

SKRIPSI

PENGEMBANGAN APLIKASI EKSTRAKSI FITUR DOMAIN
WAKTU/FREKUENSI UNTUK DATA AKSELEROMETER DI
WSN



Ivan Kristanto

NPM: 2016730082

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2020

UNDERGRADUATE THESIS

**TIME / FREQUENCY DOMAIN FEATURE EXTRACTION
APPLICATION DEVELOPMENT FOR ACCELEROMETER
DATA AT WSN**



Ivan Kristanto

NPM: 2016730082

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2020**

LEMBAR PENGESAHAN

**PENGEMBANGAN APLIKASI EKSTRAKSI FITUR DOMAIN
WAKTU/FREKUENSI UNTUK DATA AKSELEROMETER DI
WSN**

Ivan Kristanto

NPM: 2016730082

Bandung, 9 Juni 2020

Menyetujui,

Pembimbing

Elisati Hulu, M.T.

Ketua Tim Penguji

Anggota Tim Penguji

Mariskha Tri Adithia, P.D.Eng

Luciana Abednego, 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 EKSTRAKSI FITUR DOMAIN WAKTU/FREKUENSI UNTUK DATA AKSELEROMETER DI WSN

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 9 Juni 2020



Ivan Kristanto
NPM: 2016730082

ABSTRAK

Wireless Sensor Network (WSN) adalah jaringan nirkabel berupa node - node sensor yang memiliki kemampuan melakukan komputasi, komunikasi dengan node lain, dan *sensing*. Salah satu sensor yang terdapat pada sensor node adalah sensor akselerometer yang berfungsi sebagai pengukur getaran. Sensor akselerometer mengukur akselerasi pada 3 sumbu (x,y,z). Hasil dari *sensing* akselerometer belum berupa informasi dan dapat diolah dengan melakukan ekstraksi fitur domain waktu ke frekuensi. Dari hasil ekstraksi fitur dari data *sensing* akselerometer dapat digunakan sebagai deteksi getaran dan pengambilan keputusan untuk pemrosesan lanjut.

Ada beberapa teknik ekstraksi fitur domain waktu ke frekuensi. Salah satu teknik ekstraksi fitur yang ada adalah STFT (*Short Time Fourier Transform*). STFT merupakan pengembangan dari teknik ekstraksi fitur FFT (*Fast Fourier Transform*). Ekstraksi fitur dilakukan pada sensor node sehingga hasil *sensing* node merupakan hasil dari STFT.

Pada skripsi ini dibuat aplikasi untuk melakukan ekstraksi fitur untuk data akselerometer. Aplikasi ini mengatur interaksi komputer pengguna, *base station*, dan sensor node. Algoritma ekstraksi fitur yang digunakan adalah STFT. Proses ekstraksi fitur data akselerometer dilakukan di sensor node dan hasil ekstraksi fitur dikirim ke *base station* dan ditampilkan di komputer.

Hasil dari pengujian menunjukkan bahwa aplikasi ekstraksi fitur ini dapat dibangun di WSN. Hasil dari ekstraksi fitur bergantung pada letak sensor melakukan *sensing* dan tidak bergantung dengan topologi WSN yang digunakan.

Kata-kata kunci: sensor jaringan nirkabel, akselerometer, ekstraksi fitur, transformasi fourier

ABSTRACT

Wireless Sensor Network (WSN) is a wireless network in the form of sensor nodes that have the ability to compute, communicate with other nodes, and sensing. One of the sensors contained in the node is the Accelerometer sensor that measures vibration from acceleration. The Accelerometer sensor measures acceleration on 3 axes (x, y, z). The results of the sensing are not in the form of information and can be processed by time domain frequency extracting features. From the results of feature extraction from the accelerometer data, it can be used as vibration detection and decision making for further processing.

There are several time domain frequency feature extraction techniques. One of feature extraction techniques available is STFT (Short Time Fourier Transform). STFT is a development of FFT (Fast Fourier Transform). Feature extraction is done on the sensor node so the results of it sensing node are the result of STFT.

In this thesis is created an application that can extract accelerometer data feature. This application controls user computer, base station, and sensor node interactions. STFT feature extraction algorithm is used for the extraction process. The process is done in sensor node and the result is sent to base station and displayed in user computer.

The results show that this application can be built on WSN. The feature extraction result depend on where the sensor perform sensing and does not depend on the WSN topology used.

Keywords: wireless sensor network, accelerometer, feature extraction, fourier transform

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 Ekstraksi Fitur Domain Waktu/Frekuensi untuk Data Akselerometer di 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. Bapa Elisati Hulu, M.T. selaku dosen pembimbing yang telah membimbing penulis dan memberikan dukungan maupun bantuan kepada penulis dalam proses penyusunan skripsi ini.
4. Ibu Mariskha Tri Adithia, P.D.Eng dan Ibu Luciana Abednego, M.T. 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, Juni 2020

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xxi
DAFTAR TABEL	xxv
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Tujuan	1
1.4 Batasan Masalah	1
1.5 Metodologi	2
1.6 Sistematika Pembahasan	2
2 LANDASAN TEORI	3
2.1 Wireless Sensor Network	3
2.1.1 Aplikasi dari Wireless Sensor Network [1]	3
2.1.2 Sensor Node [2]	4
2.1.3 Topologi pada WSN [3]	5
2.1.4 Arsitektur WSN [4]	9
2.1.5 Sistem Operasi WSN [5]	11
2.1.6 Protokol Stack WSN [4]	12
2.2 Akselerometer [6]	13
2.3 Feature Extraction [7]	13
2.4 Fourier Transform [8]	14
2.4.1 Complex Number [8]	15
2.4.2 Discrete Fourier Transform (DFT) [9] [8]	17
2.4.3 Fast Fourier Transform (FFT) [9] [10]	17
2.4.4 Short Time Fourier Transform (STFT) [11]	19
2.5 PreonVM & Preon32	21
2.5.1 PreonVM	21
2.5.2 Preon32	23
3 ANALISIS	25
3.1 Analisis Aplikasi Ekstraksi Fitur Domain Waktu/Frekuensi Untuk Data Akselerometer di WSN	25
3.1.1 Analisis Topologi dan Arsitektur WSN	25
3.1.2 Analisis Akselerometer	26
3.1.3 Analisis Fungsi Aplikasi	26
3.1.4 Analisis Kelas	30
3.2 Analisis Proses Ekstraksi Fitur	32

3.2.1	Analisis <i>Sample</i> Data Akselerometer	32
3.2.2	Analisis Algoritma <i>Short Time Fourier Transform</i>	32
4	PERANCANGAN	35
4.1	Perancangan Interaksi Antar Node	35
4.1.1	Diagram Sequence "Check Online Node"	35
4.1.2	Diagram Sequence "Sense"	36
4.1.3	Diagram Sequence "Stop Sensing"	37
4.1.4	Diagram Sequence "Exit"	37
4.2	Perancangan Antar Muka Untuk Visualisasi Hasil Ekstraksi	38
4.2.1	Antar Muka Visualisasi Hasil Sense	38
4.2.2	Antar Muka Spectrogram	39
4.3	Perancangan Kelas Aplikasi	39
4.3.1	Package UserApp	39
4.3.2	Package BaseStation	46
4.3.3	Package SensorNode	48
4.4	Perancangan Masukan dan Keluaran	55
4.5	Perancangan Pseudocode Aplikasi	55
4.5.1	Base Station	55
4.5.2	Sensor Node	58
5	IMPLEMENTASI DAN PENGUJIAN	61
5.1	Implementasi	61
5.1.1	Lingkungan Implementasi	61
5.1.2	Hasil Implementasi Kelas	61
5.1.3	Hasil Implementasi Antar Muka Visualisasi Hasil Sense	66
5.1.4	Hasil Implementasi Antar Muka Spectrogram	67
5.2	Pengujian	68
5.2.1	Pengujian Fungsional	68
5.2.2	Pengujian Eksperimental	71
5.2.3	Masalah dalam Pengujian	76
6	KESIMPULAN DAN SARAN	77
6.1	Kesimpulan	77
6.2	Saran	77
DAFTAR REFERENSI		79
A	KODE PROGRAM	81
B	HASIL EKSPERIMEN DI ATAP	97
B.1	Topologi Star	97
B.1.1	Rectangle Window	97
B.1.2	Hanning Window	102
B.1.3	Hamming Window	107
B.2	Topologi Tree	112
B.2.1	Rectangle Window	112
B.2.2	Hanning Window	117
B.2.3	Hamming Window	122
C	HASIL EKSPERIMEN DI JALAN	129
C.1	Topologi Star	129
C.1.1	Rectangle Window	129

C.1.2	Hanning Window	134
C.1.3	Hamming Window	139
C.2	Topologi Tree	144
C.2.1	Rectangle Window	144
C.2.2	Hanning Window	149
C.2.3	Hamming Window	154

DAFTAR GAMBAR

2.1	Contoh aplikasi WSN pada bidang militer	3
2.2	Struktur Sensor Node	4
2.3	<i>Single Hop</i> dan <i>Multi Hop</i>	5
2.4	<i>Bus Topology</i>	6
2.5	<i>Tree Topology</i>	6
2.6	<i>Star Topology</i>	7
2.7	<i>Ring Topology</i>	7
2.8	<i>Mesh Topology</i>	8
2.9	<i>Circular Topology</i>	8
2.10	<i>Grid Topology</i>	9
2.11	<i>Flat Architecture</i>	9
2.12	<i>Single Hop Architecture</i>	10
2.13	<i>Multi Hop Clustering Architecture</i>	10
2.14	<i>Multi Tier Clustering Architecture</i>	11
2.15	Protokol Stack WSN	12
2.16	4 kategori dari <i>Fourier Transform</i> dan contoh bentuk sinyal	15
2.17	Contoh dari <i>complex plane</i> pada notasi <i>rectangular</i>	16
2.18	Contoh dari <i>complex plane</i> pada notasi polar	17
2.19	contoh visual <i>Decimation In Time</i> dengan panjang 8	18
2.20	Jenis gelombang	19
2.21	Contoh Spectrogram	20
2.22	<i>Virtual machine</i> yang memampukan aplikasi berjalan secara independen	21
2.23	Preon32	23
3.1	WSN dengan topologi <i>star</i> dengan komunikasi <i>single-hop</i>	25
3.2	WSN dengan topologi <i>tree</i> dengan komunikasi <i>single-hop</i> dan <i>multi-hop</i>	26
3.3	Diagram <i>use case</i> Aplikasi Ekstraksi Fitur Domain Waktu/Frekuenyi Untuk Data Akselerometer di WSN	27
3.4	Kelas Diagram Sederhana Pada Aplikasi di BaseStation	30
3.5	Kelas Diagram Sederhana Pada Aplikasi di SensorNode	30
3.6	Kelas Diagram Sederhana Pada Aplikasi di UserApp	31
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	Rancangan antar muka tampilan hasil sense	38
4.6	Rancangan antar muka tampilan hasil sense	39
4.7	Kelas diagram lengkap package UserApp	40
4.8	Perancangan Kelas Tester	40
4.9	Perancangan Kelas Plotting	42
4.10	Perancangan Kelas Visualizing	43
4.11	Perancangan Kelas Spectrogram	44

4.12 Kelas diagram lengkap package BaseStation	46
4.13 Perancangan Kelas BSManger	47
4.14 Kelas diagram lengkap package SensorNode	48
4.15 Perancangan Kelas SNManager	49
4.16 Perancangan Kelas Complex	50
4.17 Perancangan Kelas Accelerometer	51
4.18 Perancangan Kelas FFT	52
4.19 Perancangan Kelas ShortTimeFourierTransform	52
4.20 Perancangan Kelas SenseController	54
5.1 Tampilan awal aplikasi	66
5.2 Grafik hasil implementasi visualisasi hasil sense	67
5.3 Grafik hasil implementasi visualisasi hasil sense	67
5.4 Tampilan awal setelah memasukkan jumlah sensor node	68
5.5 Pengujian "Check Online node"	69
5.6 Tampilan awal setelah menjalankan fungsi "Sense"	69
5.7 Tampilan setelah mendapatkan hasil ekstraksi fitur dari sensor node	70
5.8 Pengujian input selain angka "3" "Sense"	70
5.9 Pengujian "Stop sensing"	71
5.10 Pengujian "Exit"	71
5.11 Topologi star yang digunakan	72
5.12 Topologi tree yang digunakan	72
5.13 Denah peletakan sensor node	73
B.1 Hasil Sensing Sensor1 dengan topologi star dan window <i>Rectangle</i> di Atap	97
B.2 Hasil Spectrogram Sensor1 dengan topologi star dan window <i>Rectangle</i> di Atap	98
B.3 Hasil Sensing Sensor2 dengan topologi star dan window <i>Rectangle</i> di Atap	98
B.4 Hasil Spectrogram Sensor2 dengan topologi star dan window <i>Rectangle</i> di Atap	99
B.5 Hasil Sensing Sensor3 dengan topologi star dan window <i>Rectangle</i> di Atap	99
B.6 Hasil Spectrogram Sensor3 dengan topologi star dan window <i>Rectangle</i> di Atap	100
B.7 Hasil Sensing Sensor4 dengan topologi star dan window <i>Rectangle</i> di Atap	100
B.8 Hasil Spectrogram Sensor4 dengan topologi star dan window <i>Rectangle</i> di Atap	101
B.9 Hasil Sensing Sensor5 dengan topologi star dan window <i>Rectangle</i> di Atap	101
B.10 Hasil Spectrogram Sensor5 dengan topologi star dan window <i>Rectangle</i> di Atap	102
B.11 Hasil Sensing Sensor1 dengan topologi star dan window <i>Hanning</i> di Atap	102
B.12 Hasil Spectrogram Sensor1 dengan topologi star dan window <i>Hanning</i> di Atap	103
B.13 Hasil Sensing Sensor2 dengan topologi star dan window <i>Hanning</i> di Atap	103
B.14 Hasil Spectrogram Sensor2 dengan topologi star dan window <i>Hanning</i> di Atap	104
B.15 Hasil Sensing Sensor3 dengan topologi star dan window <i>Hanning</i> di Atap	104
B.16 Hasil Spectrogram Sensor3 dengan topologi star dan window <i>Hanning</i> di Atap	105
B.17 Hasil Sensing Sensor4 dengan topologi star dan window <i>Hanning</i> di Atap	105
B.18 Hasil Spectrogram Sensor4 dengan topologi star dan window <i>Hanning</i> di Atap	106
B.19 Hasil Sensing Sensor5 dengan topologi star dan window <i>Hanning</i> di Atap	106
B.20 Hasil Spectrogram Sensor5 dengan topologi star dan window <i>Hanning</i> di Atap	107
B.21 Hasil Sensing Sensor1 dengan topologi star dan window <i>Hamming</i> di Atap	107
B.22 Hasil Spectrogram Sensor1 dengan topologi star dan window <i>Hamming</i> di Atap	108
B.23 Hasil Sensing Sensor2 dengan topologi star dan window <i>Hamming</i> di Atap	108
B.24 Hasil Spectrogram Sensor2 dengan topologi star dan window <i>Hamming</i> di Atap	109
B.25 Hasil Sensing Sensor3 dengan topologi star dan window <i>Hamming</i> di Atap	109
B.26 Hasil Spectrogram Sensor3 dengan topologi star dan window <i>Hamming</i> di Atap	110
B.27 Hasil Sensing Sensor4 dengan topologi star dan window <i>Hamming</i> di Atap	110
B.28 Hasil Spectrogram Sensor4 dengan topologi star dan window <i>Hamming</i> di Atap	111

B.29 Hasil Sensing Sensor5 dengan topologi star dan window <i>Hamming</i> di Atap	111
B.30 Hasil Spectrogram Sensor5 dengan topologi star dan window <i>Hamming</i> di Atap . . .	112
B.31 Hasil Sensing Sensor1 dengan topologi tree dan window <i>Rectangle</i> di Atap	112
B.32 Hasil Spectrogram Sensor1 dengan topologi tree dan window <i>Rectangle</i> di Atap . .	113
B.33 Hasil Sensing Sensor2 dengan topologi tree dan window <i>Rectangle</i> di Atap	113
B.34 Hasil Spectrogram Sensor2 dengan topologi tree dan window <i>Rectangle</i> di Atap . .	114
B.35 Hasil Sensing Sensor3 dengan topologi tree dan window <i>Rectangle</i> di Atap	114
B.36 Hasil Spectrogram Sensor3 dengan topologi tree dan window <i>Rectangle</i> di Atap . .	115
B.37 Hasil Sensing Sensor4 dengan topologi tree dan window <i>Rectangle</i> di Atap	115
B.38 Hasil Spectrogram Sensor4 dengan topologi tree dan window <i>Rectangle</i> di Atap . .	116
B.39 Hasil Sensing Sensor5 dengan topologi tree dan window <i>Rectangle</i> di Atap	116
B.40 Hasil Spectrogram Sensor5 dengan topologi tree dan window <i>Rectangle</i> di Atap . .	117
B.41 Hasil Sensing Sensor1 dengan topologi tree dan window <i>Hanning</i> di Atap	117
B.42 Hasil Spectrogram Sensor1 dengan topologi tree dan window <i>Hanning</i> di Atap . .	118
B.43 Hasil Sensing Sensor2 dengan topologi tree dan window <i>Hanning</i> di Atap	118
B.44 Hasil Spectrogram Sensor2 dengan topologi tree dan window <i>Hanning</i> di Atap . .	119
B.45 Hasil Sensing Sensor3 dengan topologi tree dan window <i>Hanning</i> di Atap	119
B.46 Hasil Spectrogram Sensor3 dengan topologi tree dan window <i>Hanning</i> di Atap . .	120
B.47 Hasil Sensing Sensor4 dengan topologi tree dan window <i>Hanning</i> di Atap	120
B.48 Hasil Spectrogram Sensor4 dengan topologi tree dan window <i>Hanning</i> di Atap . .	121
B.49 Hasil Sensing Sensor5 dengan topologi tree dan window <i>Hanning</i> di Atap	121
B.50 Hasil Spectrogram Sensor5 dengan topologi tree dan window <i>Hanning</i> di Atap . .	122
B.51 Hasil Sensing Sensor1 dengan topologi tree dan window <i>Hamming</i> di Atap	122
B.52 Hasil Spectrogram Sensor1 dengan topologi tree dan window <i>Hamming</i> di Atap . .	123
B.53 Hasil Sensing Sensor2 dengan topologi tree dan window <i>Hamming</i> di Atap	123
B.54 Hasil Spectrogram Sensor2 dengan topologi tree dan window <i>Hamming</i> di Atap . .	124
B.55 Hasil Sensing Sensor3 dengan topologi tree dan window <i>Hamming</i> di Atap	124
B.56 Hasil Spectrogram Sensor3 dengan topologi tree dan window <i>Hamming</i> di Atap . .	125
B.57 Hasil Sensing Sensor4 dengan topologi tree dan window <i>Hamming</i> di Atap	125
B.58 Hasil Spectrogram Sensor4 dengan topologi tree dan window <i>Hamming</i> di Atap . .	126
B.59 Hasil Sensing Sensor5 dengan topologi tree dan window <i>Hamming</i> di Atap	126
B.60 Hasil Spectrogram Sensor5 dengan topologi tree dan window <i>Hamming</i> di Atap . .	127
C.1 Hasil Sensing Sensor1 dengan topologi star dan window <i>Rectangle</i> di Jalan	129
C.2 Hasil Spectrogram Sensor1 dengan topologi star dan window <i>Rectangle</i> di Jalan . .	130
C.3 Hasil Sensing Sensor2 dengan topologi star dan window <i>Rectangle</i> di Jalan	130
C.4 Hasil Spectrogram Sensor2 dengan topologi star dan window <i>Rectangle</i> di Jalan . .	131
C.5 Hasil Sensing Sensor3 dengan topologi star dan window <i>Rectangle</i> di Jalan	131
C.6 Hasil Spectrogram Sensor3 dengan topologi star dan window <i>Rectangle</i> di Jalan . .	132
C.7 Hasil Sensing Sensor4 dengan topologi star dan window <i>Rectangle</i> di Jalan	132
C.8 Hasil Spectrogram Sensor4 dengan topologi star dan window <i>Rectangle</i> di Jalan . .	133
C.9 Hasil Sensing Sensor5 dengan topologi star dan window <i>Rectangle</i> di Jalan	133
C.10 Hasil Spectrogram Sensor5 dengan topologi star dan window <i>Rectangle</i> di Jalan . .	134
C.11 Hasil Sensing Sensor1 dengan topologi star dan window <i>Hanning</i> di Jalan	134
C.12 Hasil Spectrogram Sensor1 dengan topologi star dan window <i>Hanning</i> di Jalan . .	135
C.13 Hasil Sensing Sensor2 dengan topologi star dan window <i>Hanning</i> di Jalan	135
C.14 Hasil Spectrogram Sensor2 dengan topologi star dan window <i>Hanning</i> di Jalan . .	136
C.15 Hasil Sensing Sensor3 dengan topologi star dan window <i>Hanning</i> di Jalan	136
C.16 Hasil Spectrogram Sensor3 dengan topologi star dan window <i>Hanning</i> di Jalan . .	137
C.17 Hasil Sensing Sensor4 dengan topologi star dan window <i>Hanning</i> di Jalan	137
C.18 Hasil Spectrogram Sensor4 dengan topologi star dan window <i>Hanning</i> di Jalan . .	138
C.19 Hasil Sensing Sensor5 dengan topologi star dan window <i>Hanning</i> di Jalan	138

C.20 Hasil Spectrogram Sensor5 dengan topologi star dan window <i>Hanning</i> di Jalan	139
C.21 Hasil Sensing Sensor1 dengan topologi star dan window <i>Hamming</i> di Jalan	139
C.22 Hasil Spectrogram Sensor1 dengan topologi star dan window <i>Hamming</i> di Jalan	140
C.23 Hasil Sensing Sensor2 dengan topologi star dan window <i>Hamming</i> di Jalan	140
C.24 Hasil Spectrogram Sensor2 dengan topologi star dan window <i>Hamming</i> di Jalan	141
C.25 Hasil Sensing Sensor3 dengan topologi star dan window <i>Hamming</i> di Jalan	141
C.26 Hasil Spectrogram Sensor3 dengan topologi star dan window <i>Hamming</i> di Jalan	142
C.27 Hasil Sensing Sensor4 dengan topologi star dan window <i>Hamming</i> di Jalan	142
C.28 Hasil Spectrogram Sensor4 dengan topologi star dan window <i>Hamming</i> di Jalan	143
C.29 Hasil Sensing Sensor5 dengan topologi star dan window <i>Hamming</i> di Jalan	143
C.30 Hasil Spectrogram Sensor5 dengan topologi star dan window <i>Hamming</i> di Jalan	144
C.31 Hasil Sensing Sensor1 dengan topologi tree dan window <i>Rectangle</i> di Jalan	144
C.32 Hasil Spectrogram Sensor1 dengan topologi tree dan window <i>Rectangle</i> di Jalan	145
C.33 Hasil Sensing Sensor2 dengan topologi tree dan window <i>Rectangle</i> di Jalan	145
C.34 Hasil Spectrogram Sensor2 dengan topologi tree dan window <i>Rectangle</i> di Jalan	146
C.35 Hasil Sensing Sensor3 dengan topologi tree dan window <i>Rectangle</i> di Jalan	146
C.36 Hasil Spectrogram Sensor3 dengan topologi tree dan window <i>Rectangle</i> di Jalan	147
C.37 Hasil Sensing Sensor4 dengan topologi tree dan window <i>Rectangle</i> di Jalan	147
C.38 Hasil Spectrogram Sensor4 dengan topologi tree dan window <i>Rectangle</i> di Jalan	148
C.39 Hasil Sensing Sensor5 dengan topologi tree dan window <i>Rectangle</i> di Jalan	148
C.40 Hasil Spectrogram Sensor5 dengan topologi tree dan window <i>Rectangle</i> di Jalan	149
C.41 Hasil Sensing Sensor1 dengan topologi tree dan window <i>Hanning</i> di Jalan	149
C.42 Hasil Spectrogram Sensor1 dengan topologi tree dan window <i>Hanning</i> di Jalan	150
C.43 Hasil Sensing Sensor2 dengan topologi tree dan window <i>Hanning</i> di Jalan	150
C.44 Hasil Spectrogram Sensor2 dengan topologi tree dan window <i>Hanning</i> di Jalan	151
C.45 Hasil Sensing Sensor3 dengan topologi tree dan window <i>Hanning</i> di Jalan	151
C.46 Hasil Spectrogram Sensor3 dengan topologi tree dan window <i>Hanning</i> di Jalan	152
C.47 Hasil Sensing Sensor4 dengan topologi tree dan window <i>Hanning</i> di Jalan	152
C.48 Hasil Spectrogram Sensor4 dengan topologi tree dan window <i>Hanning</i> di Jalan	153
C.49 Hasil Sensing Sensor5 dengan topologi tree dan window <i>Hanning</i> di Jalan	153
C.50 Hasil Spectrogram Sensor5 dengan topologi tree dan window <i>Hanning</i> di Jalan	154
C.51 Hasil Sensing Sensor1 dengan topologi tree dan window <i>Hamming</i> di Jalan	154
C.52 Hasil Spectrogram Sensor1 dengan topologi tree dan window <i>Hamming</i> di Jalan	155
C.53 Hasil Sensing Sensor2 dengan topologi tree dan window <i>Hamming</i> di Jalan	155
C.54 Hasil Spectrogram Sensor2 dengan topologi tree dan window <i>Hamming</i> di Jalan	156
C.55 Hasil Sensing Sensor3 dengan topologi tree dan window <i>Hamming</i> di Jalan	156
C.56 Hasil Spectrogram Sensor3 dengan topologi tree dan window <i>Hamming</i> di Jalan	157
C.57 Hasil Sensing Sensor4 dengan topologi tree dan window <i>Hamming</i> di Jalan	157
C.58 Hasil Spectrogram Sensor4 dengan topologi tree dan window <i>Hamming</i> di Jalan	158
C.59 Hasil Sensing Sensor5 dengan topologi tree dan window <i>Hamming</i> di Jalan	158
C.60 Hasil Spectrogram Sensor5 dengan topologi tree dan window <i>Hamming</i> di Jalan	159

DAFTAR TABEL

2.1	Tabel contoh <i>bit reverse</i> dari indeks 0 sampai 7	18
2.2	Tabel Package dari Virtenio pada PreonVM	22
2.3	Tabel Package dari Virtenio pada PreonVM	22
2.4	Tabel Package dari Java pada PreonVM	23
3.1	Tabel Skenario Memeriksa sensor node yang aktif / Check Online Node.	28
3.2	Tabel Skenario Memberikan perintah sense ke sensor node	28
3.3	Tabel Skenario Memberikan perintah berhenti sense / Stop Sensing	29
3.4	Tabel Skenario Exit	29
5.1	Tabel hasil getaran yang tertangkap di Atap dengan Topologi Star	73
5.2	Tabel hasil getaran yang tertangkap di Atap dengan Topologi Tree	74
5.3	Tabel hasil getaran yang tertangkap di Jalan Raya dengan Topologi Star	75
5.4	Tabel hasil getaran yang tertangkap di Jalan Raya dengan Topologi Tree	75

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Wireless Sensor Network (WSN) adalah jaringan nirkabel berupa *node - node* dan *base station / sink*. WSN dapat memiliki sensor - sensor pada *node* yang dapat mengukur atau *sensing* suara, getaran, suhu, kelembaban udara, dan lainnya. Setiap hasil pengukuran atau *sense* dari *node* dapat dikirimkan ke *base station / sink* untuk diproses lebih lanjut atau diproses pada *node* tersebut [5].

