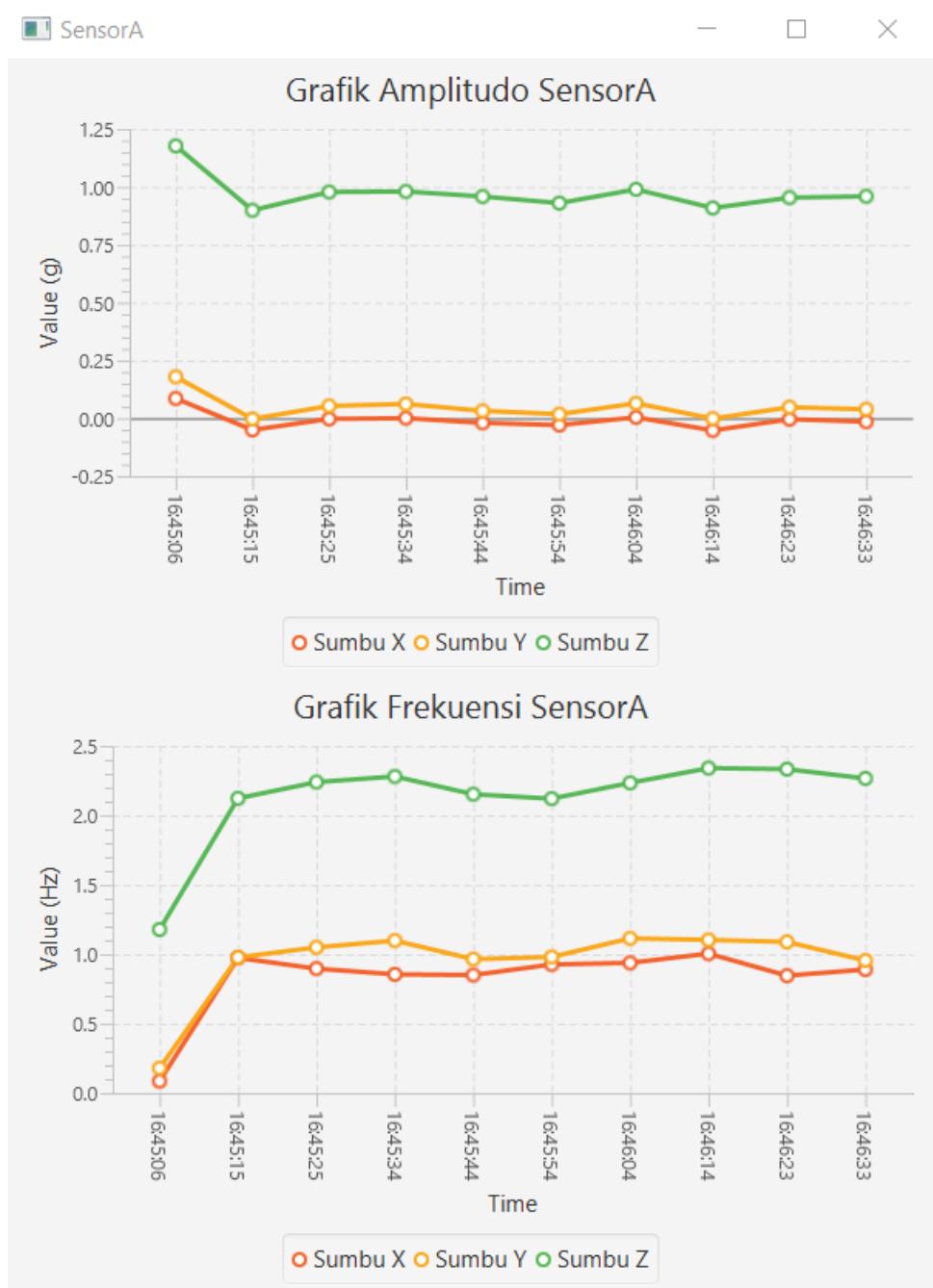
### B.1.5 SensorE



Gambar B.5: Grafik Monitoring getaran SensorE dengan topologi star

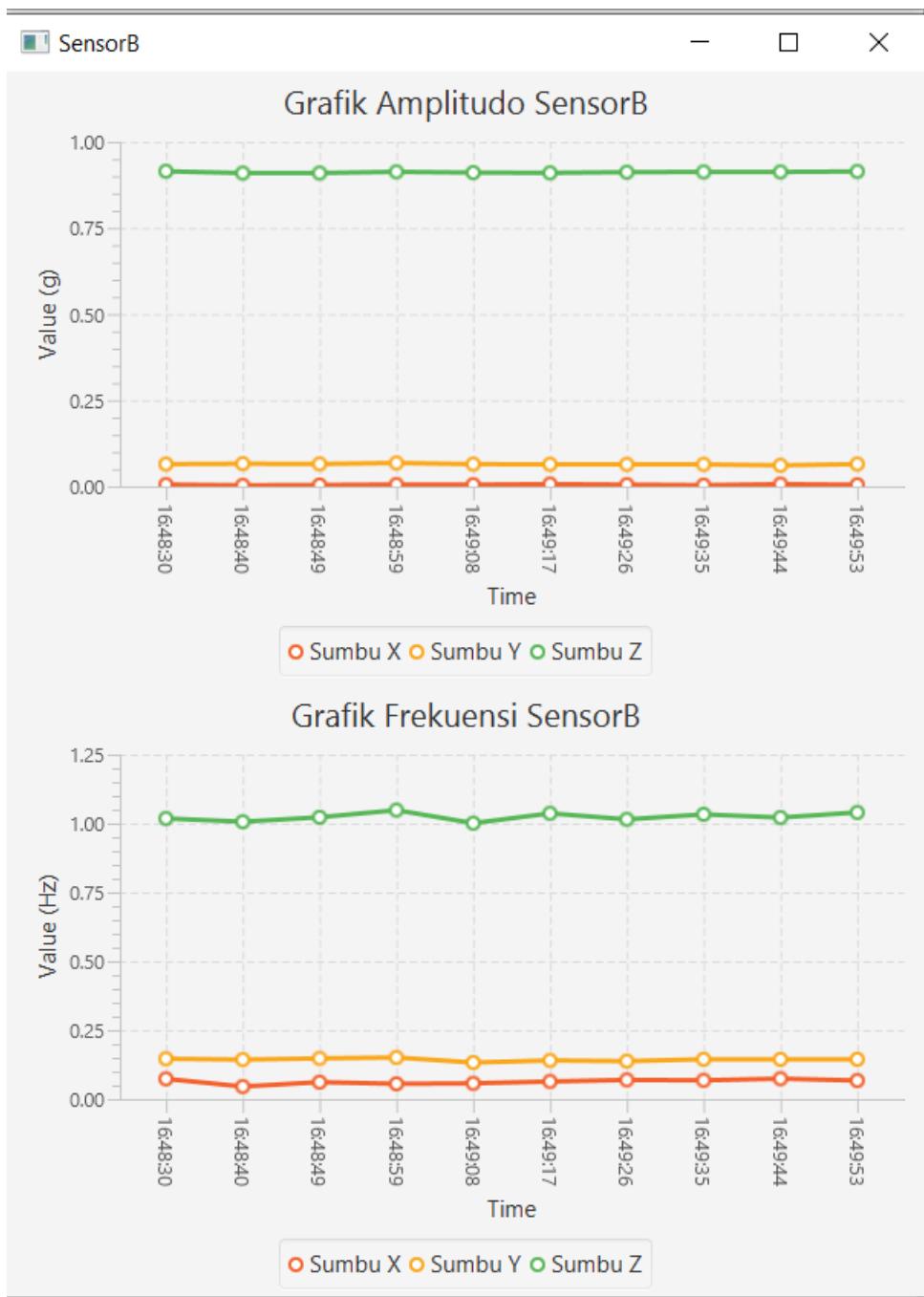
## B.2 Topologi Tree

### B.2.1 SensorA



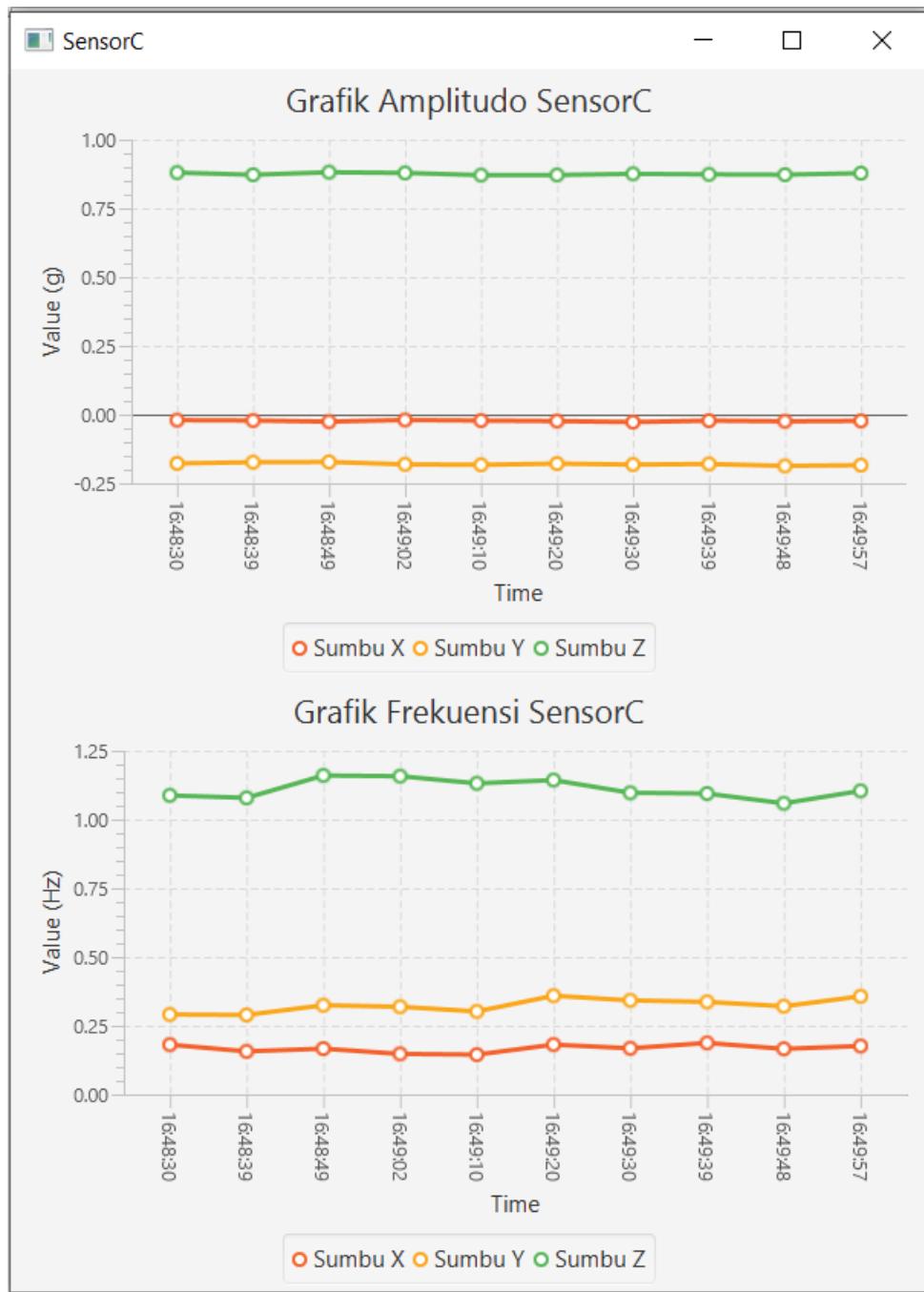