Salah satu sensor yang terdapat di WSN adalah *accelerometer* yang dapat mendeteksi getaran dan mengukur percepatan pada *node* masing - masing. Hasil *sense* dari *accelerometer* adalah berupa 3 sumbu (x,y,z) yang menunjukkan arah dari sensor *node* tersebut. Data *sense* dari *accelerometer* perlu diolah untuk dapat menjadi informasi. Salah satu informasi yang dapat didapat dari hasil *sense accelerometer* adalah frekuensi getaran.

Salah satu proses untuk mendapatkan frekuensi dari data *sense accelerometer* adalah dengan ekstraksi fitur domain waktu/frekuensi. Proses ekstraksi fitur domain waktu/frekuensi adalah proses pengambilan ciri sebuah objek yang dapat menggambarkan karakteristik dari objek tersebut pada domain waktu/frekuensi. Ada berbagai teknik analisis waktu/frekuensi yang dapat digunakan untuk ekstraksi fitur, seperti *Fast Fourier Transform* (FFT), *Short Time Fourier Transform* (STFT), *S-Transform*, dan *Wavelet Transform* [8]. Proses ekstraksi ini dilakukan di sensor *node*, sehingga setiap hasil *sense* akselerometer akan langsung diekstraksi fitur dan dikirimkan ke *sink*.

Pada skripsi ini, akan dibuat sebuah perangkat lunak pada *node* WSN yang dapat mengekstraksi fitur data *sense* dari sensor *accelerometer*. Hasil dari ekstraksi fitur ini adalah frekuensi dari data *sense accelerometer* menggunakan salah satu teknik analisis waktu/frekuensi, yaitu *Short Time Fourier Transform* (STFT).

1.2 Rumusan Masalah

Berdasarkan latar belakang, berikut rumusan dari masalah - masalah yang ada.

- Bagaimana cara mengekstraksi fitur data akselerometer di sensor node WSN?
- Bagaimana cara kerja algoritma *Short Time Fourier Transform* (STFT) untuk data akselerometer?

1.3 Tujuan

- Menerapkan algoritma *Short Time Fourier Transform* (STFT) untuk data akselerometer.
- Membangun aplikasi ekstraksi fitur untuk data akselerometer di sensor node WSN.

1.4 Batasan Masalah

Penelitian ini memiliki batasan masalah sebagai berikut :

1. Sensor yang digunakan sebagai penelitian hanya sensor akselerometer.
2. Fokus dari penelitian ini adalah membangun aplikasi ekstraksi fitur domain waktu/frekuensi untuk data akselerometer di sensor node WSN oleh karena itu tidak memperhitungkan ekstraksi fitur pada data sensor lain selain data sensor akselerometer.

1.5 Metodologi

Berikut adalah metode penelitian yang digunakan dalam penelitian ini:

- Melakukan studi literatur mengenai WSN (*Wireless Sensor Network* dan sensor *accelerometer*).
- Melakukan studi literatur mengenai proses ekstraksi dan teknik *Short Time Fourier Transform*.
- Mempelajari pemrograman di sensor *node* WSN menggunakan bahasa pemrograman Java.
- Melakukan analisis terhadap aplikasi ekstraksi fitur domain waktu/frekuensi untuk data *accelerometer* di WSN.
- Melakukan analisis proses ekstraksi fitur domain waktu/frekuensi.
- Merancang algoritma untuk proses ekstraksi dengan STFT.
- Mengimplementasikan keseluruhan algoritma yang dirancang ke *node* pada WSN.
- Melakukan pengujian (dan eksperimen) pada perangkat lunak.
- Menulis dokumen skripsi.

1.6 Sistematika Pembahasan

Berikut sistematika pada setiap bab di penelitian ini:

Bab 1 Pendahuluan, yaitu mengenai latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.

Bab 2 Landasan Teori, yaitu membahas tentang teori - teori untuk penelitian ini yaitu *Wireless Sensor Network*, *Accelerometer*, Ekstraksi Fitur, dan *Fourier Transform*.

Bab 3 Analisis, yaitu membahas mengenai analisis aplikasi ekstraksi fitur domain waktu/frekuensi untuk data akselerometer di sensor node WSN dan analisis proses ekstraksi fitur.

Bab 4 Perancangan, yaitu membahas perancangan interaksi antar node, perancangan antar muka untuk hasil *sensing* dan spectrogram, perancangan kelas aplikasi, perancangan masukan dan keluaran, dan perancangan pseudocode aplikasi.

Bab 5 Implementasi dan pengujian, yaitu membahas mengenai implementasi dan pengujian aplikasi ekstraksi fitur domain waktu/frekuensi untuk data akselerometer di sensor node WSN.

Bab 6 Kesimpulan, yaitu membahas mengenai kesimpulan dari hasil pengujian dan saran untuk penelitian ini.

BAB 2

LANDASAN TEORI

Bab ini menjelaskan dasar teori mengenai *Wireless Sensor Network*, *Accelerometer*, *Feature Extraction*, *Fourier Transform*, dan *Preon32*

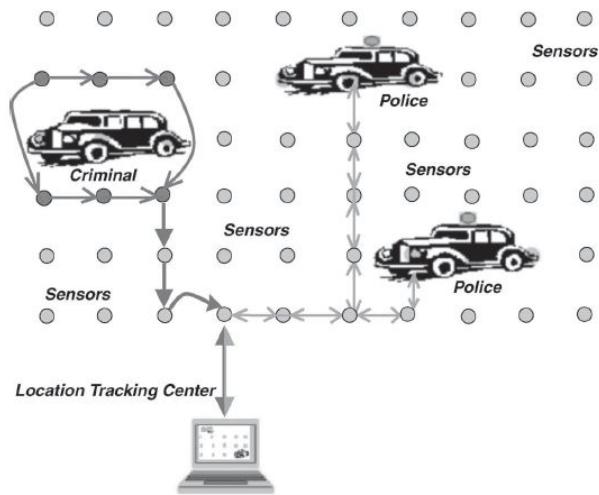
2.1 Wireless Sensor Network

Wireless Sensor Network (WSN) adalah jaringan nirkabel berupa kumpulan dari *node sensor* yang dapat saling berkomunikasi, melakukan komputasi, dan melakukan pengukuran pada lingkungan *node* tersebut diletakkan. [5] Pengukuran yang dapat dilakukan seperti getaran, suhu, kelembapan udara, suara, dan lainnya. Setiap data hasil pengukuran atau *sensing* pada *node* dikirimkan ke *base station* untuk diproses.

2.1.1 Aplikasi dari Wireless Sensor Network [1]

WSN secara umum digunakan pada *high-end application* dan daerah - daerah yang tidak dapat dicakup oleh *wiredline system* seperti sistem pendekripsi radiasi dan *nuclear-threat*. Semakin lama WSN terus dikembangkan dan juga diaplikasikan pada bidang sebagai berikut:

- **Aplikasi bidang Militer** : pemantauan serangan musuh, mendekripsi serangan nuklir/biologis/kimia, penargetan , dan lainnya (Gambar 2.1).



Gambar 2.1: Contoh aplikasi WSN pada bidang militer

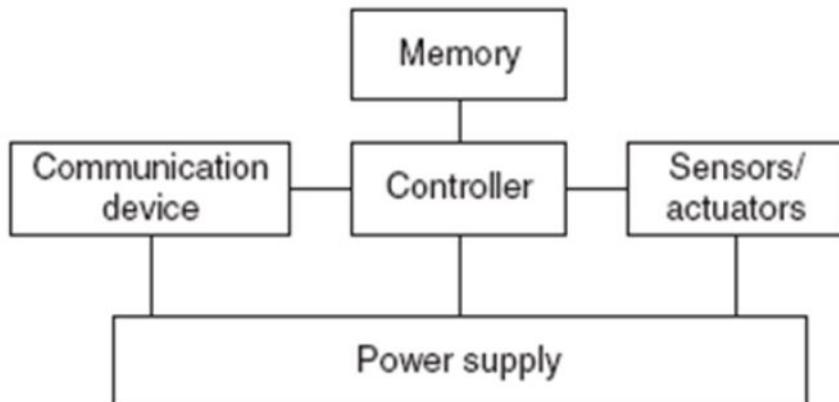
- **Aplikasi bidang Lingkungan/area** : pendekripsi kebakaran hutan, pendekripsi kebanjiran, *microclimate*(kondisi iklim daerah yang kecil), dan lainnya.

- **Aplikasi bidang Kesehatan** : pelacakkan dan pemantauan pasien dan doktor dalam rumah sakit, pemantauan pemberian obat, membantu pasien disabilitas, dan lainnya.
- **Aplikasi bidang Industri** : pendekripsi asap, pemantauan peralatan, pemantauan keamanan, automasi, dan lainnya.

Secara mendasar WSN dapat diaplikasikan saat dibutuhkan alat pemantauan yang dapat beraksisi dengan lingkungan tertentu.

2.1.2 Sensor Node [2]

Struktur pada sensor node memiliki 5 komponen utama, yaitu *power supply*, *controller*, *memory*, *sensor / actuator*, dan *communication device* (Gambar 2.2).



Gambar 2.2: Struktur Sensor Node

Power Supply

Power supply berfungsi sebagai sumber energi untuk sensor node. Secara umum *power supply* yang digunakan oleh sensor node berupa baterai.

Controller

Controller merupakan inti dari sensor node. *Controller* berfungsi untuk mengumpulkan data dari sensor, menerima data dari sensor node lain, memroses data, menentukan tujuan data dikirim, dan menentukan waktu data dikirim. Di dalam *controller* terdapat *microcontroller / microprocessor* yang berfungsi untuk mengatur data dan melakukan komputasi. Beberapa contoh dari *microcontroller* sebagai berikut :

1. Intel StrongARM
2. Texas Instruments MSP 430
3. Atmel ATmega

Memory

Memory pada sensor node adalah berupa *Random Access Memory* (RAM). *Memory* bersifat *volatile*, yaitu jika sensor node mati maka tidak ada data yang disimpan pada sensor node dan setiap data yang disimpan pada *memory* akan hilang.

Sensor / Actuator

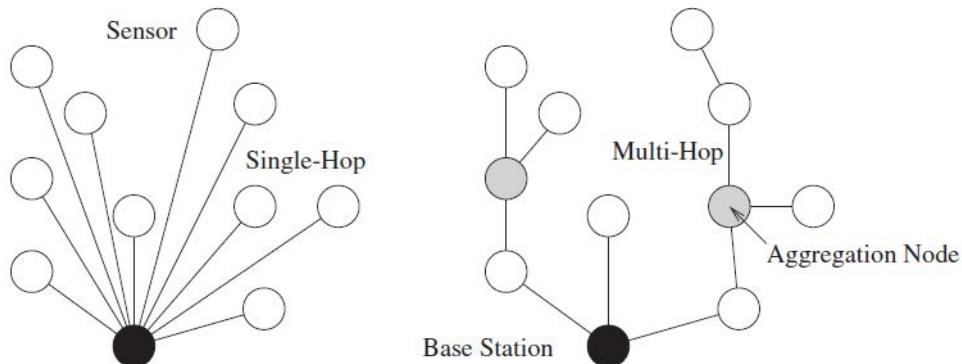
Sensor pada sensor node berfungsi sebagai antarmuka langsung terhadap lingkungan dan *Actuator* berfungsi sebagai pengubah sinyal dari lingkungan menjadi aksi fisik. Contoh dari *actuator* adalah LED yang dapat mengubah listrik menjadi cahaya. Ada beberapa tipe dari sensor:

- *Passive, omnidirectional sensors* : Sensor ini dapat mengukur lingkungan tanpa harus memanipulasi lingkungan dan tidak terpengaruh dari arah sensor tersebut diletakkan. Beberapa contohnya adalah *thermometer*, *light sensor*, *vibration / accelerometer*, *microphones*, *humidity*, *smoke detectors*, *air pressure*, dan lainnya.
- *Passive, narrow-beam sensors* : Sensor ini dapat mengukur lingkungan tanpa harus memanipulasi lingkungan, tetapi berpengaruh terhadap arah sensor tersebut diletakkan. Salah satu contohnya adalah kamera yang dapat langsung mengukur jarak.
- *Active sensors* : Sensor ini mengukur lingkungan dengan terus menerus memeriksa lingkungan. Sebagai contoh *radar*, *sonar*, atau *seismic sensor* yang memancarkan gelombang - gelombang untuk melakukan pengukuran.

Communication Device

Communication device berfungsi sebagai alat untuk mengirim dan menerima informasi dari sensor node lain. Ada beberapa tipe / jalur dari komunikasi pada sensor node (Gambar 2.3) , yaitu:

- *Single Hop* : Komunikasi ini adalah komunikasi langsung antara sensor node dengan *base station* / *sink*, sehingga seperti satu loncatan / *hop*.
- *Multi Hop* : Komunikasi ini memerlukan sensor node lain sebagai loncatan / *hop* untuk berkomunikasi dengan *base station* / *sink*. Jika sensor node ingin mengirimkan informasi ke *base station* maka sensor node mengirimkan informasi tersebut ke sensor node terdekat dan terus sampai ke *base station*.



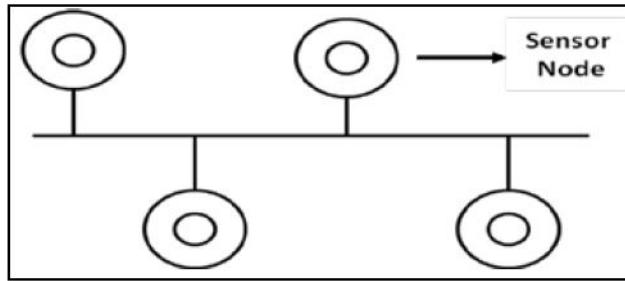
Gambar 2.3: *Single Hop* dan *Multi Hop*

2.1.3 Topologi pada WSN [3]

WSN umumnya memiliki banyak sensor node yang diletakkan pada suatu tempat. Pada WSN bisa terdapat satu atau lebih *sink* atau *base station*. *sink* adalah sensor node yang bertugas untuk mendapatkan data dari sensor node lain. Jalur dan cara komunikasi sensor node dengan sensor node lain / *sink* bergantung pada topologi WSN yang digunakan.

Bus Topology

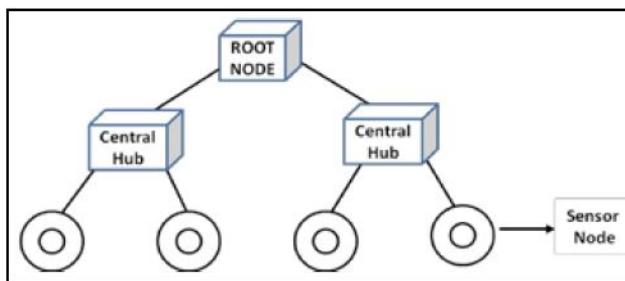
Pada *Bus Topology* (Gambar 2.4), semua node terhubung pada satu jalur untuk saling berkomunikasi. Jika sensor node mengirim pesan maka sensor node akan mengirim pesan secara bergantian dan mengirim *broadcast message* ke semua sensor node, tetapi hanya sensor node yang dituju akan menerima dan memroses message tersebut. Kelebihan dari topologi bus ini adalah mudah untuk diimplementasi, tetapi kelemahannya adalah satu jalur tunggal untuk berkomunikasi yang jika jalur tersebut bermasalah maka sensor node tidak dapat berkomunikasi.



Gambar 2.4: *Bus Topology*

Tree Topology

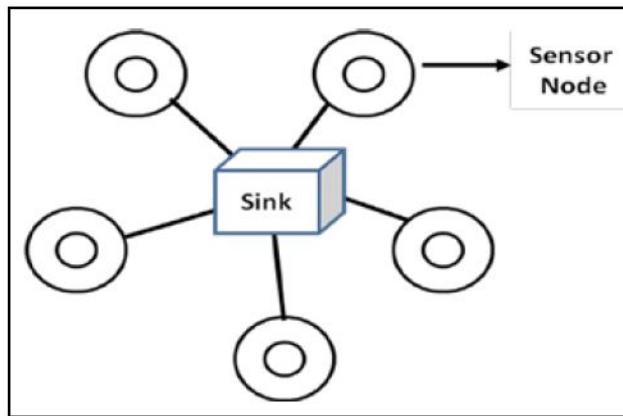
Pada *Tree Topology* (Gambar 2.5), sensor node disusun secara hierarki dan terdapat sensor node yang disebut *root node* pada level teratas. *Root node* bertugas sebagai *communication router* utama / sebagai *base station*. Selain *root node* terdapat sensor node yang disebut *children node* dan *parent node*. *Children* dari sensor node adalah sensor node yang memiliki level lebih rendah dari sensor node tersebut dan *parent* dari sensor node adalah sensor node yang memiliki level lebih tinggi dari sensor node tersebut. Saat sensor node mengirim pesan, sensor node akan meneruskan pesan dari *children* dari sensor node tersebut ke *parent* dari sensor node tersebut hingga sampai ke *base station*. Kelebihan dari topologi ini adalah jika terjadi kesalahan pada sensor node maka akan mudah diidentifikasi dengan cara menelusuri *children* dari sensor node tersebut, tetapi kelemahan dari topologi ini adalah sulit untuk dikonfigurasi dan jika ada jalur yang putus di salah satu sensor node maka akan memutuskan semua komunikasi bagi *children* dari sensor node tersebut.



Gambar 2.5: *Tree Topology*

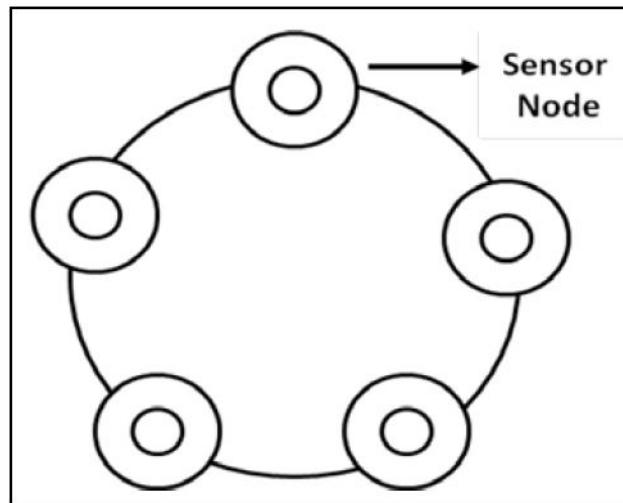
Star Topology

Pada *Star Topology* (Gambar 2.6), semua node terhubung dengan *centralized communication hub* (*sink*) dan sensor node tidak dapat secara langsung berkomunikasi dengan sensor node lainnya. Jika sensor node ingin mengirim pesan ke sensor node lain, maka sensor node harus mengirim pesan ke *sink* lalu *sink* meneruskan pesan tersebut ke sensor node yang dituju. Jika *sink* terjadi kerusakan, maka sensor node tidak dapat berkomunikasi dan jaringan tersebut mati.

Gambar 2.6: *Star Topology*

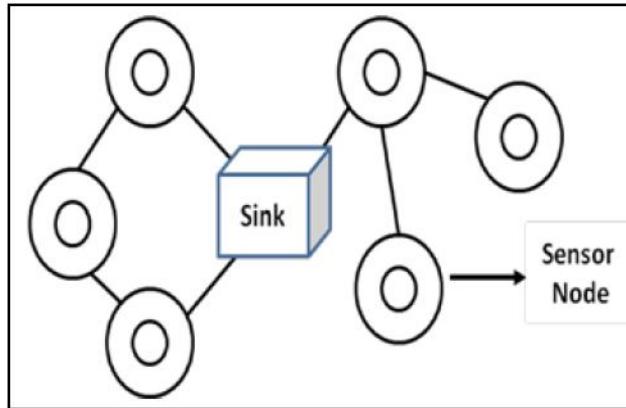
Ring Topology

Pada *Ring Topology* (Gambar 2.7), semua node pasti memiliki dua node tetangga dan dihubungkan secara melingkar. Jika sensor node mengirim pesan, maka pesan tersebut akan diteruskan ke sensor node lain secara melingkar hingga sampai ke sensor node yang dituju. Kelebihan dari sensor node ini adalah mudah untuk diimplementasikan, tetapi jika ada sensor node yang rusak dan membuat kegagalan dalam mengirim pesan, maka sensor node akan mengirimkan kembali pesan ke arah melingkar yang sebaliknya.

Gambar 2.7: *Ring Topology*

Mesh Topology

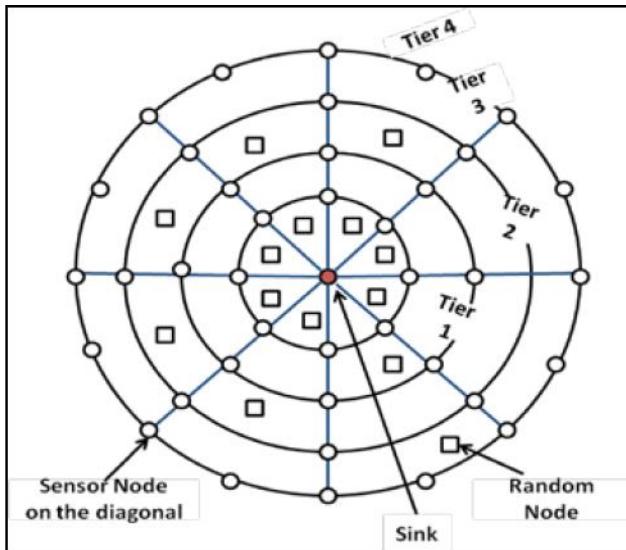
Terdapat 2 tipe dari topologi mesh yaitu, *full mesh* dan *partial mesh* (Gambar 2.8). Jika semua sensor node saling terhubung satu sama lain, maka itu adalah *full mesh*. Jika ada sensor node terhubung secara tidak langsung dengan sensor node lain, maka itu adalah *partial mesh*.



Gambar 2.8: *Mesh Topology*

Circulation Topology

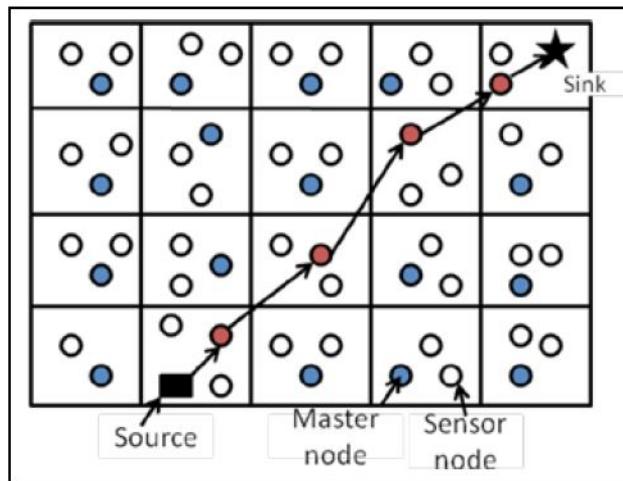
(Gambar 2.9) Topologi ini disebut juga dengan topologi *circular web* karena sensor node diletakkan seperti membentuk jaring yang melingkar dan *sink* sebagai pusat dari jaring. Kelebihan dari topologi ini adalah mudah untuk dibangun, dirawat, dan lebih efisien secara energi [3].



Gambar 2.9: *Circular Topology*

Grid Topology

Pada *Grid Topology* (Gambar 2.10), area dari jaringan dipartisi secara merata menjadi *grid - grid* dan seluruh sensor node dibagi - bagi sesuai *grid*. Pada setiap *grid*, sensor node harus aktif secara bergantian sehingga terdapat satu sensor node yang aktif dalam satu *grid* walaupun dalam *grid* tersebut terdapat lebih dari satu sensor node. Sensor node yang aktif bertugas mengirim atau meneruskan informasi ke sensor node lain sesuai dengan hasil rute dari *Grid-based multi-path routing protocol* hingga sampai ke *sink*.

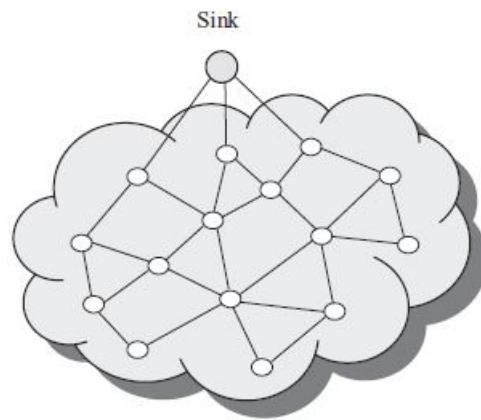
Gambar 2.10: *Grid Topology*

2.1.4 Arsitektur WSN [4]

WSN dapat dibuat dengan arsitektur *flat* atau *hierarchy*. Arsitektur pada WSN mempengaruhi tugas - tugas pada setiap sensor node pada WSN.

Flat Architecture

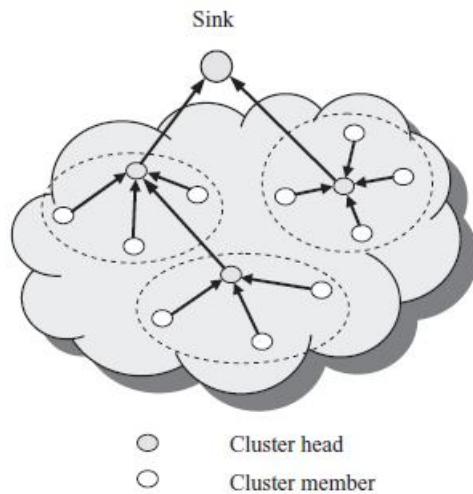
Seluruh sensor node pada arsitektur ini (Gambar 2.11) memiliki tugas yang sama dalam *sensing* dan setiap sensor node merupakan *peers*. Jika *sink* ingin mengumpulkan data maka *sink* akan mengirim *query* dengan teknik *flooding* ke seluruh sensor node dan hanya sensor node yang cocok dengan *query* tersebut akan merespon kembali ke *sink*. Seluruh sensor node melakukan komunikasi *multi hop* dengan menggunakan sensor node lain sebagai penyampai data untuk berkomunikasi dengan *sink*.

Gambar 2.11: *Flat Architecture*

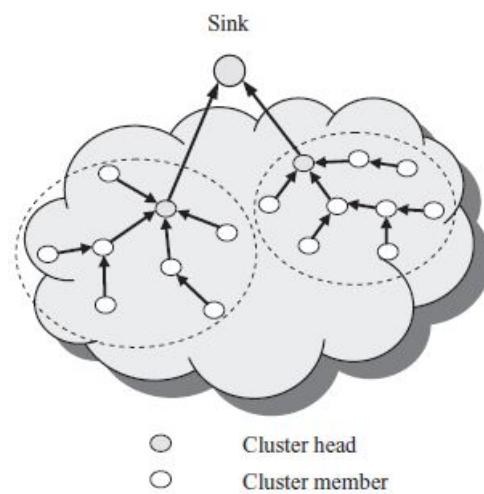
Hierarchy Architecture

Arsitektur ini mengelompokkan seluruh sensor node menjadi *cluster - cluster*. Dalam *cluster* terdapat *cluster head* dan *cluster member*. *Cluster head* bertugas untuk menerima data dari *cluster member* dan mengirimkan data tersebut ke *sink*. *Cluster member* bertugas untuk mengirimkan data hasil pengukuran ke *cluster head*. Dalam *cluster*, sensor node yang memiliki energi yang terbesar dapat dipilih menjadi *cluster head* dan yang lainnya akan menjadi *cluster member*. Selain itu *cluster head*

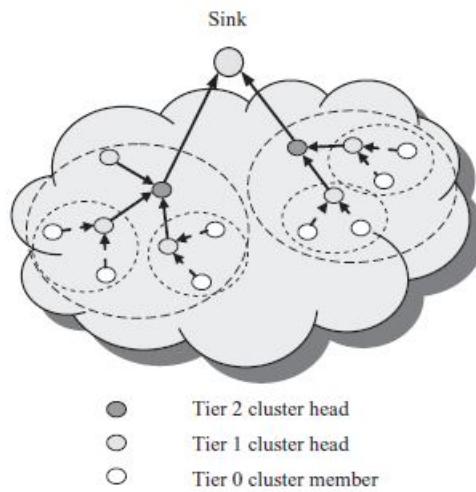
dapat dipilih berdasarkan jarak *cluster head* ke *cluster member*, *single hop* (Gambar 2.12)/*multi hop* (Gambar 2.13), dan berdasarkan jumlah *tier* (Gambar 2.14).



Gambar 2.12: *Single Hop Architecture*



Gambar 2.13: *Multi Hop Clustering Architecture*



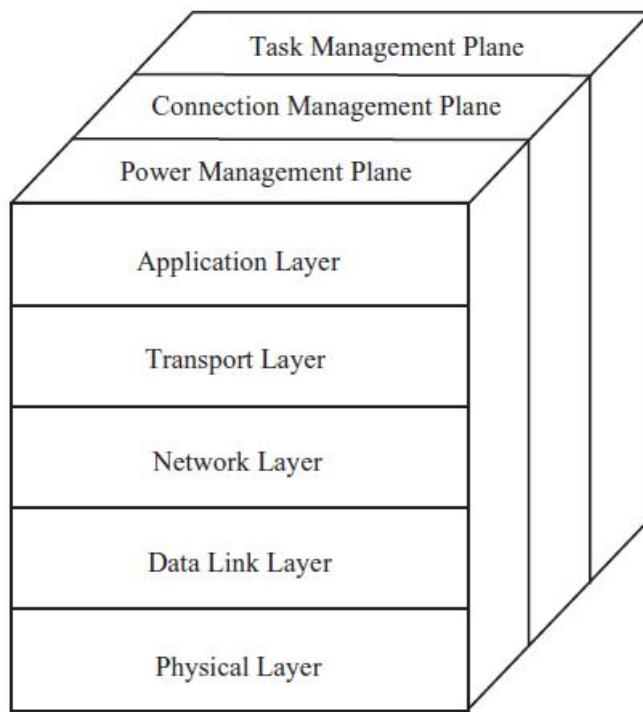
Gambar 2.14: *Multi Tier Clustering Architecture*

2.1.5 Sistem Operasi WSN [5]

Setiap sensor node pada WSN membutuhkan sistem operasi untuk mengatur berjalannya sensor node. Sistem operasi bertugas untuk aplikasi bisa berkomunikasi dengan *hardware*, *scheduling tasks* dan *prioritize tasks*, *memory management*, *power management*, *file management*, dan lainnya. Berikut contoh sistem operasi dari WSN:

- TinyOS
- Contiki
- MANTIS
- Nano-RK
- LiteOS
- PreonVM

2.1.6 Protokol Stack WSN [4]



Gambar 2.15: Protokol Stack WSN

Protocol Stack pada WSN terdiri dari tiga management (Gambar 2.15), yaitu

- *Power Management Plane* : *Power management plane* berkewajiban untuk mengatur power level dari sensor node untuk sensing, processing, dan transmission dan reception. Sebagai contoh sensor node dapat mematikan transceiver saat tidak ada data yang akan dikirim atau diterima.
- *Connection Management Plane* : *Connection management plane* berkewajiban untuk *configuration / reconfiguration* koneksi sensor node saat adanya perubahan topologi pada wsn seperti penambahan atau pengurangan sensor node, perubahan letak node, dan lainnya.
- *Task Management Plane* : *Task management plane* berkewajiban dalam mendistribusikan tugas (task) pada sensor node agar meningkatkan efisiensi energi pada WSN.

Ketiga manajemen tersebut terdapat pada 5 layer protokol stack ,yaitu

1. *Application Layer* : Pada *Application Layer* terdapat bermacam - macam *application-layer protocol* untuk *sensor network application* seperti *query dissemination*, *node localization*, *time synchronization*, dan *network security*. Sebagai contoh, *Sensor Management Protocol* (SMP) adalah *application-layer management protocol* yang menyediakan operasi perangkat lunak untuk berbagai *task*, seperti *exchanging location - related data*, *synchronization sensor nodes*, *moving sensor nodes*, *scheduling sensor nodes*, dan *querying the status of sensor nodes*.
2. *Transport Layer* : *Transport Layer* berkewajiban dalam *reliable end - to - end data delivery* untuk sensor ke sensor atau sensor ke *sink*. Terdapat 2 cara pengiriman yaitu *upstream* dan *downstream*. *Upstream* berarti sensor node mengirimkan data ke *sink*. *Downstream* berarti *sink* mengirimkan data ke sensor node. Kebutuhan *reliability* dari *upstream* dan *downstream*

berbeda. *Downstream* membutuhkan *reliable delivery* karena *sink* hanya mengirimkan data ke sensor node satu kali. *Upstream* tidak terlalu membutuhkan *reliable delivery* karena sensor node mengirimkan data hasil *sense* ke *sink* secara terus menerus sehingga dapat terjadi pengulangan.

3. *Network Layer* : *Network Layer* bertugas dalam membuat rute komunikasi dari sensor node ke *sink*. *Network Layer* menentukan *single hop / multi hop* komunikasi untuk sensor node agar optimal.
4. *Data Link Layer* : *Data Link Layer* berkewajiban dalam *data stream multiplexing, data frame creation and detection, medium access, dan error control* untuk *reliable point - to - point* dan *point - to - multipoint transmissions*. Salah satu fungsi terpenting dari data link layer adalah *medium access control (MAC)*. Objektif utama dari MAC adalah agar sensor node - sensor node dapat secara adil dan efisien berkomunikasi untuk perfomansi dalam konsumsi energi, *network throughput*, dan *delivery latency*.
5. *Physical Layer* *Physical Layer* berkewajiban dalam mengonversi *bit stream* dari *data link layer* menjadi sinyal yang cocok untuk ditransimisi pada media komunikasi.

2.2 Akselerometer [6]

Salah satu sensor yang ada pada *node* sensor adalah sensor akselerometer. Secara umum akselerometer adalah alat yang mengukur *linear (not angular) acceleration*. *Accelerometer* memiliki kemampuan untuk mengukur akselerasi, kemiringan, dan getaran statis (gravitasi bumi) atau dinamis pada 1 sumbu *x/y/z (single axis)* atau 2 sumbu *(x,y)/(x,z)/(y,z) (two axis)* atau 3 sumbu *(x,y,z) (three axis)*.

Akselerometer dapat dibagi menjadi 2 jenis yaitu :

- *Absolute accelerometer* : Akselerometer ini terpasang langsung pada objek yang diukur.
- *Relative accelerometer* : Akselerometer ini mengukur jarak antara objek yang diukur dan *reference point* yang stabil atau bergerak secara konstan. Akselerasi baru didapatkan dengan melakukan diferensiasi ganda pada jarak. Akselerometer ini sering digunakan untuk mengukur getaran dari jarak tertentu (contoh: *laser vibrometer*).

Acceleration sensor dapat diklasifikasi berdasarkan *physical principle* yang digunakan yaitu:

- *Direct Measurement of Force* : pengukuran secara langsung pada gaya, sebagai contoh *piezoelectric sensor*.
- *Indirect Measurement* : pengukuran berdasarkan perpindahan atau deformasi dari *sensing element*.

2.3 Feature Extraction [7]

Ekstraksi fitur dibagi menjadi 2 proses, yaitu

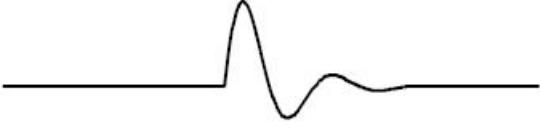
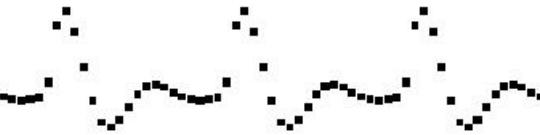
- *Feature Construction* : *Feature* sama dengan *input variable / atribut*. Proses *feature construction* disebut juga dengan *preprocessing*. Ada berbagai macam *preprocessing*, yaitu :
 - *Standardization* : menyamakan satuan input, sebagai contoh jika panjang diukur dalam meter dan lebar diukur dalam centimeter maka perlu disamakan satuannya antara meter atau centimeter.
 - *Normalization* : data dari input dinormalisasi dengan teknik normalisasi tertentu bergantung dengan tipe datanya.

- *Signal enhancement* : input berupa sinyal dapat ditingkatkan dengan melakukan *signal filter*.
 - *Extraction of local features* : melakukan *encode* pada pengetahuan yang spesifik pada suatu masalah menjadi *feature*.
 - *Linear and non-linear space embedding methods* : Jika dimensi dari data sangat besar, maka data tersebut butuh diperkecil menjadi dimensi yang lebih kecil dan mempertahankan informasi yang terkandung pada data tersebut.
 - *Non-linear expansions* : melakukan pembesaran dimensi dari data.
 - *Feature discretization* : beberapa algoritma tidak dapat menangani data yang kontinu sehingga data perlu didiskritkan.
- *Feature Selection* : proses untuk memilih fitur yang paling informatif dan relevan. Tujuan dari proses ini adalah :
 - *general data reduction* : membatasi penyimpanan dan mempercepat algoritma.
 - *feature set reduction* : menyimpan *resources* pada saat pemanfaatan.
 - *performance improvement* : meningkatkan perfomansi dan mendapatkan akurasi prediksi yang lebih baik.
 - *data understanding* : mendapatkan pengetahuan dalam proses yang membuat data tersebut atau dengan mevisualisasi data tersebut.

2.4 Fourier Transform [8]

Fourier Transform merupakan teknik matematika yang secara mendasar menguraikan sinyal menjadi *sinusoids*. Sinyal yang diuraikan dapat berupa sinyal *continuous* atau sinyal *discrete* dan *periodic* atau *aperiodic*. Sinyal *periodic* berarti sinyal tersebut akan memiliki pola yang berulang dan sinyal *aperiodic* berarti sinyal tersebut tidak memiliki pola yang berulang. *Fourier Transform* dibagi menjadi 4 kategori (Gambar 2.16) berdasarkan jenis dari sinyal yaitu:

- *Aperiodic-Continuous* : *Fourier Transform* dengan tipe sinyal ini disebut *Fourier Transform*.
- *Periodic-Continuous* : *Fourier Transform* dengan tipe sinyal ini disebut *Fourier Series*.
- *Aperiodic-Discrete* : *Fourier Transform* dengan tipe sinyal ini disebut *Discrete Time Fourier Transform*.
- *Periodic-Discrete* : *Fourier Transform* dengan tipe sinyal ini disebut *Discrete Fourier Transform / Discrete Fourier Series*.

Type of Transform	Example Signal
Fourier Transform <i>signals that are continuous and aperiodic</i>	
Fourier Series <i>signals that are continuous and periodic</i>	
Discrete Time Fourier Transform <i>signals that are discrete and aperiodic</i>	
Discrete Fourier Transform <i>signals that are discrete and periodic</i>	

Gambar 2.16: 4 kategori dari *Fourier Transform* dan contoh bentuk sinyal

Keempat kategori *fourier transform* memiliki dua versi yaitu *real version* dan *complex version*. *Real version* menggunakan bilangan *real* dan *complex version* menggunakan bilangan kompleks (*complex number*). Pada perkembangannya penggunaan bilangan kompleks lebih banyak digunakan karena bilangan kompleks mengurangi persamaan yang digunakan. Sebagai contoh algoritma *Fast Fourier Transform* berdasarkan bilangan kompleks dalam perhitungannya.

2.4.1 Complex Number [8]

Complex number adalah penjumlahan dari dua komponen yaitu komponen *real* (variabel a pada persamaan 2.1) dan komponen *imaginary* (variabel b pada persamaan 2.1). Komponen *real* adalah *real number*. Komponen *imaginary* adalah bilangan imajiner (persamaan 2.2).

$$a + bj \quad (2.1)$$

$$j = \sqrt{-1} \quad (2.2)$$

Operasi tambah, kurang, kali, dan bagi untuk *complex number* memiliki aturan sebagai berikut:

- operasi tambah

$$(a + bj) + (c + dj) = (a + c) + j(b + d) \quad (2.3)$$

- operasi kurang

$$(a + bj) - (c + dj) = (a - c) + j(b - d) \quad (2.4)$$

- operasi kali

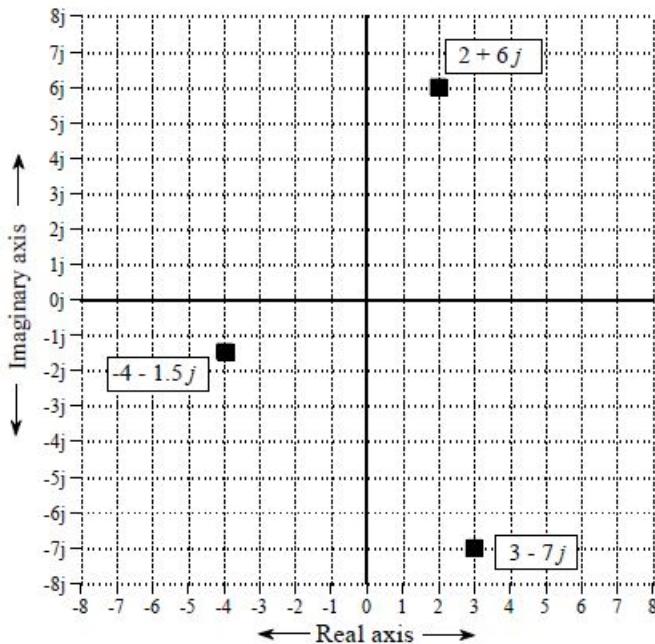
$$(a + bj)(c + dj) = (ac - bd) + j(bc + ad) \quad (2.5)$$

- operasi bagi

$$\frac{(a + bj)}{(c + dj)} = \left(\frac{ac + bd}{c^2 + d^2} \right) + j \left(\frac{bc - ad}{c^2 + d^2} \right) \quad (2.6)$$

Complex number dapat dalam bentuk 2 notasi yaitu:

- *Rectangular Notation* Pada notasi *Rectangular*, *Complex number* dapat direpresentasikan dalam *two-dimensional display/complex plane* (Gambar 2.17). Sumbu *x* pada *complex plane* merepresentasikan komponen *real* dan sumbu *y* merepresentasikan komponen *imaginary*.

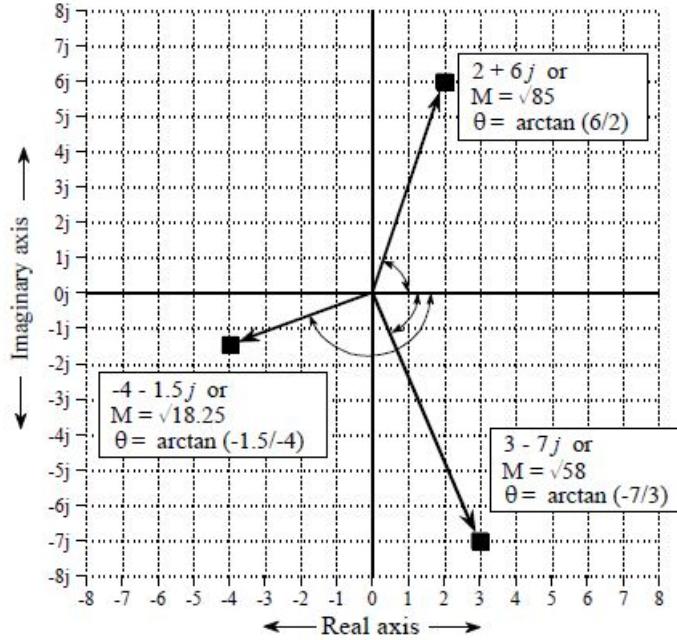


Gambar 2.17: Contoh dari *complex plane* pada notasi *rectangular*

- *Polar Notation* Pada notasi *Polar* terdapat *magnitude / panjang vektor / amplitude* dan *phase angle* yaitu besar sudut antara vektor dan sumbu positif *x*. Representasi *complex number* pada notasi *polar* seperti pada .

$$M = \sqrt{(Re A)^2 + (Im A)^2} \quad (2.7)$$

$$\Theta = \arctan \left[\frac{Im A}{Re A} \right] \quad (2.8)$$



Gambar 2.18: Contoh dari *complex plane* pada notasi polar

Pada *complex number* terdapat *euler relation* dengan persamaan 2.9.

$$e^{jx} = \cos(x) + j \sin(x) \quad (2.9)$$

2.4.2 Discrete Fourier Transform (DFT) [9] [8]

DFT dapat memiliki 2 versi yaitu *real version* dan *complex version* (2.4). DFT *complex version* dapat dikembangkan menjadi FFT (2.4.3). DFT yang menggunakan *complex number* memiliki persamaan seperti persamaan 2.10.

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{-j(\frac{2\pi}{N})nk} \quad (2.10)$$

Pada persamaan 2.10, N merupakan jumlah sample, $x(n)$ merupakan input dari sinyal pada domain waktu, dan $X(k)$ merupakan output dari sinyal pada domain frekuensi.

2.4.3 Fast Fourier Transform (FFT) [9] [10]

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

Algoritma *Cooley-Tukey* yang menggunakan panjang (N) berkelipatan 2 disebut dengan *Radix-2 Algorithm*. Pada algoritma ini ada 2 jenis proses disebut dengan *Decimation in Time* (DIT) dan *Decimation in Frequency*. Jika masukan adalah sinyal pada domain waktu, maka diproses dengan DIT dan jika masukan sinyal pada domain frekuensi, maka diproses dengan DIF. DIT akan mengubah sinyal dari domain waktu ke domain frekuensi dan DIF akan mengubah sinyal dari domain frekuensi ke domain waktu. Setiap masukan untuk proses DIT dan DIF harus dalam urutan *bit reverse* (2.4.3) agar keluaran dari DIT dan DIF dalam urutan yang sebenarnya. Algoritma dari Radix-2 DIT dapat dilihat pada 2.4.3 dan contoh visual dari DIT dengan panjang FFT adalah 8 pada Gambar 2.19.