Gambar B.6: Grafik Monitoring getaran SensorA dengan topologi tree

### B.2.2 SensorB



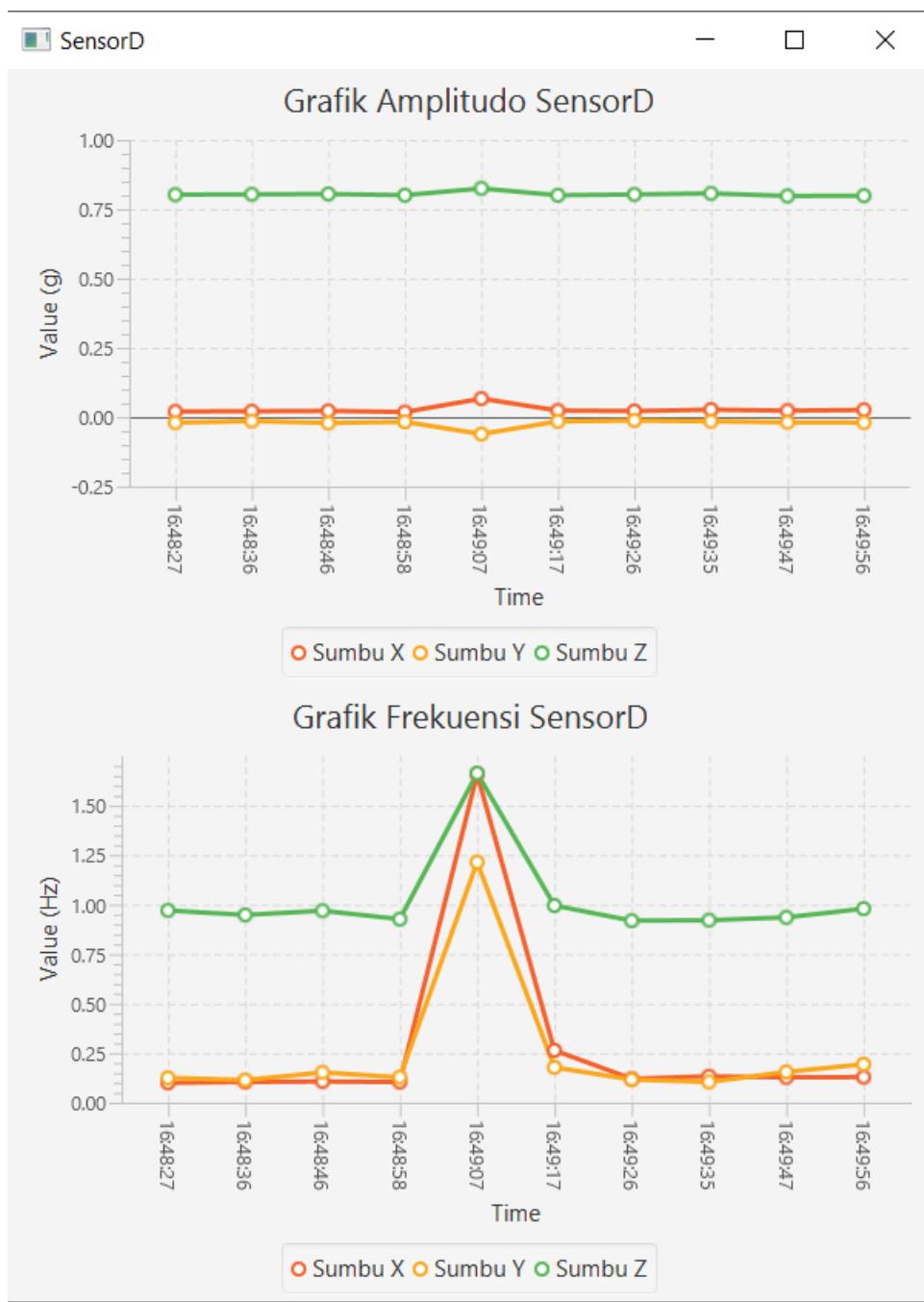
Gambar B.7: Grafik Monitoring getaran SensorB dengan topologi tree

### B.2.3 SensorC



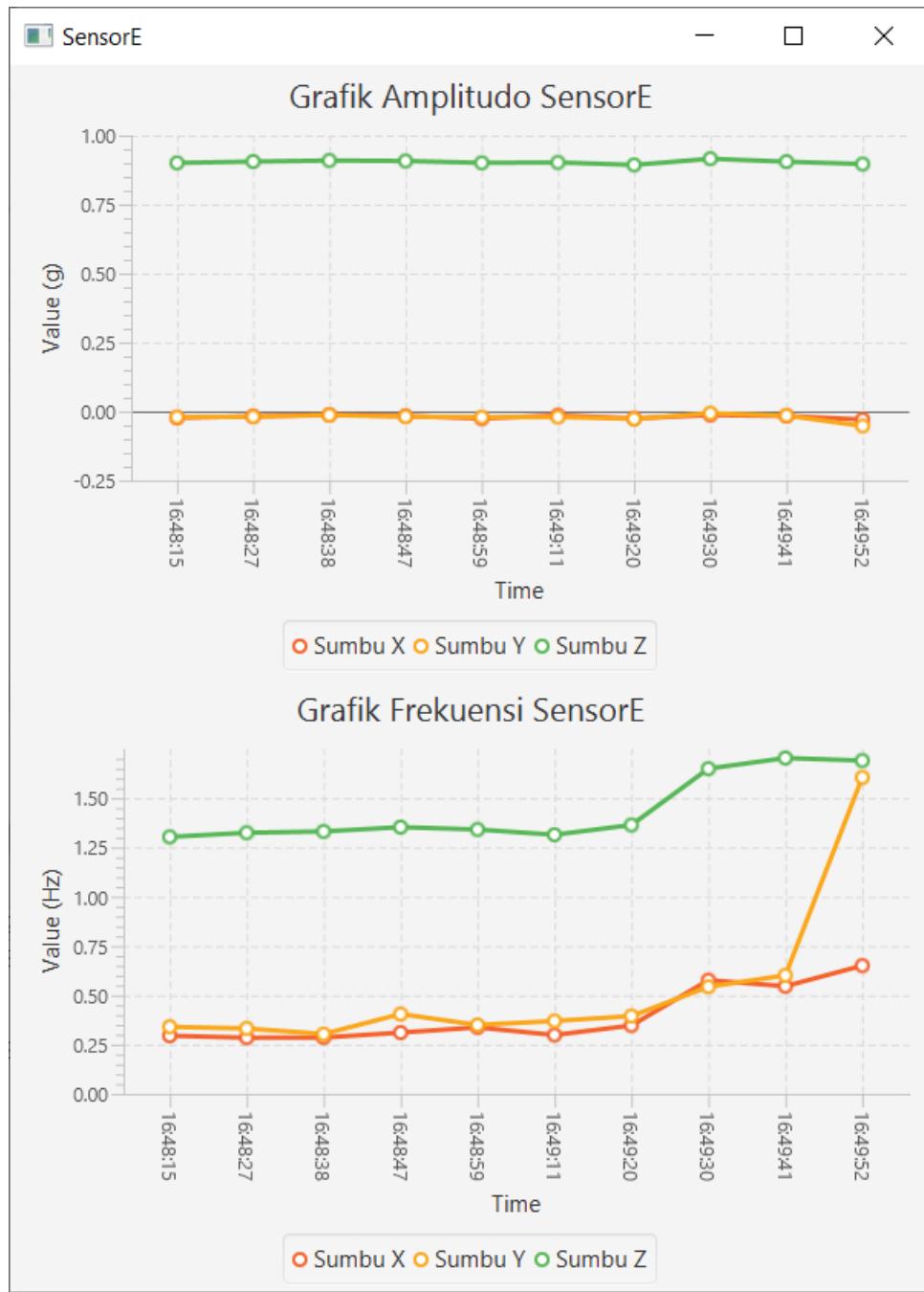
Gambar B.8: Grafik Monitoring getaran SensorC dengan topologi tree