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

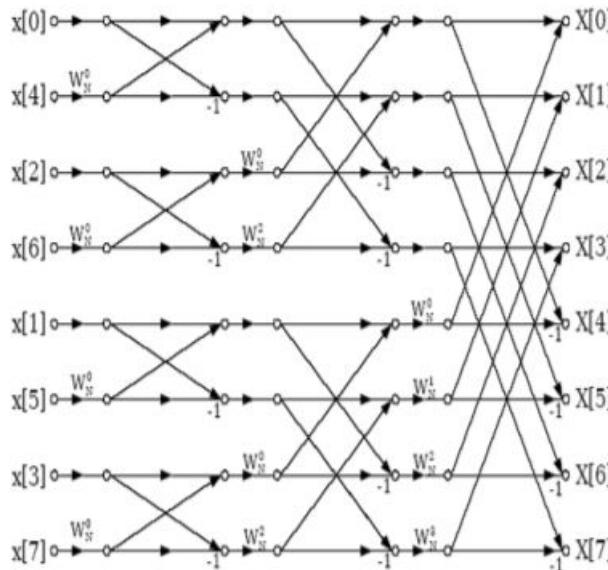
```

$$W_N^k = e^{-j(\frac{2\pi k}{N})} \quad (2.11)$$

Persamaan 2.11 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 7



Gambar 2.19: contoh visual *Decimation In Time* dengan panjang 8

Jika pada proses DIT masukan dengan indeks genap dikurang dengan masukan indeks ganjil yang sudah dikalikan dengan W_N^k dan masukan indeks ganjil ditambah dengan masukan dengan indeks ganjil yang sudah dikalikan dengan W_N^k , maka proses tersebut menjadi proses DIF.

Selain dari *Cooley-Tukey Algorithm* terdapat pula algoritma FFT lain seperti *Prime factor Algorithm*, *Rader's FFT Algorithm*, *Bluestein's FFT Algorithm*, dan *Winograd FFT Algorithm*.

2.4.4 Short Time Fourier Transform (STFT) [11]

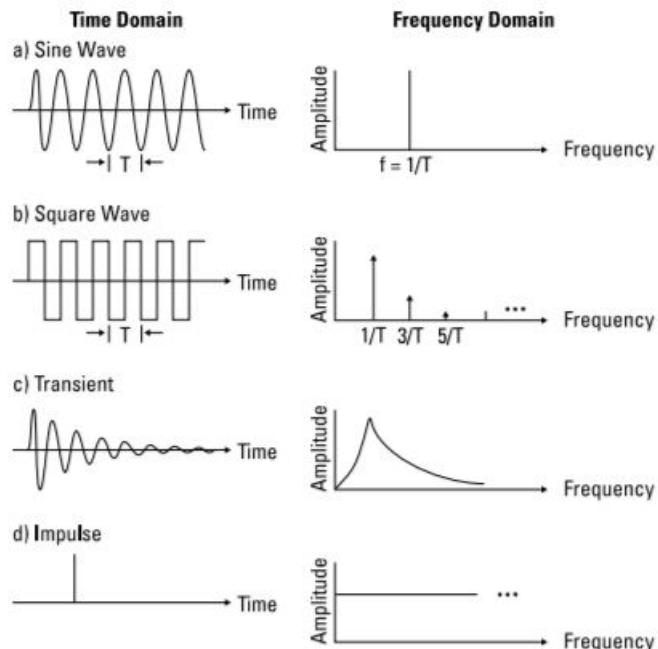
Algoritma ini adalah pengembangan dari algoritma FFT [12]. Pada algoritma ini masukan sinyal pada domain waktu dicuplik selama waktu tertentu atau sebesar *window*. Setiap window dari cuplikan waktu (*segment*) dilakukan FFT untuk mendapatkan frekuensi pada waktu cuplikan. Algoritma ini memiliki notasi sebagai berikut [13].

$$S(\omega, \tau) = F s(t) w(\tau - t) \quad (2.12)$$

Keterangan :

- $S(\omega, \tau)$ = hasil dari STFT, ω adalah frekuensi dan τ adalah waktu.
- $s(t)$ = input sinyal
- $w(t)$ = fungsi window
- F = Fourier Transform untuk input sinyal yang sudah menerapkan window

Segment dapat saling bertumpukan (*overlap*). Kegunaan dari *windows function* adalah untuk meningkatkan hasil dari *fourier transform*[14]. *Windows function* akan memengaruhi hasil FFT bergantung pada jenis gelombang. Terdapat 4 jenis gelombang yang umum yaitu, *Sinus wave*, *Square wave*, *Transient wave*, dan *Impulse wave* (Gambar 2.20).



Gambar 2.20: Jenis gelombang

Pada gelombang *transient*, *window function* yang terbaik adalah *Rectangle Window* [14]. Gelombang *trancient* bersifat *aperiodic* sehingga sinyal yang ditangkap tidak harus direduksi. Pada gelombang yang selain *trancient*, *windows function* dapat meningkatkan hasil pada *special measurement situations*. Berikut contoh - contoh dari *window function* [15] :

- Rectangular Window :

$$w_R[n] = 1, n = 0, 1, 2, \dots, N - 1; w_R[n] = 0, elsewhere \quad (2.13)$$

Keterangan :

- N = panjang window

- Hanning window :

$$w[n] = 0.5(1.0 - \cos(\frac{2n\pi}{N})), n = 0, 1, 2, \dots, N - 1 \quad (2.14)$$

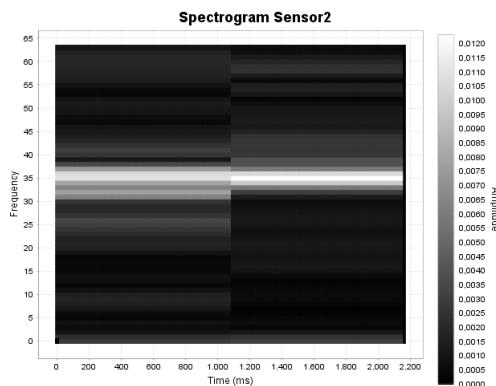
- N = panjang window

- Hamming window :

$$w[n] = 0.54 - 0.46\cos[\frac{2n\pi}{N}], n = 0, 1, 2, \dots, N - 1 \quad (2.15)$$

- N = panjang window

Hasil grafik dari *magnitude STFT* disebut dengan spectrogram. *Magnitude* didapatkan dari operasi *absolute* (2.7) hasil dari STFT. Dari spectrogram terlihat perubahan besaran amplitudo dari frekuensi pada selang waktu. Berikut contoh dari spectrogram (Gambar 2.21).



Gambar 2.21: Contoh Spectrogram

- Sumbu x pada spectrogram adalah waktu
- Sumbu y pada spectrogram adalah frekuensi
- Sumbu z pada spectrogram adalah *amplitude / magnitude* hasil STFT

Dari Gambar 2.21 terlihat perubahan warna dari warna gelap ke warna terang. Perubahan warna ini menunjukkan perubahan dari amplitudo frekuensi dari waktu ke waktu. Warna terterang pada spectrogram menunjukkan frekuensi dominan dari sampel tersebut.

2.5 PreonVM & Preon32

Preon32 adalah sensor node buatan VIRTENIO dan PreonVM merupakan *operating software* yang digunakan Preon32. Berikut penjelasan lebih detil tentang PreonVM dan Preon32.

2.5.1 PreonVM

PreonVM merupakan *virtual machine* buatan VIRTENIO untuk *embedded system* dengan sumber daya yang sangat kecil¹. *Virtual machine* dari PreonVM sudah sangat optimal dan sudah berjalan langsung di *microcontroller*, sehingga tidak membutuhkan sistem operasi tambahan pada sensor node. *Developer* dapat menggunakan *libraries* yang disediakan PreonVM dalam bahasa pemrograman Java untuk mengambil data dari sensor dan mengatur aktuator pada sensor node.

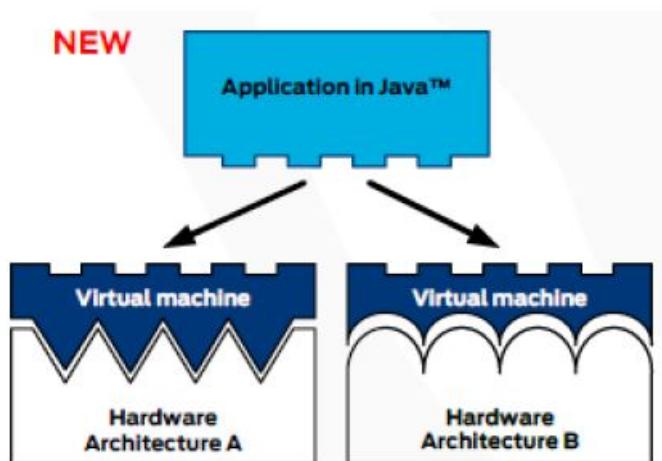
Fitur PreonVM

PreonVM memiliki fitur sebagai berikut:

- Aplikasi diprogram dengan bahasa pemrograman Java
- Mendukung semua tipe data pada Java (*char, byte, int, long, float* atau *double*)
- Jumlah *thread* tidak dibatasi
- *Exception handling* (*try, catch, Exception*, atau *Runtime Exception*)
- *Garbage collection* dengan *memory defragmentation*
- *System properties* untuk mengonfigurasi aplikasi

Kelebihan PreonVM

PreonVM menggunakan *object-oriented programming* pada *virtual machine*-nya untuk *embedded system*. PreonVM dioptimasi sedemikian rupa agar aplikasi dapat dijalankan 8-bit sampai 32-bit *microcontroller* dengan 8KB RAM dan 128KB Flash minimum. *Virtual machine* PreonVM memampukan aplikasi secara independen berjalan pada arsitektur yang digunakan. Dengan demikian arsitektur yang berbeda tidak memengaruhi aplikasi yang dijalankan (Gambar 2.22). *Virtual machine* PreonVM juga mampu memisahkan proses yang dilakukan *hardware* dan *software*.



Gambar 2.22: *Virtual machine* yang memampukan aplikasi berjalan secara independen

¹<https://www.virtenio.com/en/portfolio-items/preonvm/>

Class Library PreonVM

Packages yang terdapat pada PreonVM dapat dibagi menjadi 2 *package*, yaitu *package* dari Virtenio Tabel 2.2 dan *package* dari Java Tabel 2.4.

Tabel 2.2: Tabel Package dari Virtenio pada PreonVM

Package	Dekripsi
com.virtenio.crypt	Paket dari Virtenio untuk enkripsi dan dekripsi
com.virtenio.driver	
com.virtenio.driver.adc	
com.virtenio.driver.button	
com.virtenio.driver.can	
com.virtenio.driver.cpu	
com.virtenio.driver.device	
com.virtenio.driver.device.at86rf231	
com.virtenio.driver.flash	
com.virtenio.driver.gpio	
com.virtenio.driver.i2c	
com.virtenio.driver.irq	
com.virtenio.driver.led	
com.virtenio.driver.lin	
com.virtenio.driver.onewire	
com.virtenio.driver.pwm	
com.virtenio.driver.ram	
com.virtenio.driver.realtimeclock	
com.virtenio.driver.realtimecounter	
com.virtenio.driver.spi	
com.virtenio.driver.switch_	
com.virtenio.driver.timer	
com.virtenio.driver.usart	
com.virtenio.driver.watchdog	

Tabel 2.3: Tabel Package dari Virtenio pada PreonVM

Package	Dekripsi
com.virtenio.flashlogger	
com.virtenio.io	
com.virtenio.math	
com.virtenio.misc	
com.virtenio.net	
com.virtenio.net.lowpan	
com.virtenio.preon32.cpu	
com.virtenio.preon32.mainboard	
com.virtenio.preon32.node	
com.virtenio.preon32.shuttle	
com.virtenio.radio	
com.virtenio.radio.ieee_802_15_4	
com.virtenio.route.aodv	
com.virtenio.vm.event	

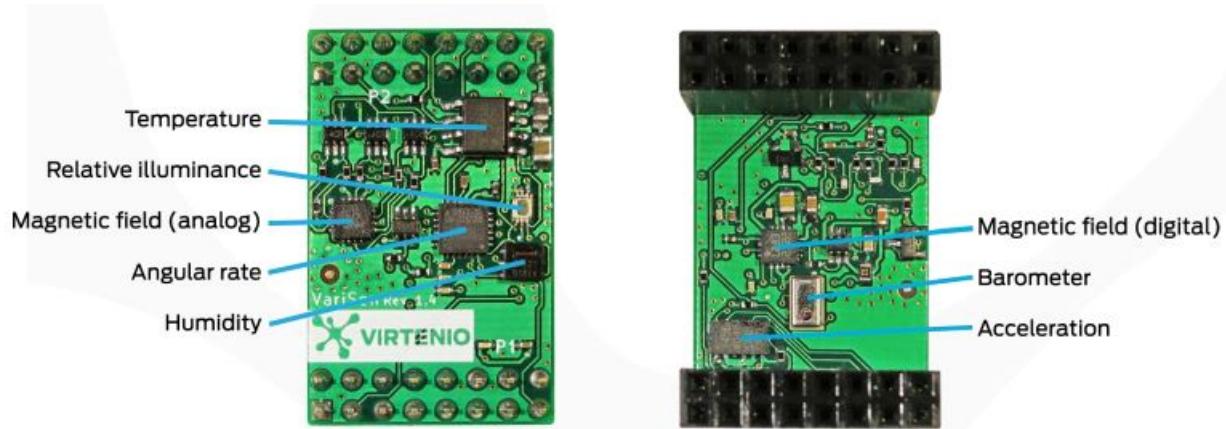
Tabel 2.4: Tabel Package dari Java pada PreonVM

Package	Deksripsi
java.io java.lang java.lang.ref java.math java.nio java.nio.channels java.text java.util java.util.concurrent java.util.concurrent.atomic java.util.concurrent.locks java.util.regex	Paket dari Java

2.5.2 Preon32

² Preon32 merupakan sensor node buatan Virtenio. PreonVM digunakan sebagai *operating software* untuk sensor node ini. Preon32 dapat menjadi 2 jenis yaitu Preon32 versi biasa dan Preon32 versi tambahan. Preon32 versi biasa memiliki sensor suhu (*temperature sensor*), sensor cahaya (*light intensity sensor*), sensor kelembaban udara (*relative humidity sensor*), sensor tekanan udara (*air pressure sensor / barometer*), dan sensor getaran (*acceleration sensor / accelerometer*). Pada Preon32 versi tambahan dilengkapi dengan sensor pendekripsi medan magnet dan *gyroscope* (Gambar 2.23).

Satuan pengukuran *raw data* dari sensor akselerometer di Preon32 adalah *bits*. Hasil pengukuran ini dapat diubah menjadi satuan standar gravitasi ($1g = 9.80665m/s^2$). ³



Gambar 2.23: Preon32

Spesifikasi Sensor Preon32

Berikut spesifikasi sensor yang dimiliki Preon32:

- sensor suhu (*temperature sensor*)
 - Manufacture : Analog Devices
 - Model : ADT7410

²<https://www.virtenio.com/en/portfolio-items/preon32/>

³https://en.wikipedia.org/wiki/Standard_gravity

- Interface : digital, I2C
- Resolution : 16-Bit
- Range : -40°C sampai +105°C
- Accuracy : ±0.5°C
- sensor cahaya (*light intensity sensor*)
 - Manufacture : Rohm
 - Model : BH1715FVC
 - Interface : digital, I2C
 - Resolution : 16-Bit
 - Range : 1 lx to 65355 lx
- sensor kelembaban udara (*relative humidity sensor*)
 - Manufacture : Sensirion
 - Model : SHT21
 - Interface : digital, I2C
 - Resolution : 12-Bit
 - Range : 0 %RH sampai 100 %RH
 - Accuracy : ±2,0 %RH (typ.)
- sensor tekanan udara (*air pressure sensor / barometer*)
 - Manufacture : Freescale
 - Model : MPL115A2
 - Interface : digital, I2C
 - Resolution : 0,15 kPa
 - Range : 50 kPa sampai 115 kPa
 - Accuracy : ±1.0 kPa
- sensor getaran (*acceleration sensor / accelerometer*)
 - Manufacture : Analog Devices
 - Model : ADXL345
 - Interface : digital, SPI
 - Resolution : 13 Bit per axis
 - Range : ±16 g, 3 axis
 - Accuracy : 3,9 mg/LSB

BAB 3

ANALISIS

Pada bab ini dijelaskan mengenai analisis aplikasi ekstraksi fitur domain waktu/frekuensi untuk data akselerometer di WSN, analisis proses ekstraksi fitur domain waktu/frekuensi untuk data akselerometer, dan analisis algoritma ekstraksi fitur domain waktu/frekuensi.

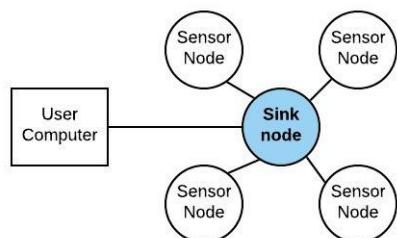
3.1 Analisis Aplikasi Ekstraksi Fitur Domain Waktu/Frekuensi Untuk Data Akselerometer di WSN

Berdasarkan latar belakang pada Bab 1 dan pembahasan landasan teori pada Bab 2, penelitian ini akan membangun aplikasi ekstraksi fitur domain waktu/frekuensi pada data akselerometer di sensor node WSN. Untuk membangun aplikasi ini akan dibuat 3 program, yaitu program yang akan berjalan di sensor node dan melakukan ekstraksi fitur, program yang akan berjalan di sensor node sebagai *base station* untuk menerima hasil ekstraksi fitur, dan program yang akan berjalan pada komputer pengguna dalam menampilkan hasil ekstraksi fitur.

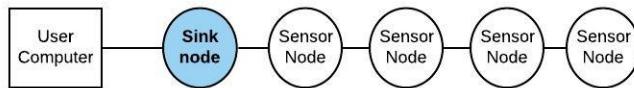
Pada subbab selanjutnya akan dijelaskan mengenai analisis topologi, arsitektur, akselerometer, fungsi aplikasi, dan kelas untuk aplikasi ini.

3.1.1 Analisis Topologi dan Arsitektur WSN

Berdasarkan pembahasan Subbab 2.1.3 mengenai topologi dan Subbab 2.1.4 mengenai arsitektur WSN, topologi WSN pada aplikasi ini dapat menggunakan WSN berarsitektur *flat* dengan topologi *star* atau *tree*. WSN dengan Arsitektur *flat* dengan seluruh sensor node memiliki tugas yang sama sangat cocok dengan penelitian ini yang hanya melakukan ekstraksi fitur data akselerometer. WSN dengan topologi *star* dan *tree* juga cocok dengan penelitian ini karena telah mencakup semua tipe komunikasi WSN. Topologi *star* memiliki komunikasi yang selalu *single-hop* dan *sink* sebagai *central communication*(Gambar 3.1). Topologi *tree* dapat memiliki komunikasi *single-hop* atau *multi-hop* dan *sink* sebagai *root* dari *tree*(Gambar 3.2).



Gambar 3.1: WSN dengan topologi *star* dan komunikasi *single-hop*



Gambar 3.2: WSN dengan topologi *tree* dengan komunikasi *single-hop* dan *multi-hop*

3.1.2 Analisis Akselerometer

Akselerometer yang digunakan pada penelitian ini adalah akselerometer ADXL345 dari sensor node Preon32 (Subbab 2.5.2). Jenis dari akselerometer ini adalah *three-axis* dan *absolute accelerometer* yang telah dibahas pada Subbab 2.2. Hasil pengukuran dari akselerometer dikonversi ke dalam satuan gravitasi (g) dan diubah ke dalam satuan akselerasi (m/s^2). Pengubahan satuan ini dilakukan hanya untuk mempermudah visualisasi dari hasil pengukuran. Pengukuran akselerometer yang divisualisasikan adalah amplitudo dari akselerasi. Dari visualisasi pengukuran ini dapat dilihat fluktiasi dari akselerasi yang terukur oleh akselerometer.

3.1.3 Analisis Fungsi Aplikasi

Fungsi utama dari aplikasi ini ada pada program yang dijalankan pada komputer pengguna. Tujuan Fungsi ini mengatur interaksi komputer pengguna, *base station*, dan sensor node. Dengan fungsi utama ini pengguna hanya perlu mengoperasikan aplikasi ini di komputer dan dapat menerima data hasilnya di komputer. Berikut fungsi - fungsi tersebut.

1. Memeriksa sensor node yang aktif
2. Menyamakan waktu semua sensor node
3. Memberikan perintah *sense* ke sensor node
4. Memberikan perintah berhenti / *stop sensing* ke sensor node
5. Menyimpan hasil ekstraksi fitur dari *sensing* sensor node
6. Menampilkan hasil ekstraksi fitur dari *sensing* sensor node

Fungsi memeriksa sensor node yang aktif diperlukan oleh pengguna untuk memastikan sensor node pada WSN sedang aktif. Jika terdapat sensor node yang tidak aktif pengguna dapat melakukan perbaikan dan konfigurasi kembali WSN tersebut.

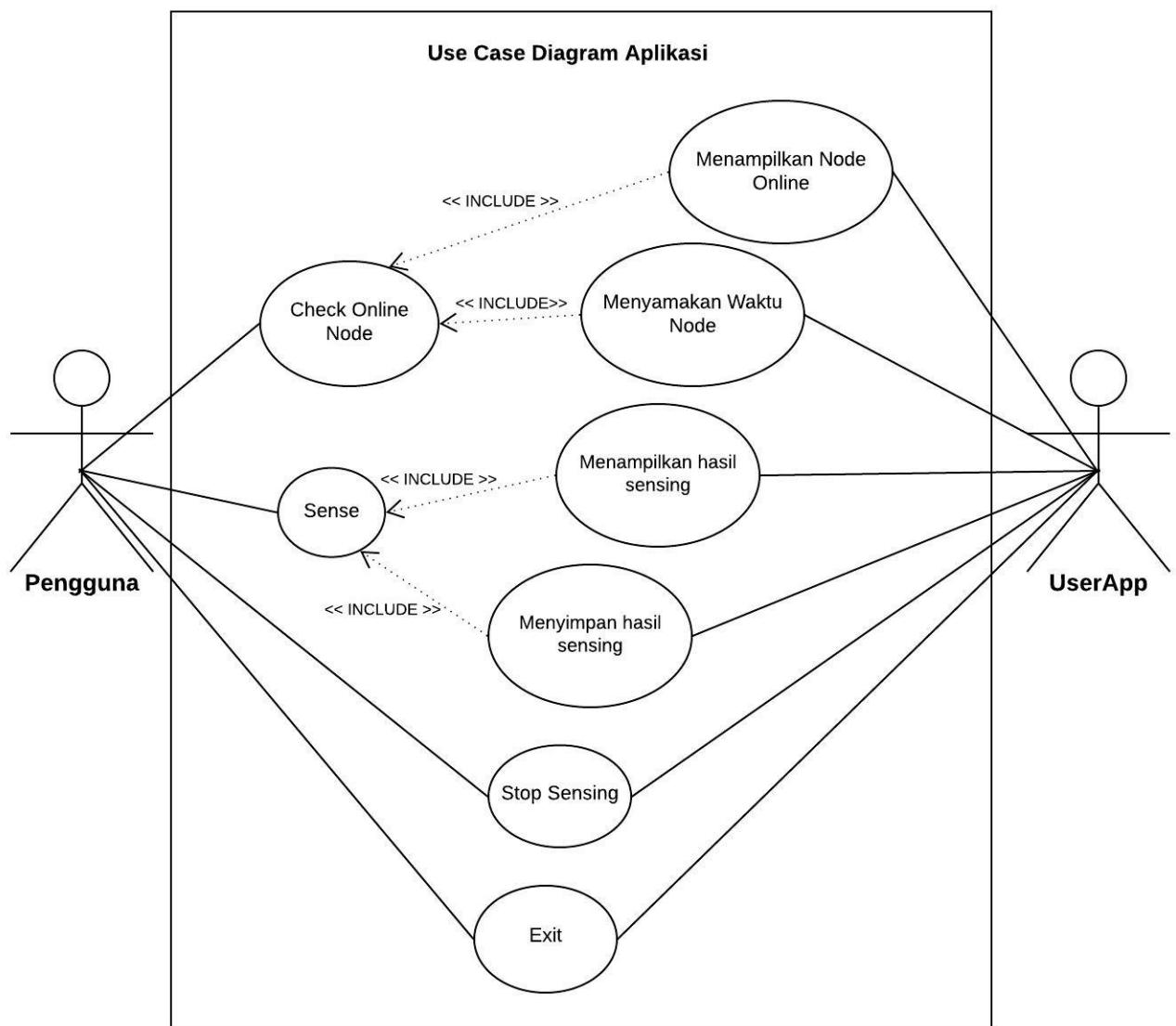
Fungsi menyamakan waktu semua sensor node diperlukan karena memory di sensor node yang bersifat *volatile* (2.1.2). Jika sensor node mati maka waktu yang ada pada sensor node adalah waktu terakhir sebelum sensor node itu mati sehingga diperlukan penyamaan waktu kembali.

Fungsi memberikan perintah *sense* ke sensor node digunakan untuk memulai *sensing* getaran pada setiap sensor node dengan akselerometer. Setiap hasil pengukuran tersebut akan langsung diekstraksi fitur dan dikirimkan ke *sink* bersama waktu saat *sensing*.

Fungsi memberikan perintah berhenti *sensing* ke sensor node digunakan untuk memberhentikan perintah *sensing* getaran pada sensor node.

Fungsi menampilkan hasil ekstraksi fitur dari *sensing* sensor node dibutuhkan untuk perangkat lunak dapat menyimpan hasil ekstraksi fitur dalam berupa file dan divisualisasikan dalam bentuk grafik.

Fungsi - fungsi tersebut hanya dapat digunakan di program yang berjalan di komputer pengguna. Program ini membantu pengguna untuk dapat berinteraksi dengan *base station* dan sensor node. Interaksi dapat dilihat pada diagram *use case* (Gambar 3.3) dan tabel skenario.



Gambar 3.3: Diagram *use case* Aplikasi Ekstraksi Fitur Domain Waktu/Frekuensi Untuk Data Akselerometer di WSN

Tabel 3.1: Tabel Skenario Memeriksa sensor node yang aktif / Check Online Node.

Nama	Memeriksa sensor node yang aktif / Check Online Node
Deskripsi	Memeriksa sensor node - sensor node yang aktif dan menyamakan waktu sensor node dengan waktu yang ada pada komputer pengguna.
Aktor	Pengguna dan UserApp
Pre-kondisi	Aplikasi baru dibuka oleh pengguna.
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem menjalankan aplikasi. 2. Pengguna memasukkan total sensor node yang dipakai. 3. Sistem menampilkan option - option fungsi yang dapat dipilih oleh pengguna. 4. Pengguna memilih option "Check Online Node". 5. Sistem menampilkan sensor node dan base station yang online bersama dengan waktu setiap sensor node.

Tabel 3.2: Tabel Skenario Memberikan perintah sense ke sensor node

Nama	Memberikan perintah sense ke sensor node / Sense
Deskripsi	Memberikan perintah sense ke setiap sensor node dan hasil sense dari setiap sensor node yang aktif akan disimpan dan ditampilkan.
Aktor	Pengguna dan UserApp
Pre-kondisi	Aplikasi sudah berjalan dan fungsi "Check Online Node" sudah digunakan.
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem menampilkan option - option fungsi yang dapat dipilih oleh pengguna. 2. Pengguna memilih option "Sense". 3. Sistem menampilkan hasil sense dan ekstrasi fitur dari setiap sensor node dan menyimpan hasil sense dari setiap node

Tabel 3.3: Tabel Skenario Memberikan perintah berhenti sense / Stop Sensing

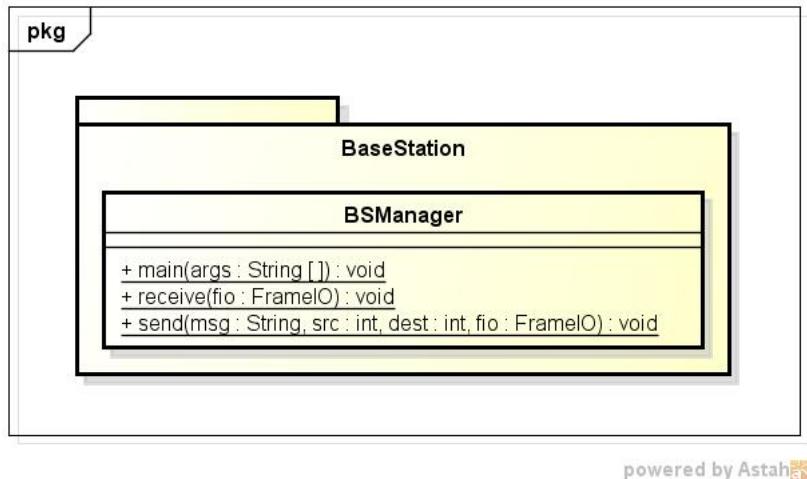
Nama	Memberikan perintah berhenti sense / Stop Sensing
Deskripsi	Memberikan perintah berhenti sense / Stop Sensing
Aktor	Pengguna dan UserApp
Pre-kondisi	Aplikasi sudah berjalan dan fungsi "Sense" sudah digunakan.
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem menampilkan option - option fungsi yang dapat dipilih oleh pengguna. 2. Pengguna memilih option "Stop Sensing" 3. Sistem memberhentikan <i>sensing</i> 4. Sistem menutup grafik visualisasi. 5. Sistem menampilkan pesan "Stop Sensing Done".

Tabel 3.4: Tabel Skenario Exit

Nama	Exit
Deskripsi	Memberikan perintah "Exit" untuk basestation dan seluruh sensor node
Aktor	Pengguna dan UserApp
Pre-kondisi	Aplikasi sudah berjalan.
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem menampilkan option - option fungsi yang dapat dipilih oleh pengguna. 2. Pengguna memilih option "Exit" 3. Sistem menampilkan pesan "program terminated". 4. Program berhenti.

3.1.4 Analisis Kelas

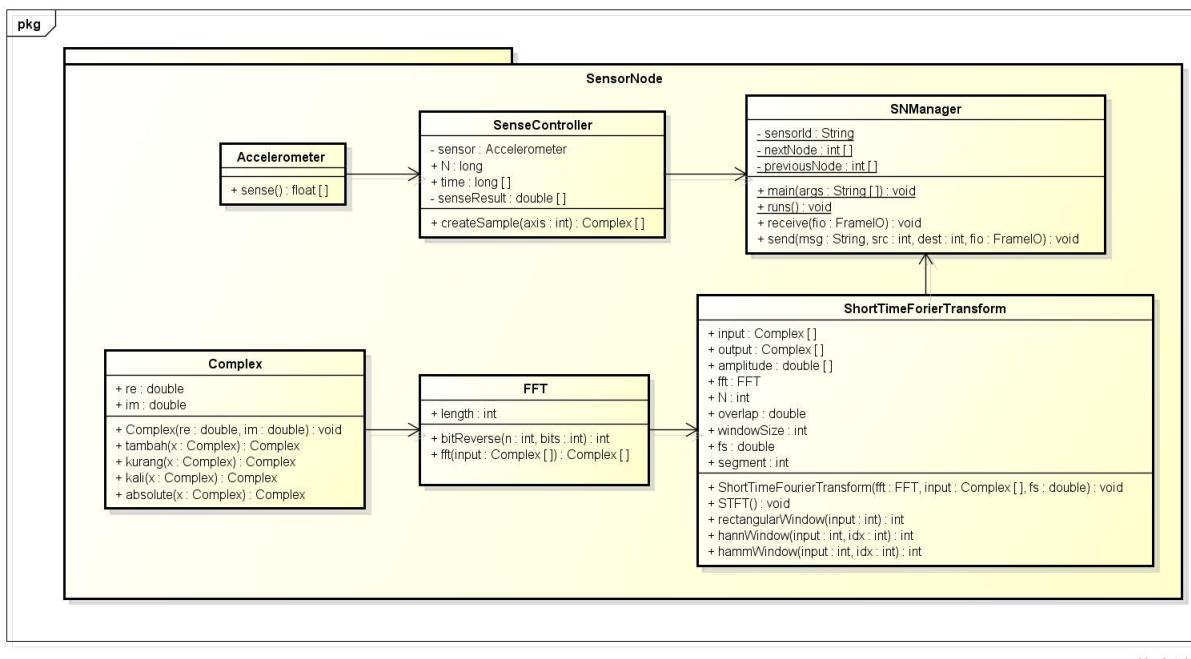
Pembuatan aplikasi ini menggunakan Eclipse IDE dan SandBox untuk sensor node Preon32 dari Virtenio. Berdasarkan analisis fungsi pada Subbab 3.1.3 maka dibuat diagram kelas sederhana untuk aplikasi ini. Berikut diagram kelas sederhana yang dibutuhkan untuk aplikasi ini.



Gambar 3.4: Kelas Diagram Sederhana Pada Aplikasi di BaseStation

Keterangan Package BaseStation:

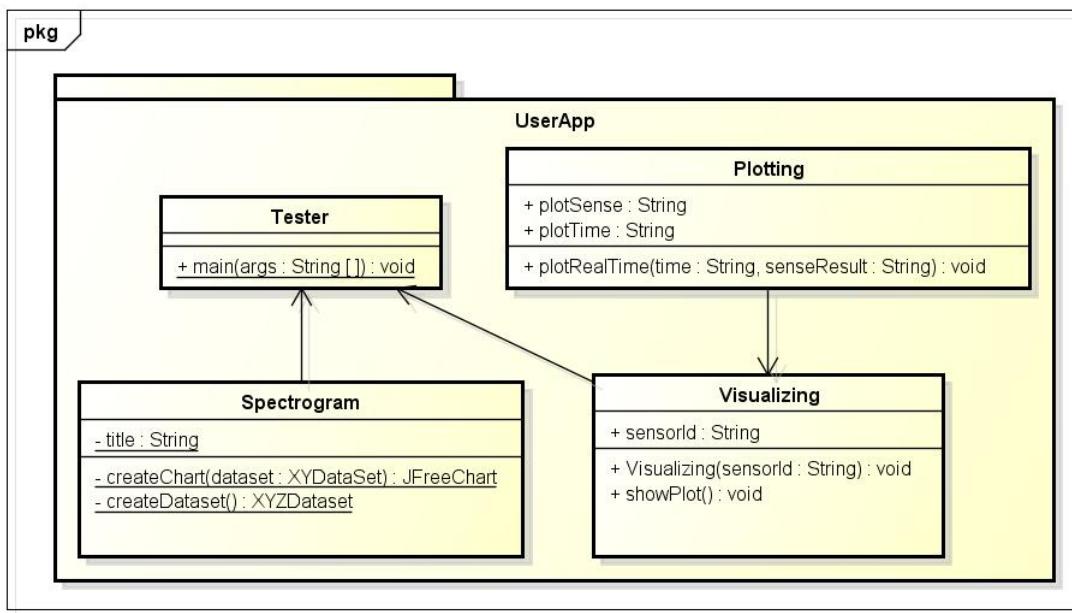
- Kelas BSManager : Kelas ini digunakan untuk sebagai **main class** dari package ini. Kelas ini akan berfungsi untuk **menerima pesan dari sensor node**, **menerima perintah dari UserApp**, dan **mengirim perintah ke sensor node**. Metode yang bertugas untuk mengirim pesan adalah metode *send()* dan Metode *receive()* berfungsi untuk menerima pesan.



Gambar 3.5: Kelas Diagram Sederhana Pada Aplikasi di SensorNode

Keterangan Package SensorNode:

- Kelas Accelerometer : Kelas ini digunakan untuk menginisialisasi akselerometer pada sensor node. Untuk melakukan *sensing* digunakan metode *sense()*.
- Kelas SenseController : Kelas ini digunakan untuk mengatur pengukuran getaran dan membuat *sample* getaran. Terdapat atribut *N* sebagai panjang *sample*, *time* untuk menyimpan waktu saat *sensing*, dan *senseResult* untuk menyimpan hasil *sensing*. Metode *createSample()* pada kelas ini digunakan untuk membuat *sample*.
- Kelas Complex : Kelas ini merepresentasikan bilangan kompleks. Terdapat atribut *re* untuk menyimpan bilangan riil dan *im* untuk menyimpan bilangan imajiner. Metode *tambah()*, *kurang()*, *kali()*, dan *absolute()* digunakan untuk operasi bilangan *Complex*.
- Kelas FFT : Kelas ini berisi implementasi dari algoritma FFT. Terdapat atribut *length* untuk menyimpan panjang dari FFT. Metode *bitReverse()* dan *fft()* sebagai implementasi dari algoritma FFT.
- Kelas ShortTimeFourierTransform : Kelas ini berisi implementasi dari algoritma STFT. Atribut *input* untuk menyimpan hasil *sensing*, *output* untuk menyimpan hasil dari STFT , *amplitude* untuk menyimpan hasil amplitude dari *output*, *fft* sebagai objek dari FFT yang digunakan, *N* sebagai panjang dari STFT, *overlap* untuk menyimpan nilai *overlap*, *windowSize* untuk menyimpan panjang dari *window* yang digunakan, *fs* untuk menyimpan nilai dari *sampling rate*, dan *segment* untuk menyimpan besar *segment* yang digunakan. Terdapat 3 metode untuk *window function* dari STFT, yaitu *rectangularWindow()*, *hannWindow*, dan *hammWindow*. Metode *STFT()* sebagai implementasi dari algoritma STFT.
- Kelas SNManager : Kelas ini menjadi main class dari package ini dan digunakan untuk menerima dan mengirim pesan / hasil ekstraksi fitur ke sensor node / base station. Atribut *sensorId* untuk menyimpan id dari sensor, *nextNode* untuk menyimpan node - node selanjutnya, dan *previousNode* untuk menyimpan node - node sebelumnya. Metode *send()* untuk mengirim pesan ke sensor node lain, *receive()* untuk menerima pesan dari sensor node lain, dan *runs()* untuk melakukan insialisasi.



powered by Astah

Gambar 3.6: Kelas Diagram Sederhana Pada Aplikasi di UserApp

Keterangan Package UserApp:

- Kelas Tester : Kelas ini digunakan sebagai main class yang menerima input dari pengguna dan menampilkan output ke pengguna.
- Kelas Visualizing : Kelas ini bertugas untuk membuat tampilan visualisasi dari hasil *sensing*. Terdapat atribut *sensorId* untuk menyimpan sensorId dan metode *showPlot()* untuk menampilkan grafik.
- Kelas Plotting : Kelas ini bertugas untuk membantu kelas Visualizing dalam membuat plot dari hasil *sensing*. Terdapat atribut *plotSense* untuk menyimpan hasil *sensing* untuk di plot dan *plotTime* untuk menyimpan waktu saat *sensing* untuk di plot. Metode *plotRealtime()* digunakan untuk menampilkan plot pada grafik di kelas Visualizing.
- Kelas Spectrogram : Kelas ini digunakan untuk menampilkan grafik spectrogram dari hasil ekstraksi fitur. Terdapat atribut *title* untuk menyimpan judul spectrogram dan metode *createDataset()* untuk membuat *data set* dari hasil ekstraksi fitur dan *createChart()* untuk membuat grafik.

3.2 Analisis Proses Ekstraksi Fitur

Pada subbab ini akan dijelaskan mengenai proses ekstraksi fitur dari hasil pengukuran akselerometer.

3.2.1 Analisis Sample Data Akselerometer

Sample data akselerometer yang digunakan untuk ekstraksi fitur diambil dari hasil pengukuran salah satu sumbu akselerometer (*x/y/z*) (3.1.2). *Sample* yang dibuat harus berkelipatan 2 (2.4.3). Pada pembuatan *sample* disimpan waktu setiap kali pengukuran dilakukan. Setelah pembuatan *sample* dihitung *sampling rate* dari pembuatan sample dengan membagi jumlah *sample* dengan waktu yang dibutuhkan untuk membuat *sample*.

3.2.2 Analisis Algoritma Short Time Fourier Transform

Proses STFT ini adalah ekstraksi fitur data akselerometer dari domain waktu ke domain frekuensi yang dicuplik sebesar *sample*(2.4.4). Besar *window* yang digunakan untuk STFT ini adalah setengah panjang *sample*. Seperti yang sudah dibahas pada Subbab 2.4.4, Algoritma STFT menggunakan algoritma FFT untuk perhitungannya. Algoritma FFT yang akan digunakan untuk aplikasi ini adalah *Cooley-Tukey Algorithm*. Algoritma ini memerlukan jumlah *sample N* berkelipatan 2 (3.2.1). *Sample* untuk algoritma ini harus berupa bilangan kompleks (2.4.2). Dalam membuat *sample* yang adalah bilangan real menjadi bilangan kompleks, maka *sample* dikonversi menjadi bilangan kompleks dengan memasukkan setiap nilai *sample* pada komponen real dan 0 pada komponen imajiner untuk setiap *sample* (2.4.1). Setelah dikonversi, *sample* akan diekstraksi fitur dengan *Rectangle window* (2.13) / *Hanning window* / *Hamming window* dengan besar setengah dari *sample* dan nilai *overlap* sebesar 0%. Dari besar *sample*, besar *window*, dan nilai *overlap* dapat dihitung besar *segment*.

Hasil dari STFT adalah berupa matrix dengan panjang kolom sebesar panjang sample dan panjang baris sebesar *segment*. Isi dari matrix adalah hasil komputasi FFT dari waktu *segment*. Setiap *segment* pada matrix dikomputasi untuk mendapatkan *amplitude*([2.7](#)). Setiap *amplitude* yang diperoleh akan diplot pada grafik spectrogram dengan frekuensi dan waktu.

Berikut pseudocode dari algoritma STFT secara sederhana.

Algorithm 2 pseudocode *STFT()* sederhana