#### B.2.4 SensorD



Gambar B.9: Grafik Monitoring getaran SensorD dengan topologi tree

### B.2.5 SensorE



Gambar B.10: Grafik Monitoring getaran SensorE dengan topologi tree



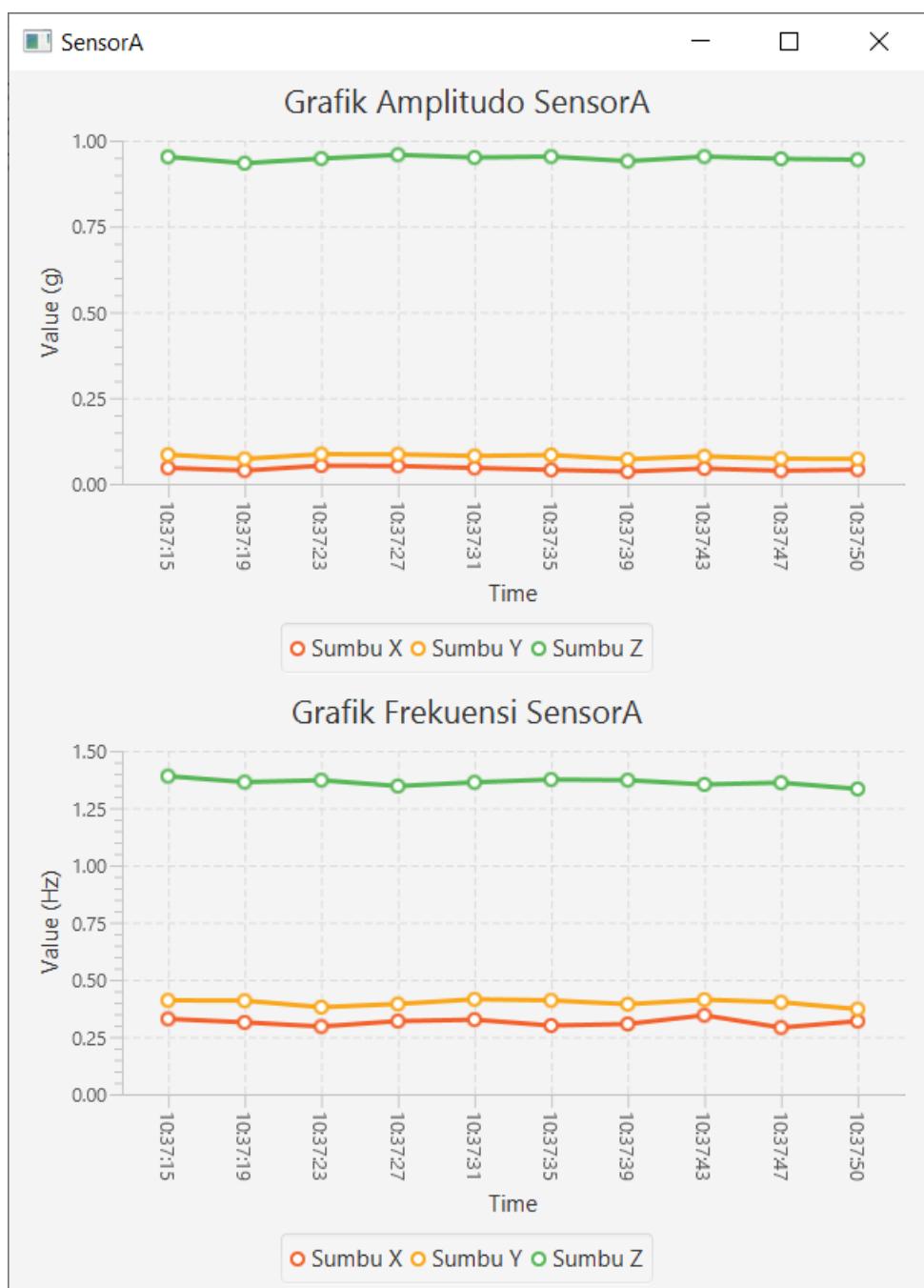


## LAMPIRAN C

### HASIL EKSPERIMEN DI ROOFTOP GEDUNG 10 UNPAR

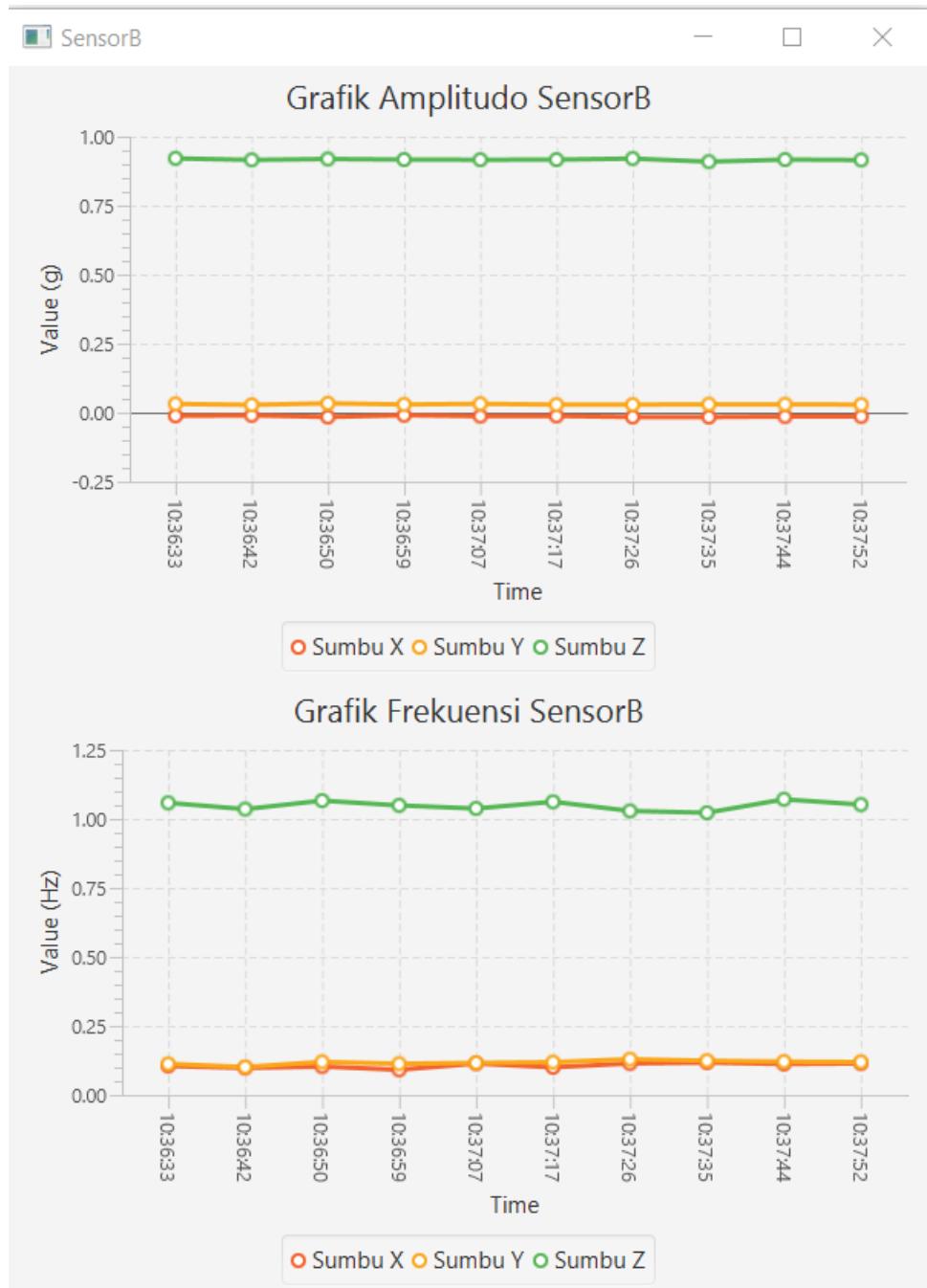
#### C.1 Topologi Star

##### C.1.1 SensorA



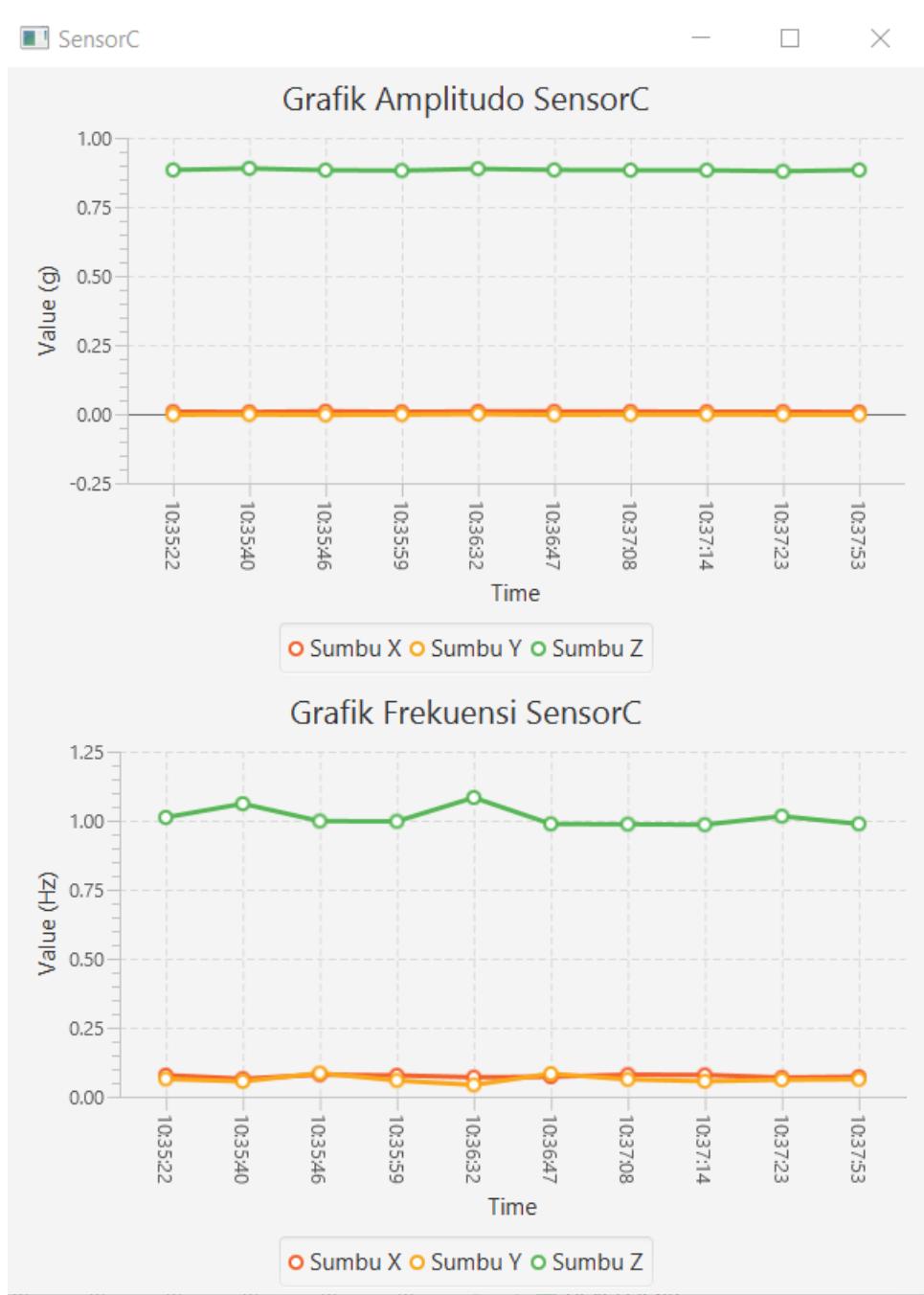
Gambar C.1: Grafik Monitoring getaran SensorA dengan topologi star