```
1: function STFT
2:   while segment masih ada do
3:     window sample dengan window function berdasarkan besar window dan overlap
4:     Isi nilai 0 pada sample yang belum di window
5:     Komputasi sample yang sudah di window dengan algoritma FFT
6:     Hitung nilai amplitude dari setiap hasil FFT
7:   end while
8: end function
```

BAB 4

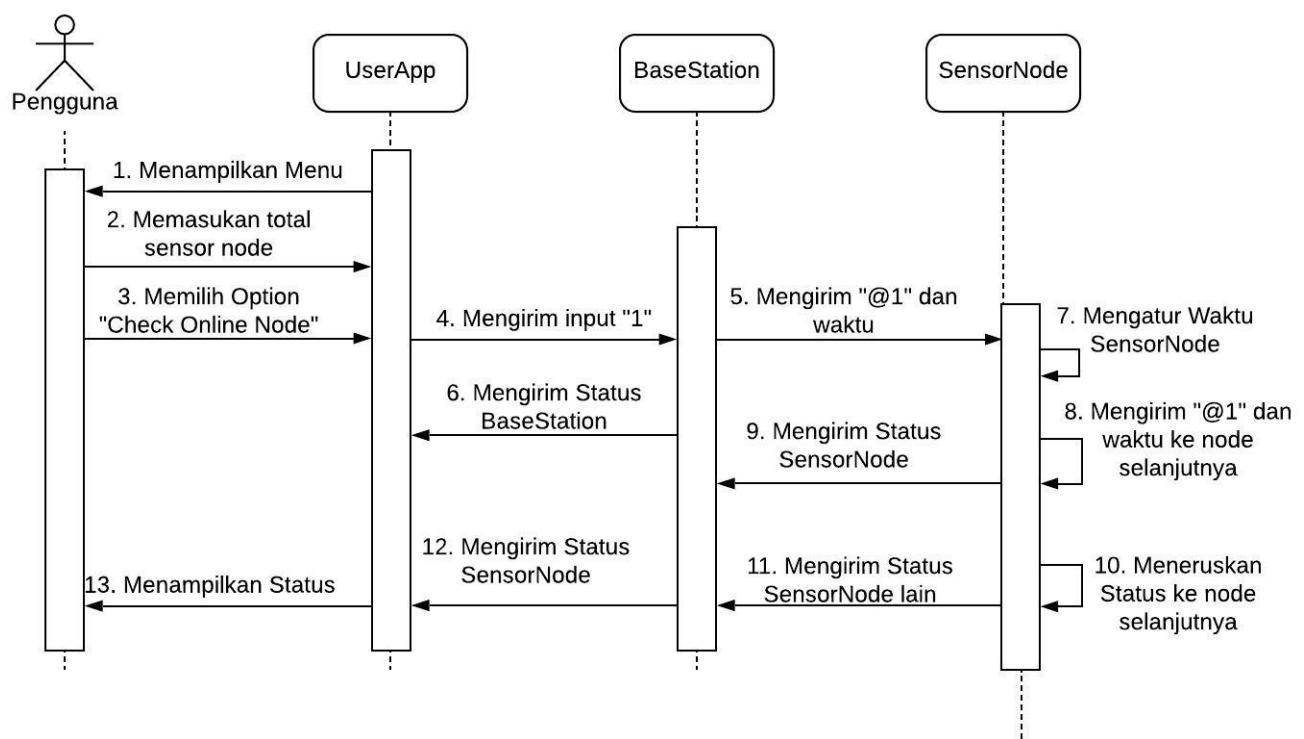
PERANCANGAN

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

4.1 Perancangan Interaksi Antar Node

Berikut penjelasan lebih lanjut mengenai fungsi - fungsi aplikasi yang sudah dijelaskan pada analisis subbab 3.1.1.

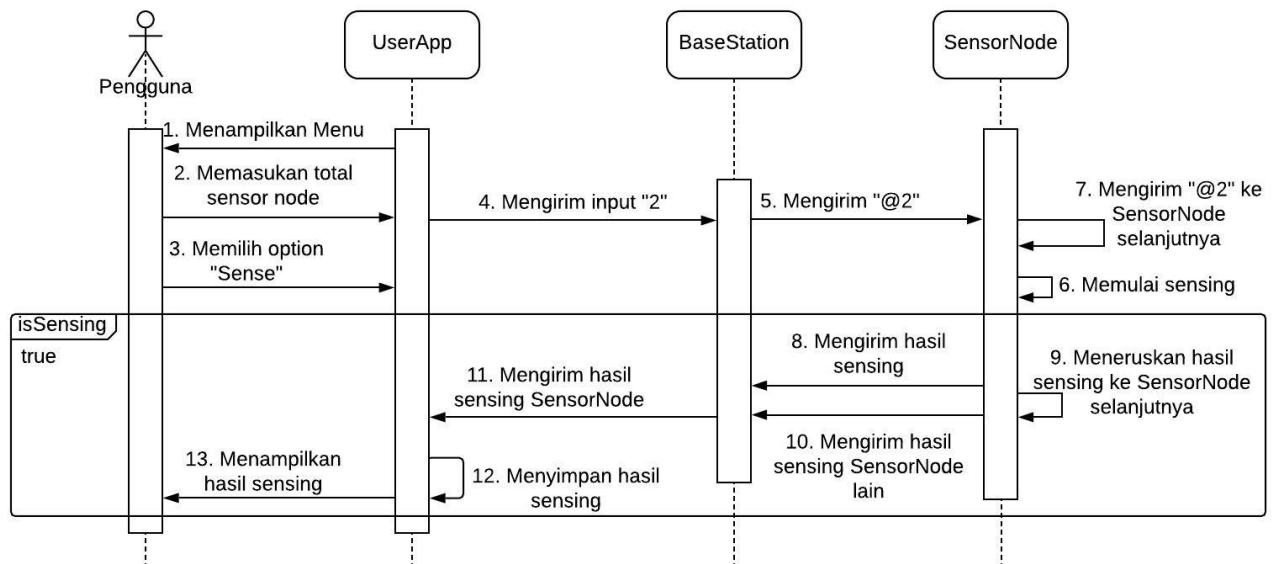
4.1.1 Diagram Sequence "Check Online Node"



Gambar 4.1: Diagram sequence "Check Online Node"

Pertama UserApp akan menampilkan menu untuk memasukkan total sensor node. Setelah pengguna memasukkan total sensor node, UserApp akan menampilkan menu fungsi - fungsi yang ada di aplikasi. Kemudian pengguna memilih option menu "Check Online Node". Fungsi Check Online Node digunakan untuk menyamakan waktu setiap sensor node dan *base station* dengan komputer pengguna dan menampilkan status online dari sensor node dan *base station*. Saat fungsi "Check Online Node" UserApp akan mengirim kode perintah "1" kepada BaseStation. Saat BaseStation menerima kode perintah "1", BaseStation akan mengirim kode perintah "@1" ke alamat sensor node yang tersimpan. Setiap SensorNode yang menerima kode perintah "@1" akan meneruskan perintah tersebut ke SensorNode selanjutnya dan mengirim status onlinenya ke BaseStation secara langsung atau melewati SensorNode lain. Jika BaseStation menerima status online dari SensorNode maka BaseStation akan meneruskan status tersebut ke UserApp dan UserApp akan menampilkan status online SensorNode tersebut.

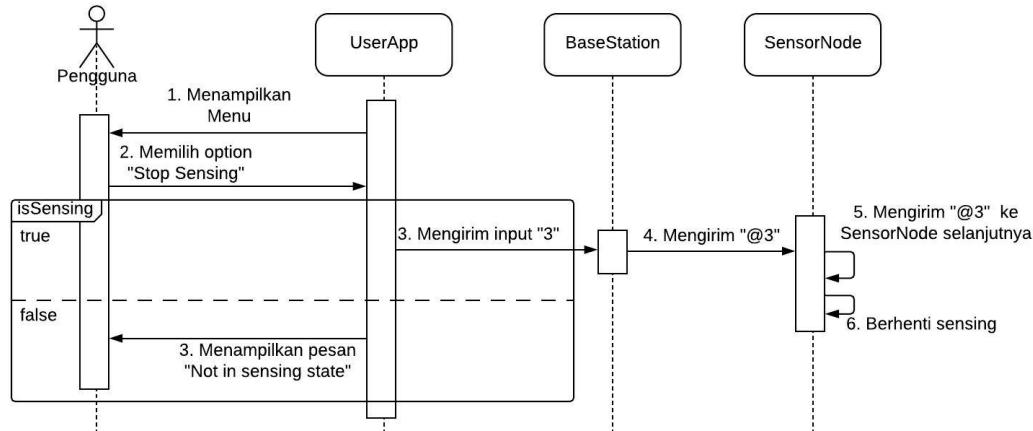
4.1.2 Diagram Sequence "Sense"



Gambar 4.2: Diagram sequence "Sense"

Pertama UserApp akan menampilkan menu untuk memasukkan total sensor node. Setelah pengguna memasukkan total sensor node, UserApp akan menampilkan menu fungsi - fungsi yang ada di aplikasi. Setelah Pengguna memilih option menu "Check Online Node", kemudian pengguna memilih option menu "Sense". Saat pengguna memilih option menu "Sense", UserApp akan mengirim kode perintah "2" ke BaseStation dan menampilkan grafik hasil sense. Setelah BaseStation menerima kode perintah "2" dari UserApp, BaseStation mengirim kode perintah "@2" ke alamat sensor node yang tersimpan. Setiap SensorNode yang menerima kode perintah "@2" akan mulai melakukan *sensing* dan meneruskan perintah tersebut ke SensorNode selanjutnya. Setiap hasil *sensing* dikirimkan ke BaseStation / SensorNode lain hingga sampai ke BaseStation. Hasil sensing yang diterima oleh BaseStation akan dikirim ke UserApp. Hasil sense akan langsung ditampilkan oleh UserApp dan disimpan dalam file oleh UserApp.

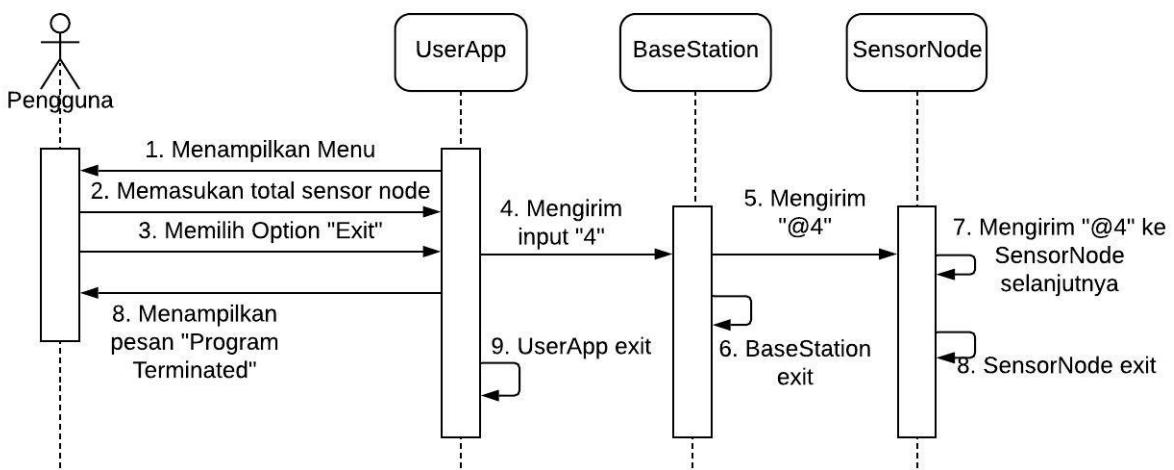
4.1.3 Diagram Sequence "Stop Sensing"



Gambar 4.3: Diagram sequence "Stop Sensing"

Pertama UserApp akan menampilkan fungsi - fungsi aplikasi. Kemudian pengguna memilih option menu "Stop Sensing". Jika SensorNode tidak sedang dalam keadaan "Sensing" maka UserApp akan menampilkan pesan "Sensor Node not in sensing state". Jika SensorNode sedang dalam keadaan sensing maka UserApp akan mengirim perintah "3" ke BaseStation. Saat kode perintah "3" diterima, BaseStation akan mengirim kode perintah "@3" ke alamat SensorNode yang tersimpan. Setiap SensorNode yang menerima kode perintah "@3" akan berhenti melakukan "sensing" dan meneruskan perintah ke SensorNode selanjutnya. Saat semua SensorNode berhenti melakukan "Sensing" UserApp akan menutup grafik hasil sense dan mengampilkan pesan "Stop Sensing Done".

4.1.4 Diagram Sequence "Exit"



Gambar 4.4: Diagram sequence "Exit"

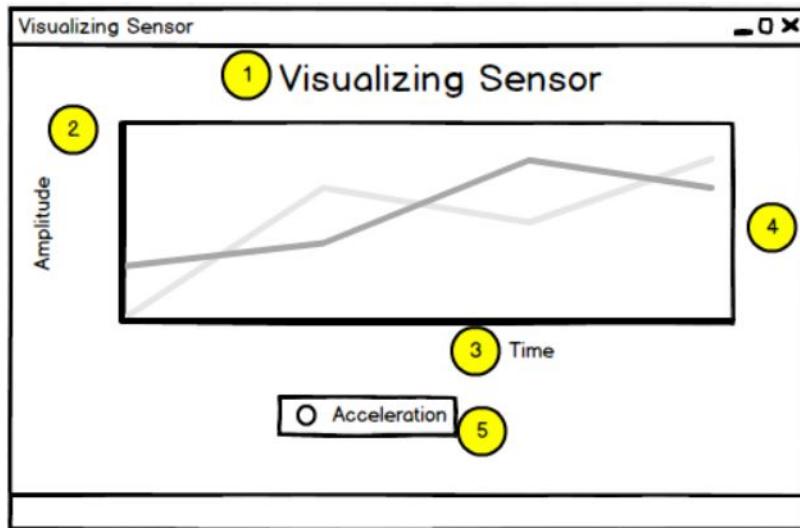
Pertama UserApp akan menampilkan menu untuk memasukkan total sensor node. Setelah pengguna memasukkan total sensor node, UserApp akan menampilkan menu fungsi - fungsi yang ada di

aplikasi. Kemudian pengguna memilih option menu "Exit". Saat pengguna memilih option menu "Exit", UserApp akan mengirim perintah "4" ke BaseStation untuk mematikan program pada BaseStation. Saat kode perintah "4" diterima, BaseStation mengirim kode perintah "@4" ke alamat SensorNode yang tersimpan. SensorNode yang menerima kode perintah "@4" akan memberhentikan program dan meneruskan kode perintah ke SensorNode selanjutnya.

4.2 Perancangan Antar Muka Untuk Visualisasi Hasil Ekstraksi

Pada Subbab 3.1.3 telah dijelaskan bahwa terdapat fungsi untuk menampilkan hasil sense. Perancangan antar muka ini memanfaatkan library JFreeChart¹. Berikut perancangan antar muka untuk tampilan hasil ekstraksi fitur.

4.2.1 Antar Muka Visualisasi Hasil Sense



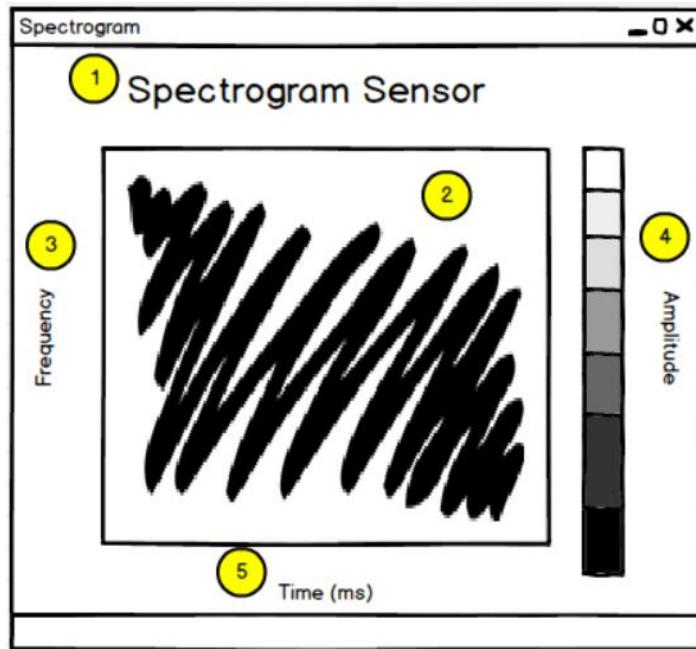
Gambar 4.5: Rancangan antar muka tampilan hasil sense

Keterangan Gambar :

- Nomor 1 : Judul dari visualisasi berdasarkan nama sensor
- Nomor 2 : label 'Amplitude' untuk sumbu y
- Nomor 3 : label 'Time' pada sumbu x
- Nomor 4 : Grafik plot dari hasil sense
- Nomor 5 : Keterangan plot pada grafik yang ditampilkan.

¹<http://www.jfree.org/jfreechart/>

4.2.2 Antar Muka Spectrogram



Gambar 4.6: Rancangan antar muka tampilan hasil sense

Keterangan Gambar :

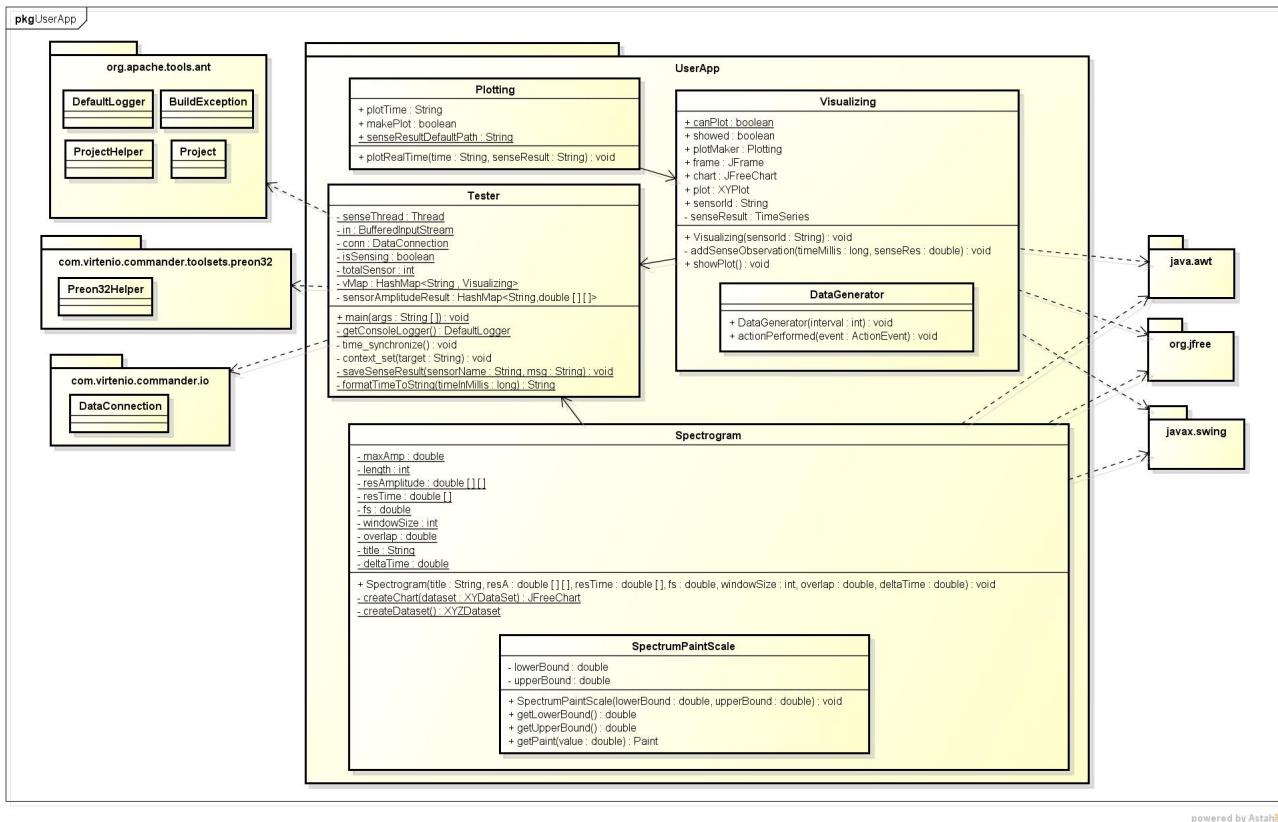
- Nomor 1 : Judul dari spectrogram berdasarkan nama sensor
- Nomor 2 : Grafik plot dari hasil ekstraksi fitur sensor node
- Nomor 3 : label 'Frequency' pada sumbu y
- Nomor 4 : label 'Amplitude' pada sumbu z berupa skala perubahan warna
- Nomor 5 : label 'Time (ms)' pada sumbu x

4.3 Perancangan Kelas Aplikasi

Pada Subbab 3.1.4 telah dijelaskan kelas diagram sederhana untuk fungsi - fungsi aplikasi ini. Berikut kelas diagram lengkap dan detil kelas dari package *UserApp*, *BaseStation*, dan *SensorNode*

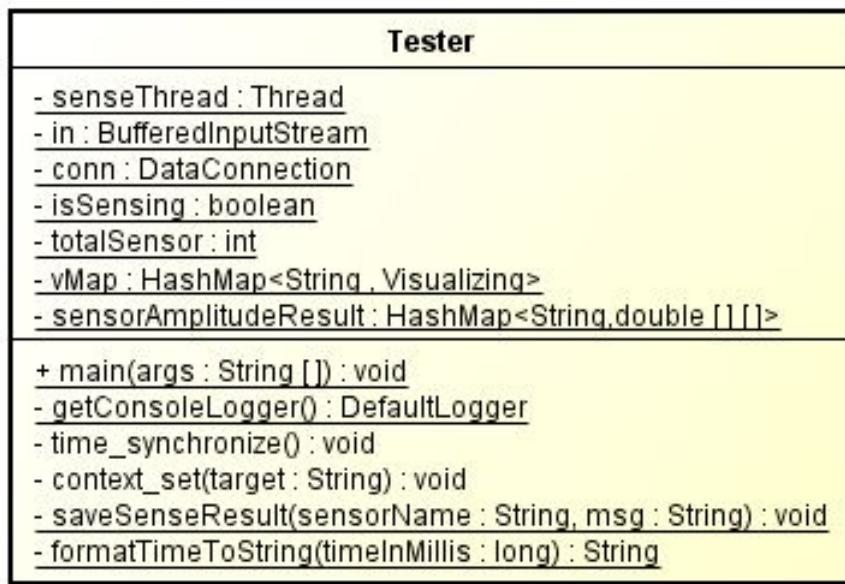
4.3.1 Package UserApp

Package ini berisi kelas *Tester* , kelas *Plotting*, kelas *Visualizing*, dan kelas *Spectrogram*.



Gambar 4.7: Kelas diagram lengkap package UserApp

Kelas Tester



Gambar 4.8: Perancangan Kelas Tester

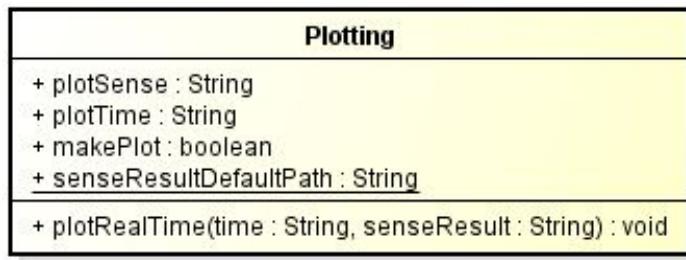
Kelas ini digunakan sebagai main class yang menerima input dari pengguna dan menampilkan output ke pengguna. Atribut - atribut yang ada pada kelas ini adalah sebagai berikut:

- private static Thread senseThread;
Atribut ini digunakan untuk membuat *thread* baru yang bertugas menerima hasil sense dari sensor node saat dalam keadaan *sensing*.
- private static BufferedInputStream in;
Atribut ini digunakan untuk membaca input dari BaseStation.
- private static DataConnection conn;
Atribut ini digunakan untuk membuat koneksi dengan BaseStation.
- private static volatile boolean isSensing;
Atribut ini digunakan untuk sebagai penanda kondisi SensorNode. Jika SensorNode dalam keadaan *sensing* maka atribut ini akan bernilai *true*.
- private static int totalSensor;
Atribut ini digunakan untuk inisialisasi jumlah plot Visualizing dan Spectrogram sesuai dengan jumlah sensor yang digunakan.
- private static HashMap <String,Visualizing> vMap;
Atribut ini digunakan untuk menyimpan objek - objek Visualizing pada HashMap.
- private static HashMap<String, double[][]> sensorAmplitudeResult;
Atribut ini digunakan untuk menyimpan hasil dari ekstraksi fitur dari sensor node - sensor node ke dalam sebuah HashMap.

Metode - metode yang ada pada 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
- private static void saveSenseResult(String sensorName, String msg)
Metode ini digunakan untuk menyimpan semua hasil sensing ke dalam file.
- private static String formatTimetoString(long timeInMillis)
Metode ini digunakan untuk mengubah format waktu dalam milli menjadi sebuah *string*.

Kelas Plotting



Gambar 4.9: Perancangan Kelas Plotting

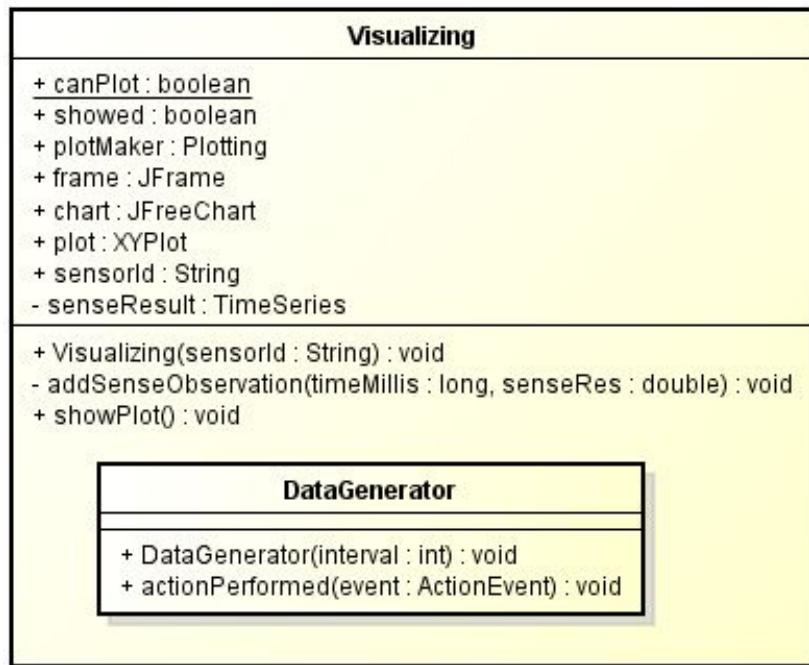
Kelas ini bertugas untuk menyimpan nilai waktu dan nilai hasil sense untuk diperbaharui oleh kelas Visualizing dalam membuat plot dari hasil *sensing*. Atribut - atribut yang ada pada kelas ini adalah sebagai berikut:

- public String plotTime;
Atribut ini menyimpan nilai waktu untuk ditampilkan pada visualisasi.
- public double plotSense;
Atribut ini menyimpan nilai sense untuk ditampilkan pada visualisasi.
- public boolean makePlot;
Atribut ini digunakan sebagai penanda pembuatan plot. Jika bernilai *true* maka Visualizing akan menampilkan plot dari nilai yang tersimpan pada plotTime dan plotSense.
- public static String senseResultDefaultPath;
Atribut ini menyimpan *directory path* penyimpanan data hasil *sensing* di komputer pengguna.

Metode - metode yang ada pada ini adalah sebagai berikut:

- public void plotRealTime(String time, String senseResult)
Metode ini digunakan untuk menyimpan nilai waktu ke dalam atribut plotTime ,nilai hasil sense ke dalam atribut plotSense, dan memperbolehkan kelas Visualizing untuk memperbaharui visualisasi yang sedang ditampilkan.

Kelas Visualisizing



Gambar 4.10: Perancangan Kelas Visualizing

Kelas ini bertugas untuk membuat dan menampilkan plot dari Kelas Plotting. Kelas ini akan terus memperbaharui plot yang sedang ditampilkan saat melakukan *sensing*. Atribut - atribut yang ada pada kelas ini adalah sebagai berikut:

- public static volatile boolean canPlot;
Atribut ini digunakan sebagai penanda dapat plot atau tidak
- public boolean showed;
Atribut ini digunakan sebagai penanda sudah ditampilkan atau belum
- public final Plotting plotMaker = new Plotting();
Atribut ini digunakan untuk membuat objek Plotting
- public JFrame frame;
Atribut ini digunakan untuk menampung object JFrame
- public JFreeChart chart;
Atribut ini digunakan untuk menampung object JFreeChart yang
- public XYPlot plot;
Atribut ini digunakan untuk mengatur plot yang divisualisasikan
- public String sensorId;
Atribut ini digunakan untuk menyimpan SensorId
- private TimeSeries amplitude;
Atribut ini digunakan untuk membuat plot time series dari amplitude
- private TimeSeries frequency;
Atribut ini digunakan untuk membuat plot time series dari frequency

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

- Visualizing(String sensorId)
Metode ini digunakan sebagai konstruktor dari Kelas Visualizing
- private void addAmplitudeObservation(long milis, double y) Metode ini digunakan untuk menambahkan plot amplitude pada grafik
- private void addFreqObservation(long milis, double y) Metode ini digunakan untuk menambahkan plot frequency pada grafik

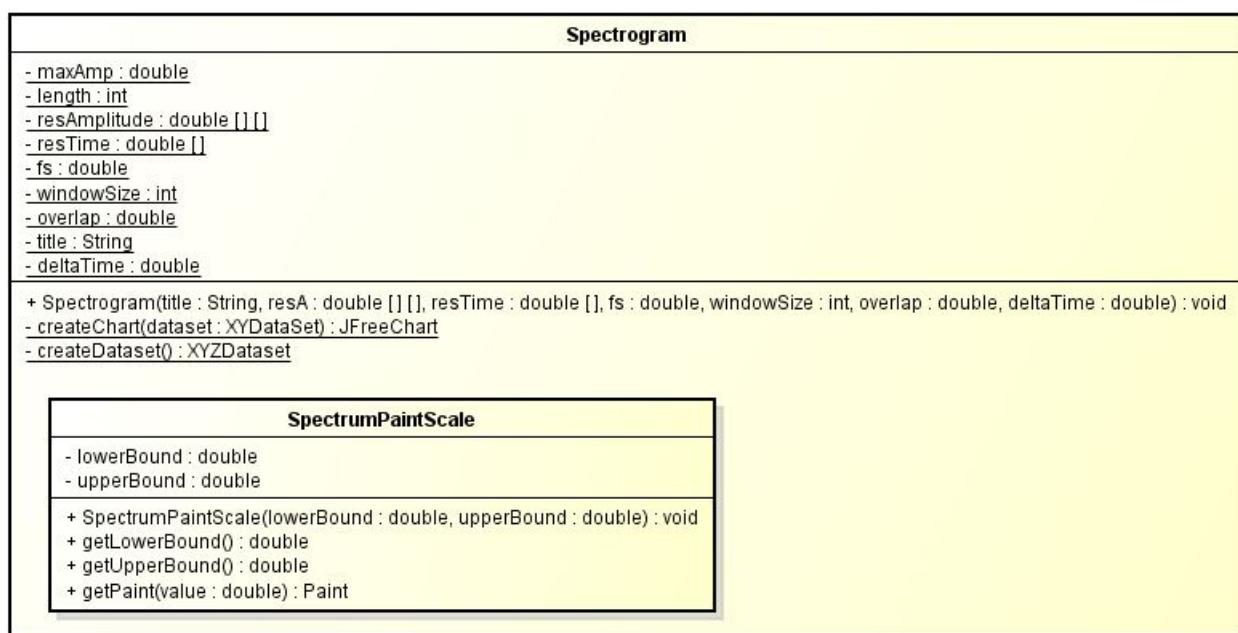
Pada kelas ini terdapat *inner class* DataGenerator yang memiliki atribut :

- private Long lastTime = new Long(0);
Atribut ini digunakan sebagai penanda waktu terakhir

kelas ini juga memiliki metode - metode sebagai berikut:

- DataGenerator(int interval)
Metode ini digunakan sebagai konstruktor dari kelas DataGenerator
- public void actionPerformed(ActionEvent event)
Metode ini digunakan sebagai aksi yang dilakukan dalam interval tertentu
- public void showPlot()
Metode ini digunakan untuk menampilkan plot pada tampilan

Kelas Spectrogram



Gambar 4.11: Perancangan Kelas Spectrogram

Kelas ini bertugas untuk membuat plot berupa Spectrogram yang menampilkan hasil ekstraksi fitur dari sensor node. Atribut - atribut yang ada pada kelas ini adalah sebagai berikut:

- private static double maxAmp;
Atribut ini menyimpan amplitudo maksimal dari hasil ekstraksi fitur sensor node.

- private static int length;
Atribut ini menyimpan panjang dari hasil ekstraksi fitur sensor node.
- private static double[][] resAmplitude;
Atribut ini menyimpan hasil dari ekstraksi fitur sensor node.
- private static double[] resTime;
Atribut ini menyimpan waktu dari ekstraksi fitur.
- private static double fs;
Atribut ini menyimpan *sampling frequency* dari ekstraksi fitur.
- private static int windowSize;
Atribut ini menyimpan besar *window* yang digunakan oleh sensor node untuk ekstraksi fitur.
- private static double overlap;
Atribut ini menyimpan nilai *overlap* yang digunakan oleh sensor node untuk ekstraksi fitur.
- private static String title;
Atribut ini menyimpan judul dari Spectrogram.
- private static double deltaTime;
Atribut ini menyimpan selisih waktu dari setiap hasil ekstraksi fitur.

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

- public Spectrogram(String title, double[][] resA, double[] resTime, double fs, int windowSize, double overlap, double deltaTime)
Metode ini digunakan sebagai konstruktor dari kelas Spektrogram.
- private static JFreeChart createChart(XYDataset dataset)
Metode ini berfungsi untuk membuat grafik spectrogram dari *dataset* yang sudah dibuat.
- private static XYZDataset createDataset()
Metode ini berfungsi untuk membuat *dataset* dari hasil ekstraksi fitur sensor node.

Pada kelas ini terdapat *inner class* SpectrumPaintScale yang memiliki atribut - atribut sebagai berikut:

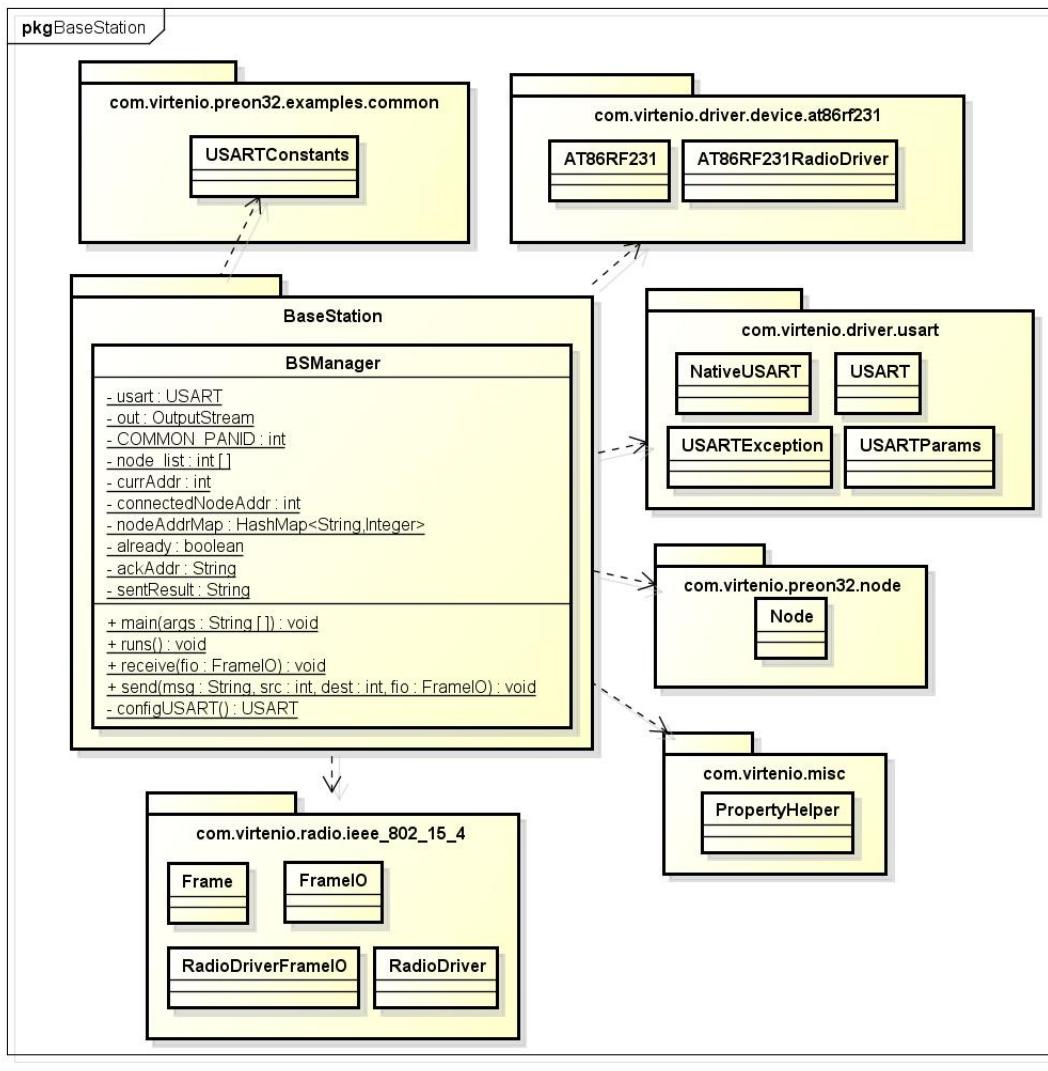
- private final double lowerBound;
Atribut ini digunakan untuk menyimpan nilai amplitudo terendah dari ekstraksi fitur.
- private final double upperBound;
Atribut ini digunakan untuk menyimpan nilai amplitudo tertinggi dari hasil ekstraksi fitur.

Kelas ini juga memiliki metode - metode sebagai berikut:

- public SpectrumPaintScale(double lowerBound, double upperBound)
Metode ini digunakan sebagai konstruktor dari kelas SpectrumPaintScale.
- public double getLowerBound()
Metode ini mengembalikan nilai dari atribut lowerBound.
- public double getUpperBound()
Metode ini mengembalikan nilai dari atribut upperBound.
- public Paint getPaint(double value)
Metode ini mengembalikan objek *Paint* berdasarkan nilai *value* yang dibandingkan dengan atribut *lowerBound* dan *upperBound*.

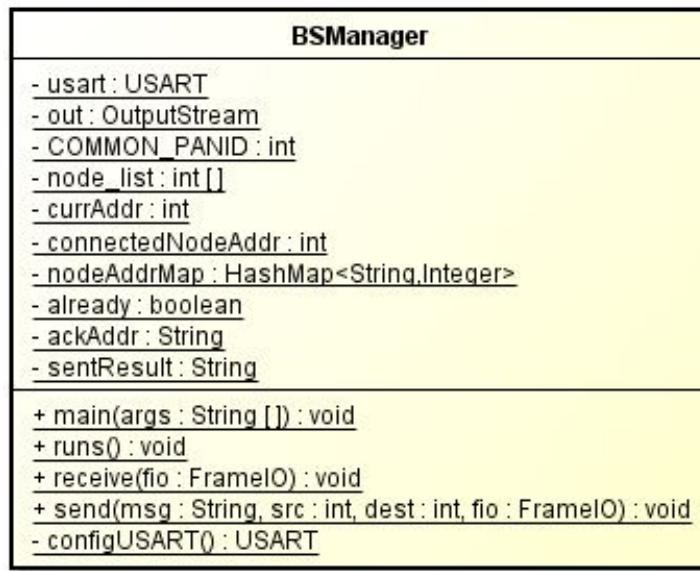
4.3.2 Package BaseStation

Package ini berisi kelas *BSManager*.



Gambar 4.12: Kelas diagram lengkap package BaseStation

Kelas BSManager



Gambar 4.13: Perancangan Kelas BSManager

Kelas ini digunakan untuk sebagai main class dari package ini. Kelas ini akan berfungsi untuk menerima perintah dari UserApp, meneruskan perintah ke sensor node, menerima pesan dari sensor node, dan meneruskan pesan ke UserApp. Kelas ini memiliki atribut - atribut sebagai berikut :

- private static USART usart; Atribut ini digunakan untuk mengatur usart pada BaseStation.
- private static OutputStream out; Atribut ini digunakan untuk mengatur outputstream ke UserApp.
- private static int COMMON_PANID; Atribut ini digunakan untuk mengatur panID.
- private static int[] node_list; Atribut ini menyimpan alamat - alamat dari sensor node.
- private static int currAddr = node_list[0]; Atribut ini menyimpan alamat dari BaseStation.
- private static int[] connectedNodeAddr; Atribut ini menyimpan alamat - alamat sensor node yang terhubung dengan BaseStation.
- private static HashMap<String, Integer> nodeAddrMap; Atribut ini menyimpan alamat - alamat dari sensor node yang terhubung dengan BaseStation ke dalam HashMap.
- private static boolean already; Atribut ini digunakan untuk menunjukkan sudahnya SensorNode mengirim data.
- private static String ackAddr; Atribut ini digunakan untuk menyimpan sensorId yang sedang mengirim data.
- private static String sentResult; Atribut ini digunakan untuk menyimpan data yang terkirim oleh sensor node.

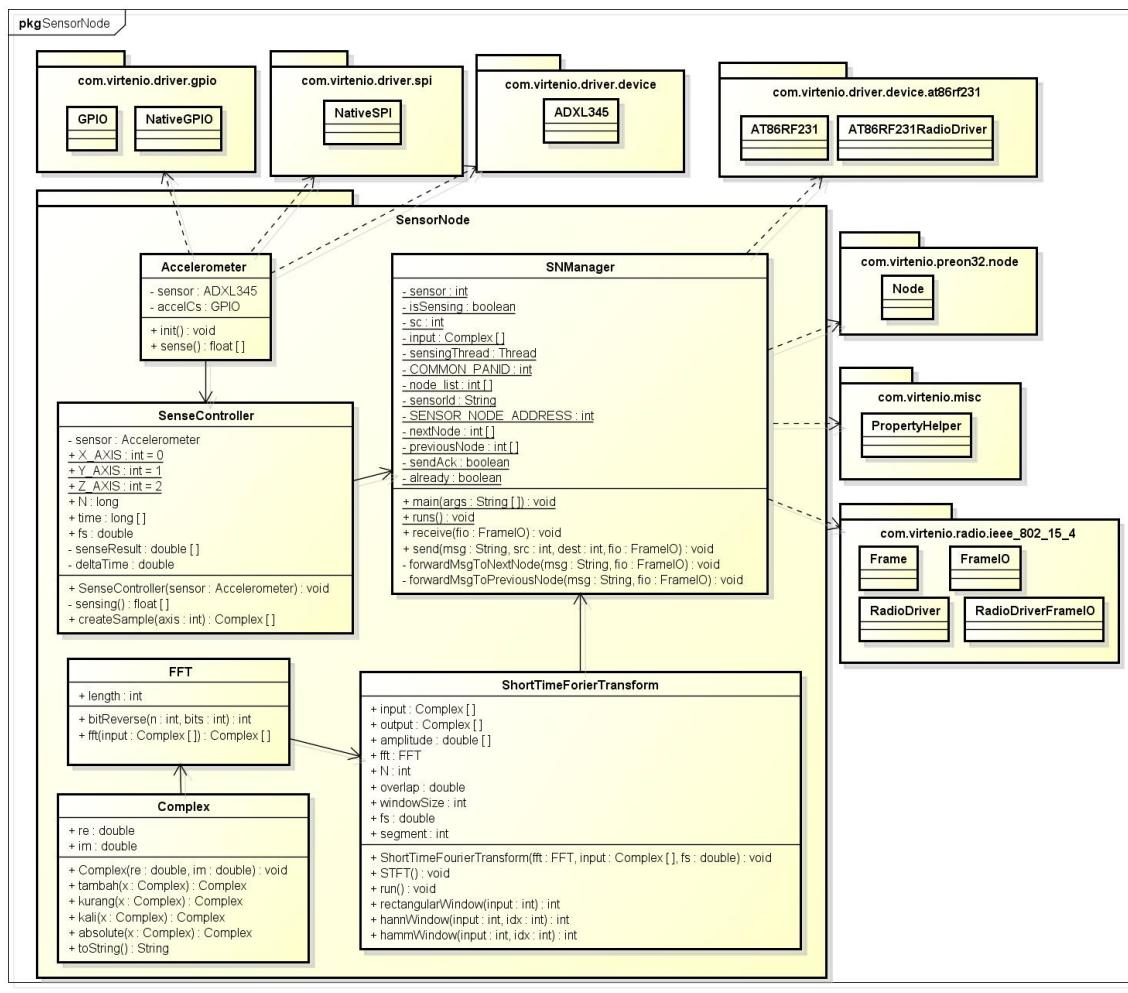
Metode - metode yang ada pada ini adalah sebagai berikut:

- public static void main(String[] args)
- Metode ini digunakan sebagai metode main kelas BSManager.

- public static void runs()
Metode ini digunakan sebagai inisialisasi radio dan memulai *thread* untuk menerima dan mengirim pesan.
- public static void receive(final FrameIO fio)
Metode ini digunakan untuk menerima pesan yang dikirim oleh sensor node.
- public static void send(String msg, int src, int dest, FrameIO fio)
Metode ini digunakan untuk mengirim pesan ke sensor node.
- private static USART configUSART()
Metode ini digunakan untuk inisialisasi usart.

4.3.3 Package SensorNode

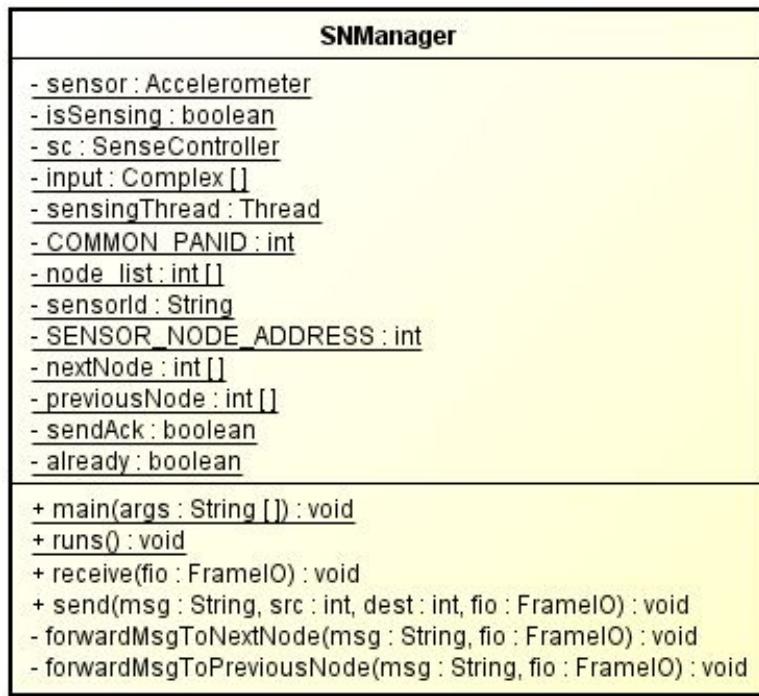
Package ini berisi kelas *SNManager*, *Complex*, *FFT*, *ShortTimeFourierTransform*, *Accelerometer*, dan *SenseController*.



powered by Astah

Gambar 4.14: Kelas diagram lengkap package SensorNode

Kelas SNManager



Gambar 4.15: Perancangan Kelas SNManager

Kelas ini digunakan untuk sebagai main class dari package ini. Kelas ini akan berfungsi untuk menerima pesan dari sensor node lain, mengirim pesan ke sensor node yang terhubung, meneruskan pesan ke sensor node lain, dan mengirim hasil ekstraksi fitur ke BaseStation atau sensor node sebelumnya. Kelas ini memiliki atribut - atribut sebagai berikut :

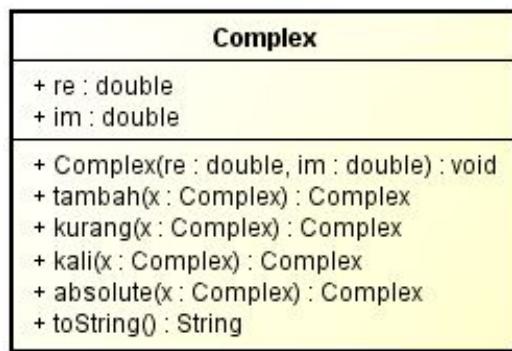
- private static final Accelerometer sensor
Atribut ini menyimpan objek kelas Accelerometer.
- private static boolean isSensing
Atribut ini menandakan kondisi sensing SensorNode. Jika *isSensing* bernilai *true* maka SensorNode sedang dalam keadaan *sensing*.
- private static SenseController sc
Atribut ini menyimpan objek SenseController.
- private static Complex[] input;
Atribut ini menyimpan input dari hasil *sensing* akselerometer.
- private static Thread sensingThread;
Atribut ini digunakan untuk membuat *thread* baru untuk melakukan *sensing*.
- private static int COMMON_PANID
Atribut ini menyimpan PanID.
- private static int[] node_list
Atribut ini menyimpan alamat - alamat dari sensor node.
- private static final String sensorId
Atribut ini menyimpan id dari sensor node.

- private static int SENSOR_NODE_ADDRESS
Atribut ini menyimpan alamat sensor node dirinya sendiri.
- private static int[] nextNode
Atribut ini menyimpan alamat - alamat dari sensor node selanjutnya.
- private static int[] previousNode
Atribut ini menyimpan alamat - alamat dari sensor node sebelumnya
- private static boolean sendAck;
Atribut ini digunakan sebagai penanda bahwa SensorNode dapat mengirim data ke BaseStation. Jika bernilai *true* maka sensor node dapat mengirim pesan ke BaseStation.
- private static boolean already;
Atribut ini digunakan sebagai penanda bahwa SensorNode sudah mengirim data ke BaseStation. Jika bernilai *true* maka BaseStation sudah menerima data yang dikirim oleh sensor node.

Metode - Metode yang ada pada ini adalah sebagai berikut:

- public static void main(String[] args)
Metode ini digunakan sebagai metode main kelas BSManager.
- public static void runs()
Metode ini digunakan sebagai inisialisasi radio dan memulai *thread* baru untuk menerima pesan dari BaseStation atau sensor node lain.
- public static void receive(final FrameIO fio)
Metode ini digunakan untuk menerima pesan yang terkirim dan mengolah perintah yang diterima.
- public static void send(String msg, int src, int dest, FrameIO fio)
Metode ini digunakan untuk mengirim pesan ke sensor node atau BaseStation.
- private static void forwardMsgToNextNode(String msg, FrameIO fio)
Metode ini digunakan untuk meneruskan pesan ke sensor node selanjutnya.
- private static void forwardMsgToPreviousNode(String msg, FrameIO fio)
Metode ini digunakan untuk meneruskan pesan ke sensor node sebelumnya.

Kelas Complex



Gambar 4.16: Perancangan Kelas Complex

Kelas ini digunakan untuk sebagai objek yang merepresentasikan bilangan kompleks. Kelas ini memiliki atribut - atribut sebagai berikut :

- public double re;

Atribut ini digunakan untuk menyimpan bilangan real

- public double im;

Atribut ini digunakan untuk menyimpan bilangan imaginer

Metode - metode yang ada pada ini adalah sebagai berikut:

- public Complex(double r, double i)

Metode ini digunakan sebagai konstruktor dari kelas Complex.

- public Complex tambah(Complex x)

Metode ini digunakan untuk melakukan penjumlahan antar bilangan kompleks.

- public Complex kurang(Complex x)

Metode ini digunakan untuk melakukan pengurangan antar bilangan kompleks.

- public Complex kali(Complex x)

Metode ini digunakan untuk melakukan perkalian antar bilangan kompleks.

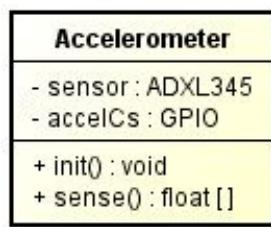
- public double absolute()

Metode ini digunakan untuk mendapat amplitude dari bilangan kompleks.

- public String toString()

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

Kelas Accelerometer



Gambar 4.17: Perancangan Kelas Accelerometer

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

- private ADXL345 sensor; Atribut ini digunakan sebagai objek driver untuk akselerometer ADXL345.
- private GPIO accelCs; Atribut ini digunakan sebagai objek GPIO.

Metode - metode yang ada pada ini adalah sebagai berikut:

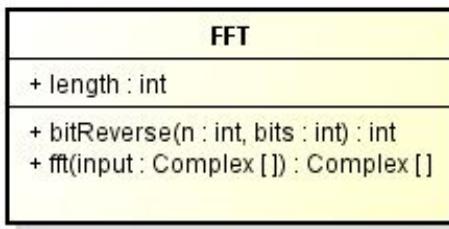
- public void init()

Metode ini digunakan untuk menginisialisasi sensor akselerometer ADXL345.

- public float[] sense()

Metode ini digunakan untuk mendapatkan data pengukuran akselerasi dari ADXL345.

Kelas FFT



Gambar 4.18: Perancangan Kelas FFT

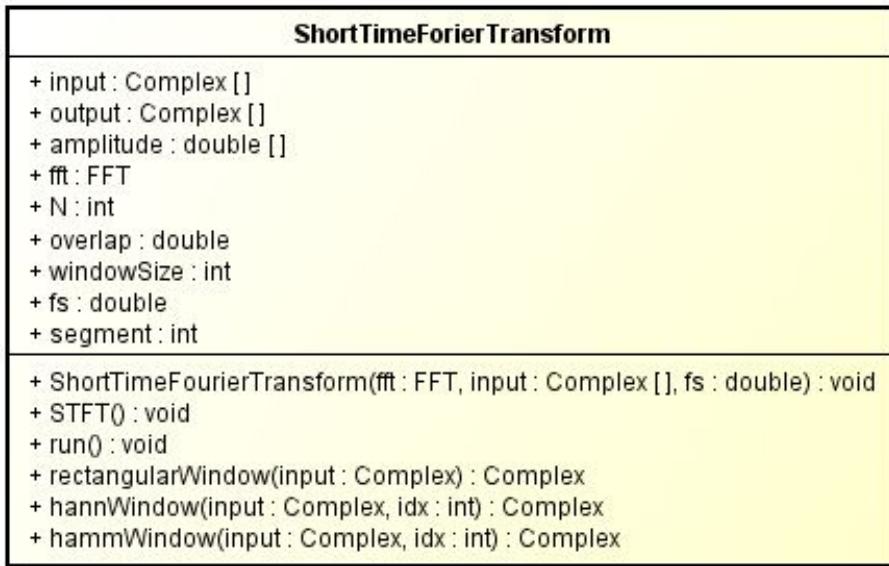
Kelas ini digunakan untuk melakukan komputasi FFT dari input bilangan complex . Kelas ini memiliki atribut sebagai berikut:

- public int length;
Atribut ini menyimpan panjang dari algoritma FFT.

Metode - metode yang ada pada ini adalah sebagai berikut:

- public int bitReverse(int n, int bits)
Metode ini digunakan untuk melakukan bit reverse pada bilangan masukan.
- public Complex [] fft(Complex[] input)
Metode ini digunakan untuk melakukan komputasi FFT pada input yang dimasukkan.

Kelas ShortTimeFourierTransform



Gambar 4.19: Perancangan Kelas ShortTimeFourierTransform

Kelas ini digunakan untuk melakukan komputasi ShortTimeFourierTransform. Kelas ini memiliki atribut - atribut sebagai berikut :

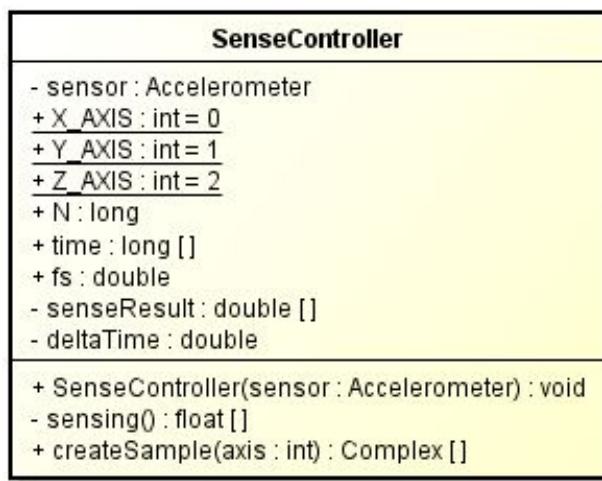
- private Complex [] input;
Atribut ini digunakan untuk menyimpan input data hasil *sensing* akselerometer.

- public Complex [] output;
Atribut ini digunakan untuk menyimpan output hasil STFT dari data input
- public double [] amplitude;
Atribut ini digunakan untuk menyimpan hasil amplitudo dari output
- public FFT fft;
Atribut ini digunakan untuk menyimpan objek kelas FFT yang akan digunakan untuk komputasi STFT.
- public int N;
Atribut ini digunakan untuk menyimpan besar *sample*.
- public double overlap = 0;
Atribut ini menyimpan nilai overlap yang digunakan untuk komputasi algoritma STFT.
- public int windowSize; Atribut ini menyimpan panjang dari window yang digunakan untuk komputasi STFT.
- public double fs;
Atribut ini menyimpan nilai *sampling rate* dari *sensing*.
- public int segment;
Atribut ini menyimpan banyak segment yang digunakan untuk komputasi algoritma STFT.

metode - metode yang ada pada ini adalah sebagai berikut:

- public ShortTimeFourierTransform(FFT fft , Complex [] input, double fs)
Metode ini digunakan sebagai konstruktor dari kelas ShortTimeFourierTransform.
- public void STFT()
Metode ini digunakan untuk melakukan komputasi algoritma STFT pada input hasil *sensing*.
- public void run()
Metode ini digunakan untuk memanggil metode STFT().
- public Complex rectangularWindow(Complex input)
Metode ini digunakan untuk mengaplikasikan *Rectangular Window* pada input hasil *sensing*.
- public Complex hannWindow(Complex input, int idx)
Metode ini digunakan untuk mengaplikasikan *Hanning Window* pada input hasil *sensing*.
- public Complex hammWindow(Complex input, int idx)
Metode ini digunakan untuk mengaplikasikan *Hamming Window* pada input hasil *sensing*.

Kelas SenseController



Gambar 4.20: Perancangan Kelas SenseController

Kelas ini digunakan untuk mengatur pengukuran akselerometer dan membuat *sample* getaran. Kelas ini memiliki atribut - atribut sebagai berikut :

- private Accelerometer sensor;
Atribut ini digunakan sebagai objek Accelerometer
- public static final int X_AXIS = 0;
Atribut ini digunakan sebagai input pada metode sensing().
- public static final int Y_AXIS = 1;
Atribut ini digunakan sebagai input pada metode sensing().
- public static final int Z_AXIS = 2;
Atribut ini digunakan sebagai input pada metode sensing().
- public final int N;
Atribut ini digunakan untuk menyimpan besar *sample*.
- public long [] time;
Atribut ini digunakan untuk menyimpan waktu saat *sensing*.
- public double fs;
Atribut ini digunakan untuk menyimpan *sampling rate*.
- public float [] senseResult;
Atribut ini digunakan untuk menyimpan hasil *sensing* dari akselerometer untuk dijadikan *sample*.
- public double deltaTime;
Atribut ini digunakan untuk menyimpan selisih waktu dari *sensing*.

metode - metode yang ada pada ini adalah sebagai berikut:

- public SenseController(Accelerometer sensor)
Metode ini digunakan sebagai konstruktor dari kelas ShortTimeFourierTransform

- private float[] sensing()
Metode ini digunakan untuk melakukan sensing dengan Accelerometer
- public Complex[] createSample(int axis)
Metode ini digunakan untuk membuat sample pada sumbu berdasarkan input

4.4 Perancangan Masukan dan Keluaran

Aplikasi ini menggunakan *command line interface* untuk menerima input dan menampilkan keluaran dari aplikasi. Setiap masukan untuk aplikasi ini bertipe data *integer*. Aplikasi ini menampilkan 4 pilihan fungsi utama (4.1) dan pengguna dapat memasukkan nilai 1 - 4 sebagai masukan. Berikut penjelasan mengenai tiap masukan yang dapat dimasukkan oleh pengguna.

- Masukan dengan nilai 1 / "Check Online Node" digunakan untuk mengetahui status dari setiap sensor node dan menyamakan waktu - waktu yang ada di sensor node
- Masukan dengan nilai 2 / "Sense" digunakan untuk memberikan perintah sense ke SensorNode dan menerima hasil ekstraksi fitur dari hasil sensing.
- Masukan dengan nilai 3 / "Stop Sensing" digunakan untuk memberikan perintah berhenti kepada SensorNode.
- Masukan dengan nilai 4 / "Exit" digunakan untuk memberhentikan program UserApp dan program pada SensorNode

Apabila pengguna memasukkan perintah selain "Stop sensing" di saat SensorNode sedang sensing maka aplikasi akan menampilkan pesan "IN SENSING STATE.. TO STOP USE OPTION '3'".

4.5 Perancangan Pseudocode Aplikasi

Berdasarkan analisis pada Subbab (3.2) akan dibuat pseudocode untuk aplikasi ini.

4.5.1 Base Station

Base Station berfungsi untuk mengirim dan menerima pesan dari sensor node atau komputer pengguna. Metode untuk menerima pesan dari komputer pengguna dan mengirimkannya ke sensor node dinamakan *runs()* dan metode untuk menerima pesan dari sensor node dan sekaligus mengirim pesan ke komputer pengguna dinamakan *receive()*. Berikut adalah pseudocode dari metode *runs()* dan *receive()*.

- Pseudocode *receive()*

Algorithm 3 Metode *receive()*