### C.1.2 SensorB



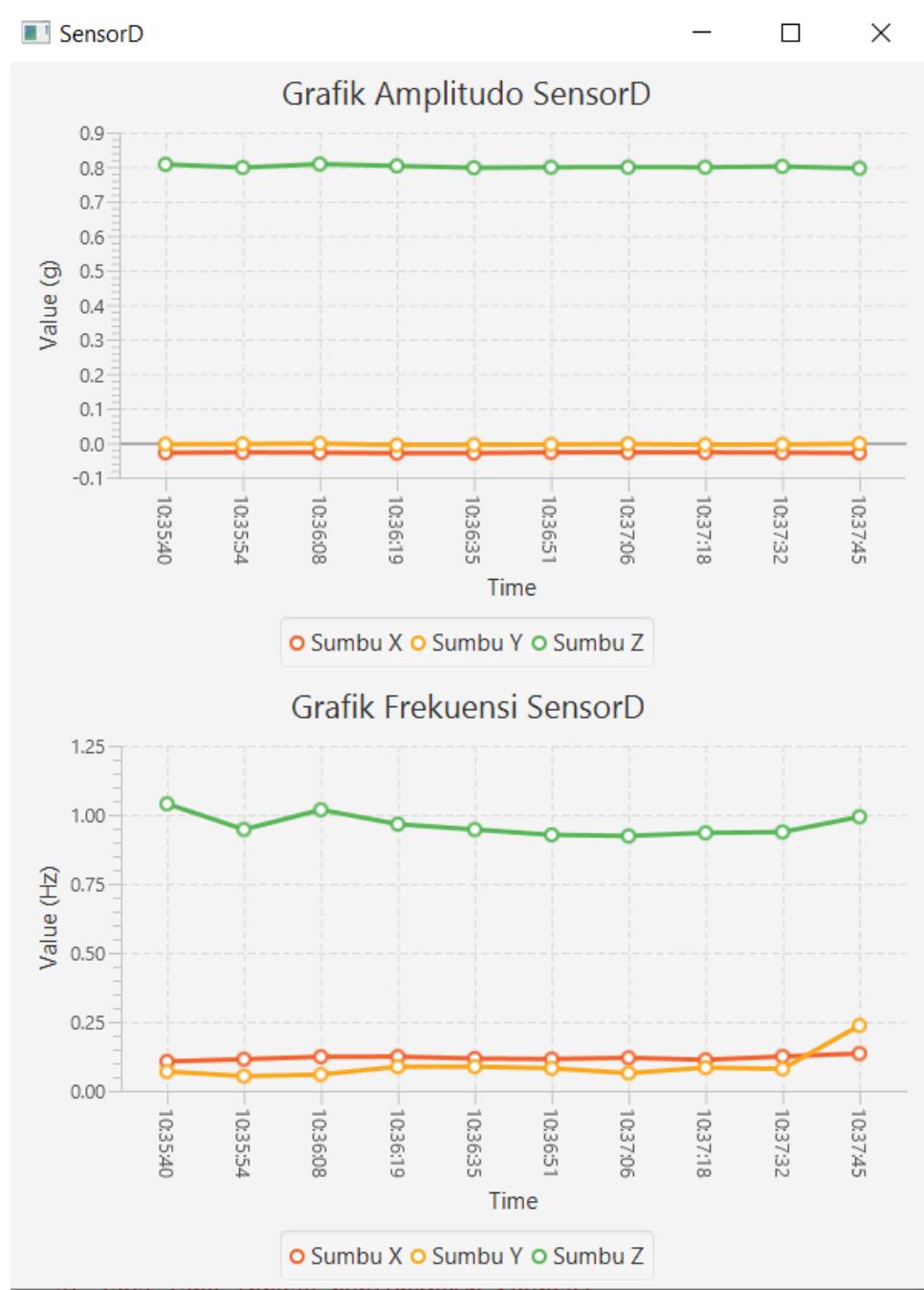
Gambar C.2: Grafik Monitoring getaran SensorB dengan topologi star