```

1: Input : fio
2: Output : -
3: function RECEIVE(fio)
4:   frame ← frame yang diterima oleh basestation
5:   try
6:     content ← mengambil isi dari frame
7:     str ← mengubah content menjadi String
8:     res ← variabel lokal String
9:     if huruf pertama dari str adalah '4' then

```

```

10:    if 5 huruf pertama str sama dengan 'ackAddr' then
11:        atur ulang atribut penanda sudah diterima dan alamat yang diterima
12:    end if
13:    res ← str dari huruf ke 1 sampai ke 5
14:    try
15:        kirim res ke komputer pengguna
16:        catch USARTException
17:    end try
18: end if
19: if huruf pertama str adalah 'A' dan panjang str > 3 dan huruf kedua str adalah 'S' then
20:     if 'ackAddr' masih kosong then
21:         ackAddr ← str
22:     end if
23:     if ackAddr tidak kosong dan ackAddr sama dengan str then
24:         if ackAddr sama dengan str dan penanda data terkirim adalah true then
25:             already ← false // penanda data
26:             ackAddr ← ""
27:         else
28:             if alamat ackAddr tidak terdaftar then
29:                 kirim "ACK" + ackAddr dari huruf kedua ke semua alamat yang terdaftar
30:             else
31:                 kirim "ACK" + ackAddr dari huruf kedua ke alamat ackAddr
32:             end if
33:             sentResult ← ""
34:         end if
35:     end if
36: else
37:     if huruf pertama str adalah '1' then
38:         res ← "_" + str dari huruf kedua + "_"
39:         try
40:             kirim res ke komputer pengguna
41:             catch USARTException
42:         end try
43:     end if
44:     if huruf pertama str adalah '2' then
45:         res ← "_" + str dari huruf kedua + "_"
46:         if sentResult masih kosong dan already == false dan res dari
47:         huruf kedua sampai ke 9 adalah ackAddr dari huruf kedua then
48:             sentResult ← res
49:             try
50:                 kirim res ke komputer pengguna
51:                 catch USARTException
52:             end try
53:             already ← true
54:         end if
55:     end if

```

```

56:    if already == true then
57:        if alamat ackAddr tidak terdaftar then
58:            kirim "ACK2"+ackAddr dari huruf kedua ke semua alamat yang terdaftar
59:        else
60:            kirim "ACK2"+ackAddr dari huruf kedua ke alamat ackAddr
61:        end if
62:    end if
63: end if
64: catch IOException
65: catch InterruptedException
66: end try
67: end function

```

- Pseudocode *runs()*

Algorithm 4 Metode *runs()*

```

1: Input : -
2: Output : -
3: function RUNS
4:   try
5:     inisialisasi transceiver dan radio
6:     buat Thread baru do
7:       function R(u)n
8:         while true do
9:           res ← variabel lokal
10:          input ← variabel lokal
11:          try
12:            input ← membaca input dari komputer
13:            if input == 1 then
14:              curTime ← basestation saat ini
15:              res ← "_BaseStation Online #" + curTime+"_"
16:              kirim ke komputer pengguna
17:              kirim ke alamat sensor node yang tersimpan
18:            else if input == 2 then
19:              kirim "@2" ke alamat sensor node selanjutnya yang tersimpan
20:            else if input == 3 then
21:              kirim "@3" ke alamat sensor node selanjutnya yang tersimpan
22:              atur ulang usart, already, ackAddr, dan sentResult
23:            else if input == 4 then
24:              kirim "@4" ke alamat sensor node selanjutnya yang tersimpan
25:            end if
26:            Thread.sleep selama 50ms
27:            catch USARTException e
28:            catch IOException e
29:            catch InterruptedException e

```

```

30:         end try
31:     end while
32: end function
33: jalankan thread baru
34: buat Thread baru untuk receive()
35: catch Exception
36: end try
37: end function

```

4.5.2 Sensor Node

Pada Sensor node terdapat beberapa metode untuk membuat *sample* dinamakan *createSample()*, melakukan algoritma FFT dinamakan *fft()* (2.4.3), *bit reverse()* (2.4.3) dan melakukan algoritma STFT dinamakan *STFT()*. Berikut pseudocode dari metode - metode tersebut.

- Pseudocode *createSample()*

Algorithm 5 Metode *createSample()*