### C.1.3 SensorC



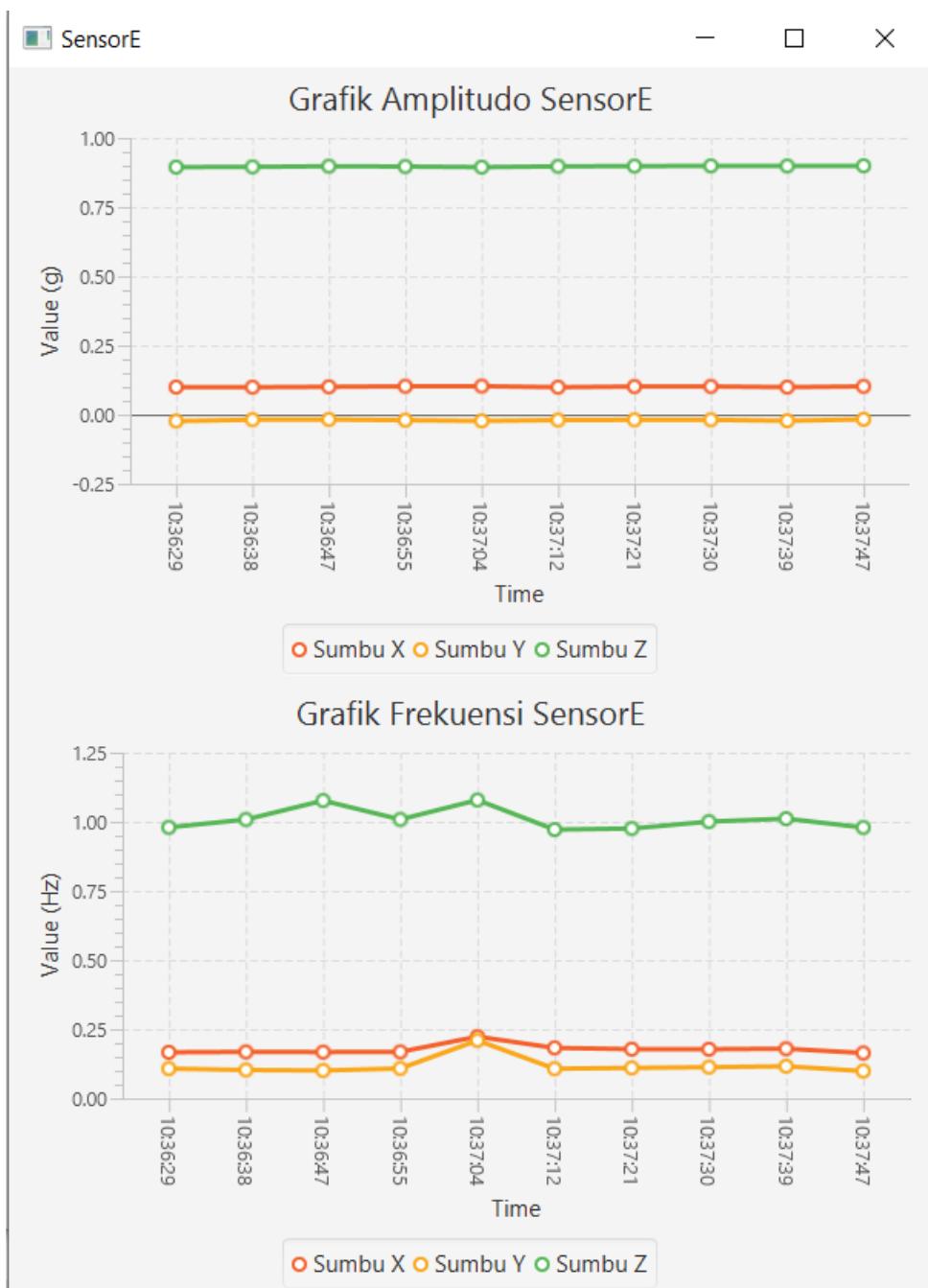
Gambar C.3: Grafik Monitoring getaran SensorC dengan topologi star

#### C.1.4 SensorD



Gambar C.4: Grafik Monitoring getaran SensorD dengan topologi star

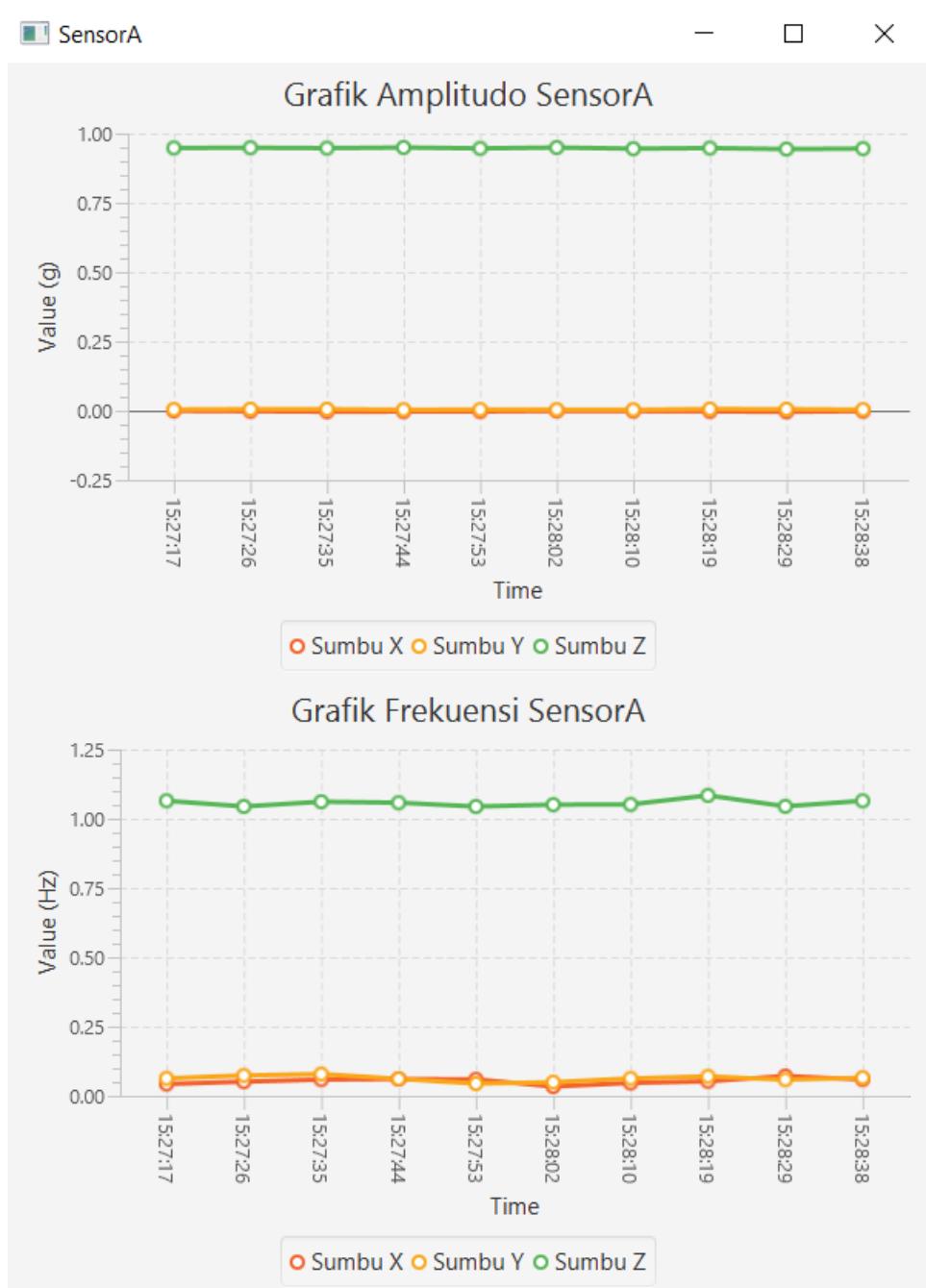
### C.1.5 SensorE



Gambar C.5: Grafik Monitoring getaran SensorE dengan topologi star

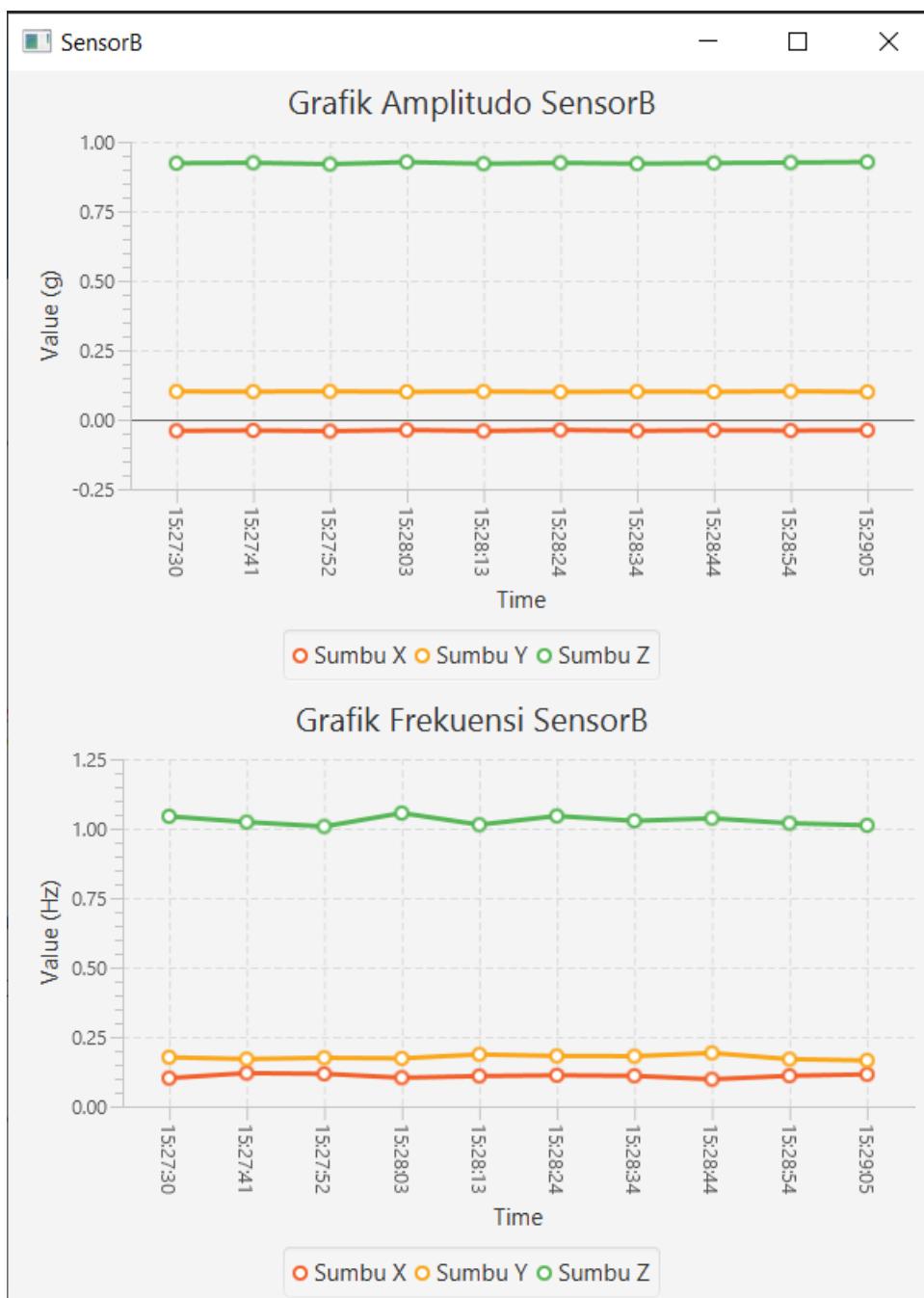
## C.2 Topologi Tree

### C.2.1 SensorA



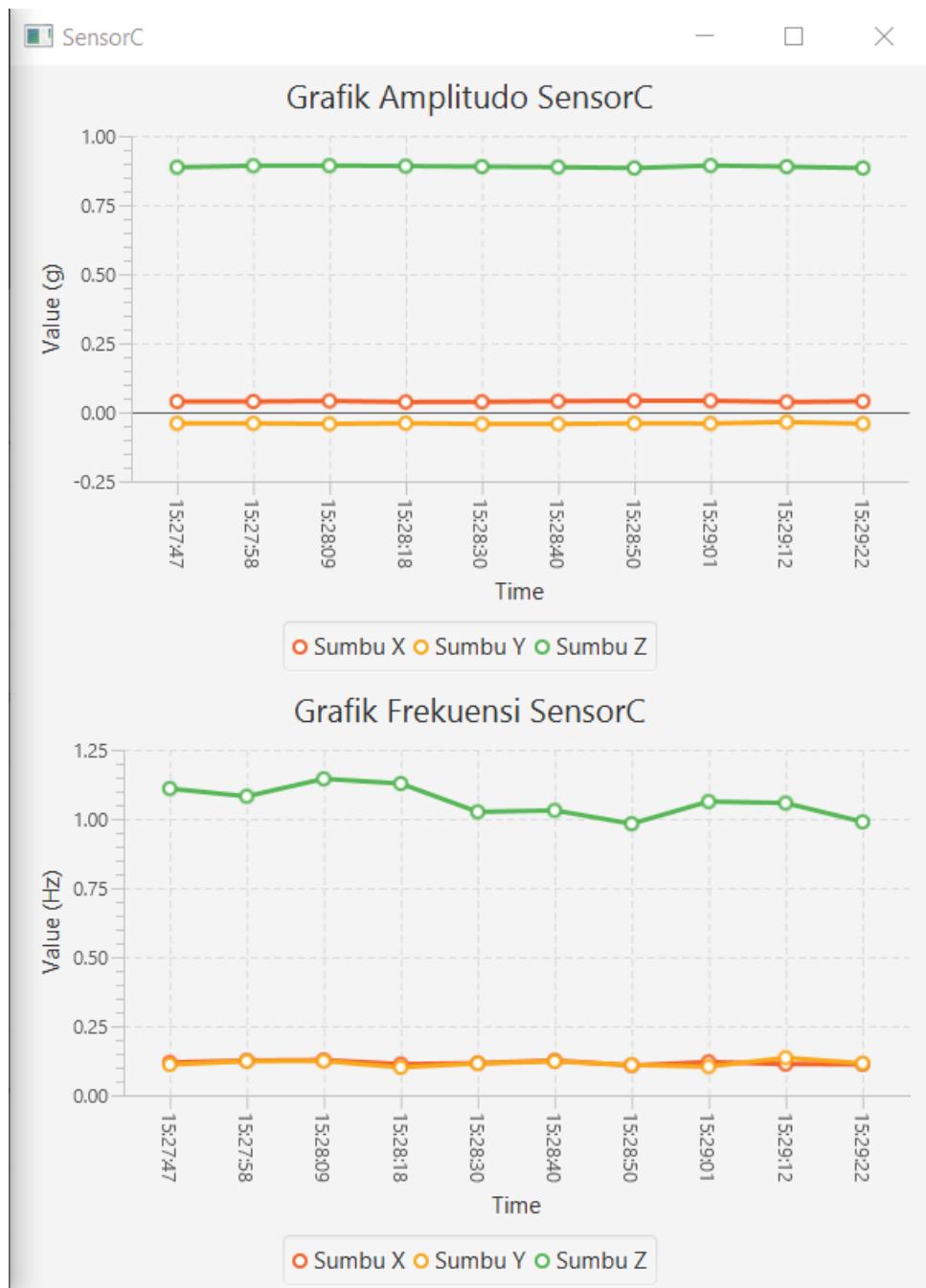
Gambar C.6: Grafik Monitoring getaran SensorA dengan topologi tree

### C.2.2 SensorB



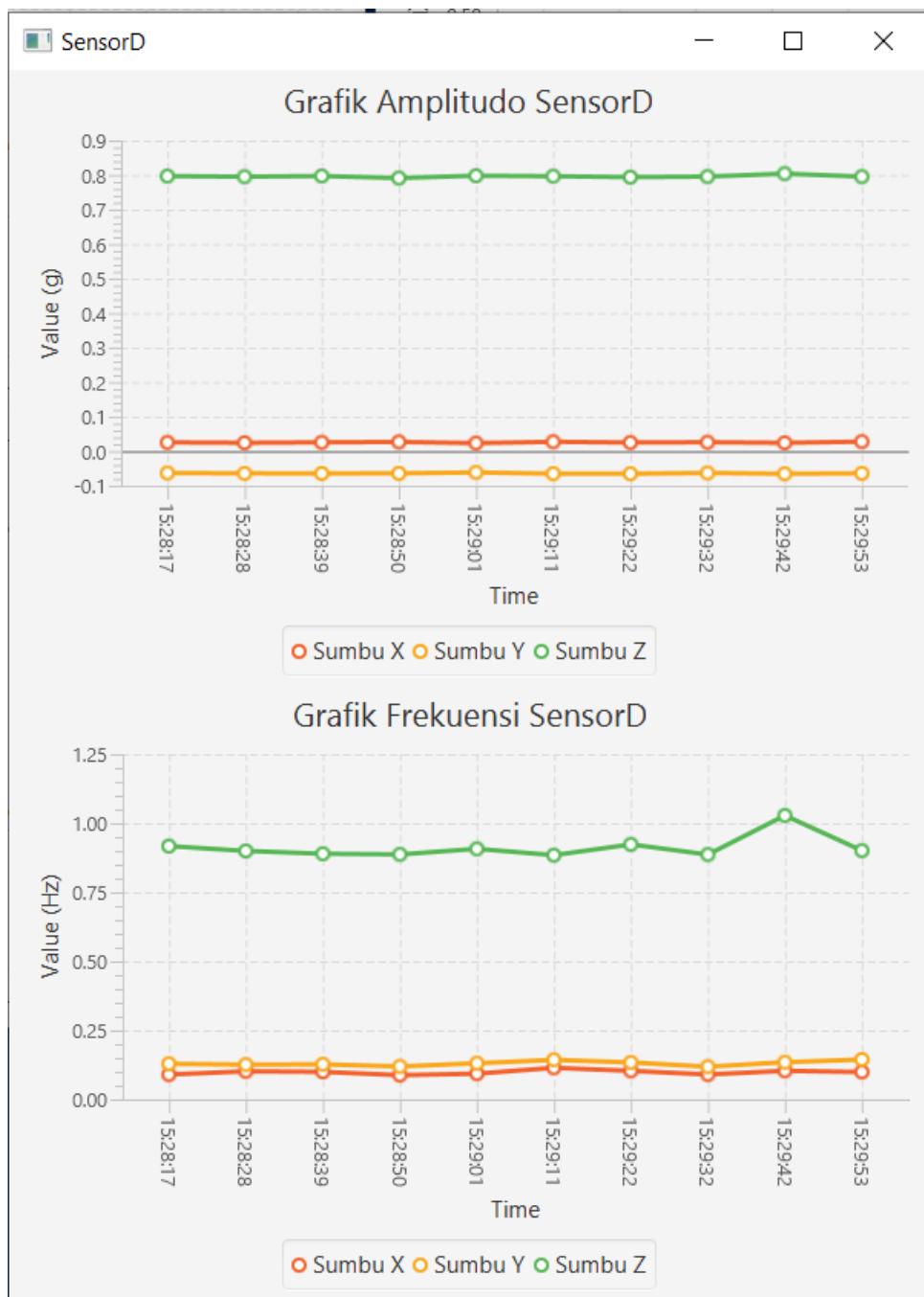
Gambar C.7: Grafik Monitoring getaran SensorB dengan topologi tree

### C.2.3 SensorC



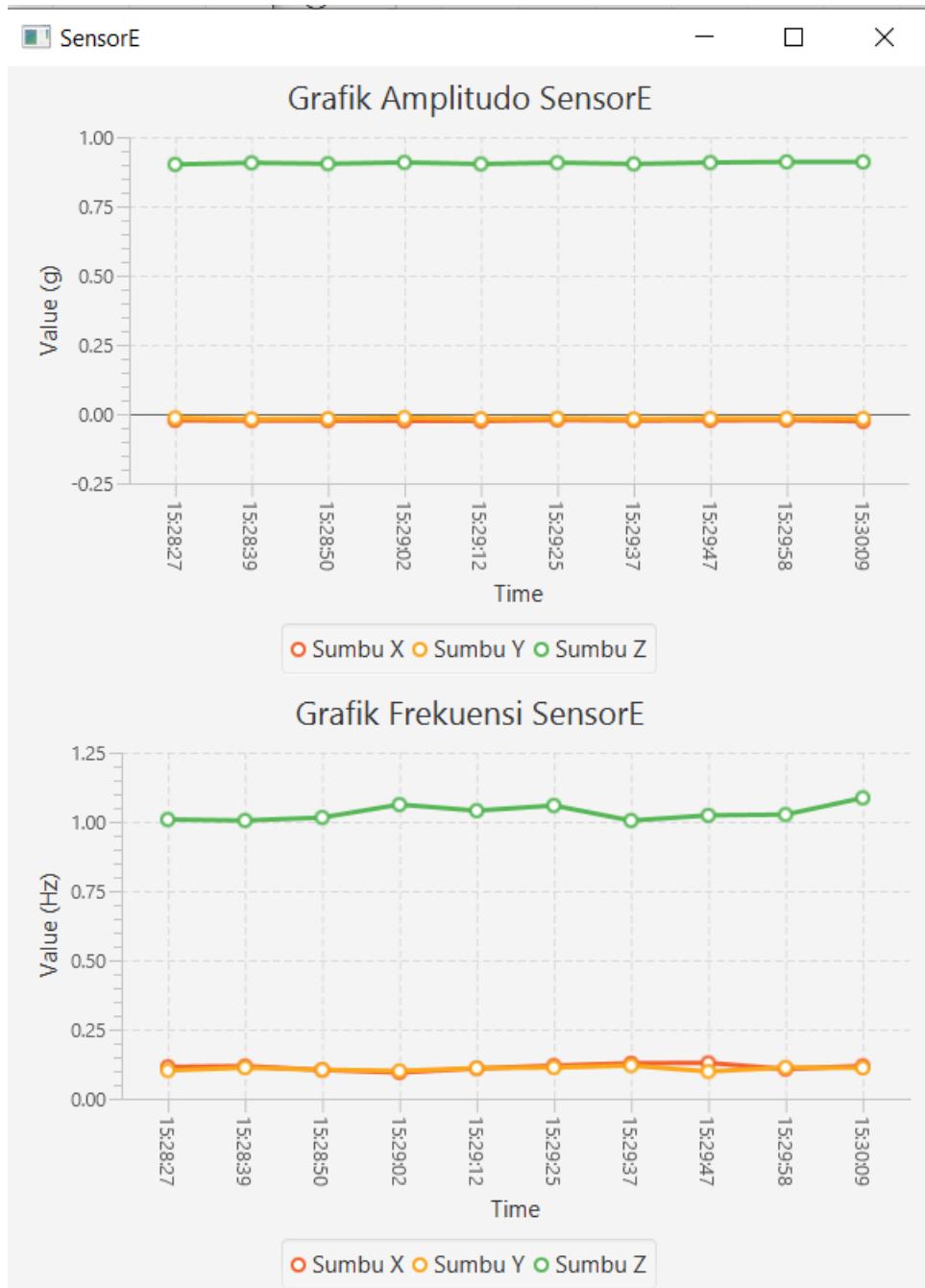
Gambar C.8: Grafik Monitoring getaran SensorC dengan topologi tree

#### C.2.4 SensorD



Gambar C.9: Grafik Monitoring getaran SensorD dengan topologi tree

### C.2.5 SensorE



Gambar C.10: Grafik Monitoring getaran SensorE dengan topologi tree