```

1: Input : axis
2: Output : Complex [ ]
3: function CREATESAMPLE
4:     res ← insialisasi array Complex sebesar N
5:     time ← insialisasi array long sebesar N
6:     while i ← 0 to N , i←i + 1 do
7:         senseResult[i] ← sensing akcelerometer pada sumbu axis x/y/z
8:         res[i] ← buat objek Complex(senseResult[i],0)
9:         time[i] ← Time.currentTimeMillis();
10:        Thread.sleep(16);
11:    end while
12:    fs ← N / ((time[N-1] - time[0])/1000);
13:    deltaTime ← time[1] - time[0];
14:    return res;
15: end function

```

- Pseudocode *bitReverse()*

Algorithm 6 Metode *bitReverse(n,bits)*

```

1: Input : n , bits
2: Output : int
3: function CREATESAMPLE(n,bits)
4:     if n == 0 then
5:         return 0;
6:     end if temp ← n menjadi binaryString
7:     if panjang temp < bits then
8:         k ← bits - panjang temp
9:         while i ← 0 to k , i←i + 1 do
10:            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
16:       res ← res + 2i
17:     end if
18:   end for
19:   return res
20: end function

```

- Pseudocode *fft()*

Algorithm 7 Metode *fft(input)*

```

1: Input : input
2: Output : Complex [ ]
3: function FFT(input)
4:   bits ← Logpanjanginput/log2
5:   finalOrder ← insialisasi array Complex sebesar panjang input
6:   for i ← 0 ; i < panjang input ; i++ do
7:     order ← bitReverse(i,bits)
8:     finalOrder[i] ← input[order]
9:   end for
10:  for N ← 2 ; N <= panjang finalOrder; N ← N × 2 do
11:    for i ← 0; i < panjang finalOrder ; i ← i + N do
12:      for k ← 0 ; k < N/2 ; k ← k + 1 do
13:        first ← finalOrder[i+k]
14:        second ← finalOrder[i+k+(N/2)]
15:        w ← (-2Πk)/N
16:        exp ← new Complex(cos(w),sin(w)).kali(second)
17:        finalOrder[i+k] ← first tambah exp
18:        finalOrder[i+k+(N/2)] ← first kurang exp
19:      end for
20:    end for
21:  end for
22:  return finalOrder
23: end function

```

- Pseudocode *STFT()*

Algorithm 8 Metode *STFT()*

```

1: Input : -
2: Output : -
3: function STFT
4:   idx ← 0
5:   while idx < segment do
6:     windowedInput ← new Complex[panjang fft]

```

```
7:   for i ← 0 ; i < panjang window ; i← i +1 do
8:     inputIdx ← insialisasi variabel lokal
9:     if overlap == 0 then
10:       inputIdx ← i + (idx × panjangwindow – 1)
11:     else
12:       inputIdx ← i + (idx × panjangwindow – 1) * overlap
13:     end if
14:     windowedInput[inputIdx] ← hasil dari window function
15:   end for
16:   for i ← 0 ; i < panjang windowedInput ; i← i +1 do
17:     if windowedInput[i] == null then
18:       windowedInput[i] ← new Complex(0,0)
19:     end if
20:   end for
21:   output ← hasil dari fft(windowedInput)
22:   for i ← 0 ; i < panjang output ; i← i +1 do
23:     amplitude[idx][i] ← absolute dari output[i]
24:   end for
25:   idx ← idx + 1
26: end while
27: end function
```

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri atas implementasi, pengujian, dan masalah yang dihadapi.

5.1 Implementasi

5.1.1 Lingkungan Implementasi

Berikut adalah spesifikasi *laptop* yang digunakan untuk implementasi:

1. *Processor* : Intel (R) Core (TM) i7-2630QM
2. *Memory* : 8192 MB RAM
3. *Operating System* : Windows 10 Home 64-bit

Berikut adalah spesifikasi perangkat lunak yang digunakan untuk implementasi:

1. IDE : version 4.12.0
2. Bahasa Pemrograman : Java
3. Java Library : Java 1.8.0_191

Berikut adalah spesifikasi node sensor yang digunakan untuk implementasi:

1. Nama : Preon32
2. *Processor* : Cortex-M3
3. *Operating System* : PreonVM
4. Penyimpanan Sistem : 64 kByte SRAM
5. Penyimpanan Data : 256 kByte Flash
6. Pita frekuensi : 2400.0 - 2483.5 MHz
7. Jangkauan : 250 meter (luar ruangan) dan 30 meter (dalam ruangan)
8. Sensor-sensor : Sensor suhu, sensor cahaya, sensor kelembaban, sensor tekanan udara, dan sensor getaran.

5.1.2 Hasil Implementasi Kelas

Berdasarkan pembahasan pada bab 4, terdapat kelas - kelas yang diimplementasi di sensor node yaitu kelas *Accelerometer*, kelas *Complex*, kelas *FFT*, kelas *SenseController*, kelas *ShortTimeFourierTransform*, kelas *SNManager* dan kelas *BSManager*. Kelas - kelas yang diimplementasi di sensor node akan diatur oleh kelas *SNManager* atau *BSManager* jika sensor node tersebut adalah *base station*. Selain dari kelas - kelas tersebut, terdapat kelas yang diimplementasi di komputer sebagai penghubung antara *base station* dengan komputer.

Kelas Accelerometer

Pada kelas ini terdapat metode *init* yang berfungsi untuk meinisialisasi objek ADXL345, GPIO, dan NativeSPI agar dapat mengatur sensor akselerometer pada sensor node. Setelah metode *init* terdapat metode *sense* dengan keluaran berupa *array*. Metode init berfungsi untuk memberikan perintah kepada akselerometer untuk melakukan pengukuran dan mengonversi pengukuran ke dalam satuan gravitasi. Kode program kelas ini dapat dilihat pada lampiran A bagian A.5.

Kelas Complex

Kelas ini merepresentasikan bilangan kompleks. Pada kelas ini terdapat metode *tambah* untuk melakukan operasi tambah pada bilangan kompleks, *kurang* untuk melakukan operasi pengurangan pada bilangan kompleks, *kali* untuk melakukan operasi perkalian pada bilangan kompleks, *bagi* untuk melakukan operasi pembagian pada bilangan kompleks dan *absolute* untuk memutlakkan bilangan kompleks. Selain itu terdapat metode *toString* untuk menjadikan bilangan kompleks sebuah *string*. Kode program kelas ini dapat dilihat pada lampiran A bagian A.6.

Kelas FFT

Kelas ini merupakan implementasi dari algoritma *Cooley-Tukey FFT Algorithm*. Pada kelas ini terdapat metode *bitReverse* yang berfungsi untuk melakukan *in bit reverse order* pada input FFT. Metode *bitReverse* diimplementasi berdasarkan pseudocode 4.5.2. Listing 5.1 menunjukkan hasil implementasi metode *bitReverse*.

Kode 5.1: Metode *bitReverse()*

```
public int bitReverse(int n, int bits) {
    if (n == 0) {
        return 0;
    }

    String temp = Integer.toBinaryString(n);
    if (temp.length() < bits) {
        int k = bits - temp.length();
        for (int i = 0; i < k; i++) {
            temp = "0" + temp;
        }
    }

    int res = 0;
    for (int i = temp.length() - 1; i >= 0; i--) {
        if (temp.charAt(i) == '1') {
            res += Math.pow(2, i);
        }
    }

    return res;
}
```

Selain itu terdapat metode *FFT* yang berfungsi untuk melakukan operasi FFT. Metode *fft* diimplementasi berdasarkan pseudocode 4.5.2. Listing 5.2 menunjukkan hasil implementasi dari metode *fft*.

Kode 5.2: Metode *fft()*

```
public Complex[] fft(Complex[] input) {
    int bits = (int) (Math.log(input.length) / Math.log(2));
    Complex[] finalOrder = new Complex[input.length];
    for (int i = 0; i < input.length; i++) {
        int order = bitReverse(i, bits);
        finalOrder[i] = input[order];
    }

    for (int N = 2; N <= finalOrder.length; N = N * 2) {
        for (int i = 0; i < finalOrder.length; i += N) {
            for (int k = 0; k < N / 2; k++) {

                Complex first = finalOrder[i + k];
                Complex second = finalOrder[i + k + (N / 2)];

                double w = (-2 * Math.PI * k) / (double) N;
                Complex exp = (new Complex(Math.cos(w), Math.sin(w)).kali(second));

                finalOrder[i + k] = first.tambah(exp);
                finalOrder[i + k + (N / 2)] = first.kurang(exp);
            }
        }
    }
}
```

```

    }
    return finalOrder;
}

```

Kode program kelas ini secara lengkap dapat dilihat pada lampiran A bagian A.7.

Kelas SenseController

Kelas ini memiliki metode *sensing* untuk akselerometer melakukan pengukuran dan atribut X_AXIS, Y_AXIS, dan Z_AXIS untuk menentukan sumbu yang akan diukur akselerometer dan dijadikan *sample*. Pembuatan *sample* diatur oleh metode *createSample*. Metode *createSample* diimplementasi berdasarkan pseudocode 4.5.2. Listing 5.3 menunjukkan hasil implementasi dari metode *createSample*.

Kode 5.3: Metode *createSample()*

```

public Complex[] createSample(int axis) throws InterruptedException {
    Complex[] res = new Complex[N];
    time = new long[N];
    for (int i = 0; i < N; i++) {
        senseResult[i] = this.sensing()[axis];
        res[i] = new Complex(senseResult[i], 0);
        time[i] = Time.currentTimeMillis();
        Thread.sleep(16);
    }
    this.fs = N / ((time[N-1] - time[0])/1000); //sampling rate
    this.deltaTime = time[1] - time[0];
    return res;
}

```

Pada metode *createSample* setiap waktu pengukuran akselerometer pada atribut *time* akan disimpan, hasil *sensing* dikonversi menjadi objek *Complex*, dan *sampling frequency* dihitung dan disimpan pada atribut *fs*. Kode program kelas ini dapat dilihat pada lampiran A bagian A.8.

Kelas ShortTimeFourierTransform

Kelas ini memiliki merupakan implementasi dari algoritma *Short Time Fourier Transform*. Kelas ini memiliki atribut *N* untuk panjang dari STFT, *overlap* untuk nilai *overlap*, *windowSize* untuk panjang *window*, dan *segment* untuk panjang *segment* dari STFT. Untuk melakukan operasi STFT, kelasi ini metode *STFT* yang diimplementasi berdasarkan pseudocode 4.5.2. Listing 5.4 menunjukkan hasil implementasi dari metode *STFT*.

Kode 5.4: Metode *STFT()*

```

public void STFT() {
    int idx = 0;
    while (idx < segment) {
        Complex[] windowedInput = new Complex[fft.length]; // panjang arr window sama dengan fft
        for (int i = 0; i < windowSize; i++) {
            int inputIdx;
            if (overlap == 0) {
                inputIdx = i + (idx * (windowSize - 1));
            } else {
                inputIdx = (int) (i + (idx * (windowSize - 1)) * overlap));
            }
            windowedInput[inputIdx] = rectangularWindow(input[inputIdx]);
            windowedInput[inputIdx] = hannWindow(input[inputIdx], i);
            windowedInput[inputIdx] = hammWindow(input[inputIdx], i);
        }
        // zero padded
        for (int i = 0; i < windowedInput.length; i++) {
            if (windowedInput[i] == null) {
                windowedInput[i] = new Complex(0, 0);
            }
        }
        output = this.fft.fft(windowedInput);
        System.out.println("selesai");
        for (int i = 0; i < output.length; i++) {
            amplitude[idx][i] = output[i].absolute();
        }
        idx++;
    }
}

```

Pada kelas ini terdapat pula metode *rectangularWindow*, *hannWindow*, dan *hammWindow* untuk menerapkan *window function* pada input data hasil *sensing*. Kode program kelas ini dapat dilihat pada lampiran A bagian A.9.

Kelas SNManager

Kelas ini merupakan *main class* pada sensor node. Saat sensor node dijalankan, kelas ini akan menginisialisasi atribut - atribut dan memanggil metode *runs*. Metode *runs* akan menginisialisasi *tranceiver*, *RadioDriver*, dan *FrameIO*. *Tranceiver* dan *RadioDriver* berfungsi untuk komunikasi antar sensor node dan *FrameIO* untuk pengiriman/penerimaan pesan.

Setelah itu metode *receive* dijalankan. Pada kelas ini terdapat 2 metode pengiriman yaitu metode *forwardMsgToNextNode* dan *forwardMsgToPreviousNode*. Metode *forwardMsgToNextNode* akan meneruskan pesan yang diterima oleh sensor node ke setiap alamat sensor node selanjutnya yang disimpan pada atribut *nextNode*. Metode *forwardMsgToPreviousNode* akan meneruskan pesan yang diterima oleh sensor node ke setiap alamat sensor node yang sebelumnya yang disimpan pada atribut *previousNode*.

Saat sensor node menerima pesan "@1", metode *receive* akan menyamakan waktu sensor node dengan waktu dari pesan, mengirimkan status "ONLINE" ke *previous node* dengan metode *forwardMsgToPreviousNode* dan mengirimkan pesan "@1" dan waktu dari pesan ke sensor node selanjutnya dengan metode *forwardMsgToNextNode*. Jika sensor node menerima pesan "@2" maka sensor node akan memanggil metode *forwardMsgToNextNode* dan mulai *thread* baru untuk memulai perintah *sensing*. Hasil dari *sensing* yang sudah diekstraksi fitur langsung dikirimkan dengan metode *forwardMsgToPreviousNode* dan diakhiri dengan pesan "4done". Saat sensor node menerima pesan "@3", metode *receive* akan meneruskan pesan "@3" dengan metode *forwardMsgToNextNode* dan menghentikan proses *sensing* dan ekstraksi fitur. Setelah itu jika sensor node menerima pesan "@4", sensor node akan meneruskan pesan "@4" dengan metode *forwardMsgToNextNode* dan menghentikan program. Selain pesan dengan awalan "@" akan diteruskan dengan metode *forwardMsgToPreviousNode*. Kode program kelas ini dapat dilihat pada lampiran A bagian A.10.

Kelas BSManager

Kelas ini merupakan *main class* pada sensor node yang menjadi *base station*. Saat program dijalankan, kelas ini akan menginisialisasi USART yang berfungsi untuk menerima masukan dan mengirim pesan dari komputer pengguna. Setelah itu kelas ini akan mulai *thread* baru dan memanggil metode *runs*. Metode *runs* menginisialisasi *transceiver*, *RadioDriver*, dan *FrameIO* dan mulai *thread* baru untuk menerima input dari komputer. Saat *base station* menerima input, *base station* akan mengirimkan pesan dengan metode *send* ke alamat sensor node yang disimpan di atribut *connectedNodeAddr*. Saat *base station* menerima pesan dari sensor node dengan metode *receive*, *base station* mengubah format pesan dan mengirimkan pesan ke komputer dengan *USART*. Metode *receive* diimplementasi sesuai dengan pseudocode 4.5.1. Listing 5.5 menunjukkan hasil implementasi dari metode *receive*.

Kode 5.5: Metode *receive()*

```
public static void receive(final FrameIO fio) {
    while (true) {
        Frame frame = new Frame();
        try {
            fio.receive(frame);
            byte[] content = frame.getPayload();
            String str = new String(content, 0, content.length);
            String res;
            // done
            if (str.trim().charAt(0) == '4') {
                if (str.substring(5).equals(ackAddr)) {
                    already = false;
                    ackAddr = "";
                }
                res = str.substring(1, 5);
                try {
                    out.write(res.getBytes());
                    usart.flush();
                } catch (USARTException e) {
                }
            }
            // ASensor#
            if (str.charAt(0) == 'A' && str.length() > 3 && str.charAt(1) == 'S') {
                // first set ackAddr
                if (ackAddr.equals("")) {
                    ackAddr = str;
                }
            }
        }
    }
}
```

```

        }
        // send ACKSensor#
        if (ackAddr.length() != 0 && ackAddr.equals(str)) {
            if (ackAddr.equals(str) && already) {
                already = false;
                ackAddr = "";
            } else {
                if (!nodeAddrMap.containsKey(ackAddr)) {
                    for (int i = 0; i < connectedNodeAddr.length; i++) {
                        send("ACK" + ackAddr.substring(1), currAddr, connectedNodeAddr[i], fio);
                    }
                } else {
                    send("ACK" + ackAddr.substring(1), currAddr, nodeAddrMap.get(ackAddr), fio);
                }
                sentResult = "";
            }
        }
    } else {
        if (str.charAt(0) == '1') {
            res = "-" + str.substring(1) + "_";
            try {
                out.write(res.getBytes());
                usart.flush();
            } catch (USARTException e) {
            }
        }
        // receive result
        if (str.charAt(0) == '2') {
            res = "-" + str.substring(1) + "-";
            if (sentResult.equals("") && !already && res.substring(1, 8).equals(ackAddr.substring(1))) {
                sentResult = res;
                try {
                    out.write(res.getBytes());
                    usart.flush();
                } catch (USARTException e) {
                }
            }
            already = true;
        }
    }

    // already send ACK2Sensor#;
    if (already) {
        if (!nodeAddrMap.containsKey(ackAddr)) {
            for (int i = 0; i < connectedNodeAddr.length; i++) {
                send("ACK2" + ackAddr.substring(1), currAddr, connectedNodeAddr[i], fio);
                Thread.sleep(10);
            }
        } else {
            send("ACK2" + ackAddr.substring(1), currAddr, nodeAddrMap.get(ackAddr), fio);
        }
    }
}

} catch (IOException e) {
} catch (InterruptedException e) {
}
}

```

Kode program kelas ini dapat dilihat pada lampiran A bagian A.11.

Selain dari kelas - kelas yang diimplementasikan di sensor node, terdapat kelas *Plotting*, *Visualizing*, *Spectrogram*, dan *Tester* yang diimplementasikan di komputer pengguna.

Kelas Plotting

Kelas ini digunakan untuk menyimpan hasil dari *sensing* ke dalam atribut - atribut kelas ini. Terdapat metode *plotRealTime* yang dipanggil saat kelas *Visualizing* dijalankan. Kode program kelas ini dapat dilihat pada lampiran A bagian A.1.

Kelas Visualizing

Kelas ini berfungsi untuk mevisualisasikan hasil dari *sensing*. Saat kelas ini dijalankan, kelas ini menginisialisasi atribut - atribut dan membuat *ChartPanel* baru untuk menampilkan plot. Terdapat metode *showPlot* yang menginisialisasi *JFrame* untuk menampilkan *ChartPanel* pada layar dan *inner class DataGenerator* yang memperbaharui data pada plot dengan metode *addSenseObservation*. Kode program kelas ini dapat dilihat pada lampiran A bagian A.11.

Kelas Spectrogram

Kelas ini berfungsi untuk menampilkan grafik hasil dari ekstraksi fitur data *sensing*. Pada kelas ini terdapat atribut *resAmplitude* dan *resTime* untuk menyimpan hasil ekstraksi fitur. Pada kelas

ini terdapat metode *createDataset* yang berfungsi untuk membuat *dataset* dari hasil ekstraksi fitur dan metode *createChart* untuk menampilkan hasil grafik spectrogram dari ekstraksi fitur. Kode program kelas ini dapat dilihat pada lampiran A bagian A.4.

Kelas Tester

Kelas ini merupakan *main class* pada komputer pengguna. Kelas ini berfungsi sebagai penghubung untuk menginisialisasi komunikasi dan memberikan perintah kepada *base station*. Sebelum kelas ini dijalankan *base station* harus sudah terhubung dengan komputer melalui USB port.

Saat kelas ini dijalankan, penghubung komputer dan *base station* diinisialisasi dengan metode *context_set* dan *base station* melakukan sinkronisasi waktu dengan komputer dengan metode *time_synchronize*. Setelah itu dibuat objek *Preon32Helper* untuk menjalankan module pada *base station* dan mengatur koneksi komunikasi dengan *base station*.

Setelah itu kelas ini menampilkan perintah untuk memasukkan jumlah sensor node. Setelah pengguna memasukkan jumlah sensor node, aplikasi menampilkan 4 pilihan yang dapat dipilih oleh pengguna yaitu "Check Online Node", "Sense", "Stop Sensing", dan "Exit". Jika pengguna memasukkan angka "1", maka aplikasi akan menampilkan node - node yang berstatus "ONLINE" beserta waktu pada node - node tersebut. Jika pengguna memasukkan angka "2" maka aplikasi akan menampilkan status "SENSING" dan menampilkan plot setiap sensor node. Saat pada status "SENSING" aplikasi akan menyimpan semua hasil sensing dengan metode *saveSenseResult*. Jika pengguna memasukkan angka "3", maka saat aplikasi dalam status sensing akan berhenti. Jika pengguna memasukkan angka "4", maka pengguna aplikasi akan berhenti. Kode program kelas ini dapat dilihat pada lampiran A bagian A.2.

5.1.3 Hasil Implementasi Antar Muka Visualisasi Hasil Sense

Berikut contoh hasil dari implementasi antar muka visualisasi hasil sense dengan menggunakan *library* dari *JFreeChart*.

```

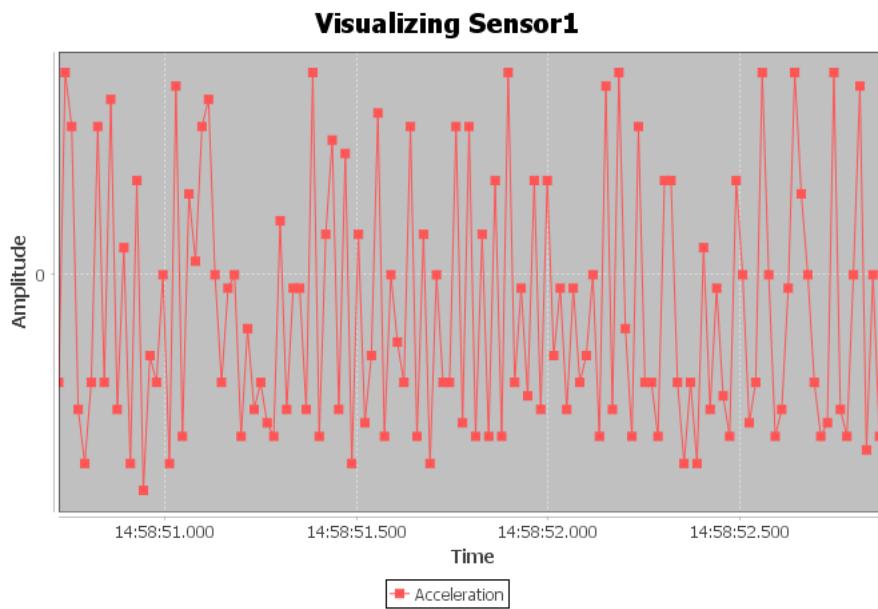
MINGW64;/c/Users/lfunk/Desktop/DEMO
$ java -jar Tester.jar

Please insert total Sensor Node (not including basestation) :
5
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :

```

Gambar 5.1: Tampilan awal aplikasi

Pada tampilan awal (Gambar 5.1) pengguna perlu memilih option *Sense*. Tampilan visualisasi hasil sense akan muncul setelah sensor node mengirimkan hasil sense.

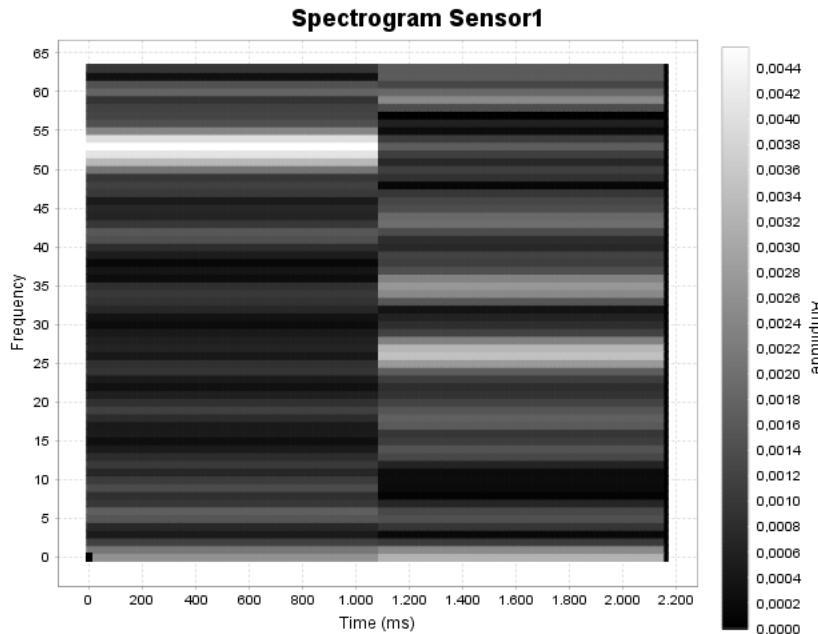


Gambar 5.2: Grafik hasil implementasi visualisasi hasil sense

Gambar 5.2 menunjukkan grafik hasil *sensing* dari Sensor1. Grafik ini menunjukkan amplitudo dari akselerasi yang terekam oleh Sensor1.

5.1.4 Hasil Implementasi Antar Muka Spectrogram

Berikut contoh hasil dari implementasi antar muka spectrogram hasil ekstraksi fitur dari *sensing* Sensor1 dengan menggunakan *library* dari *JFreeChart*. Seperti antar muka visualisasi hasil sense, pada tampilan awal (Gambar 5.1) pengguna perlu memilih option *Sense*. Tampilan spectrogram akan muncul setelah sensor node mengirimkan hasil ekstraksi fitur.



Gambar 5.3: Grafik hasil implementasi visualisasi hasil sense

Gambar 5.3 menunjukkan grafik hasil ekstraksi fitur dari hasil *sensing* Sensor1. Dari Grafik ini dapat terlihat frekuensi yang dominan dan frekuensi lain yang terekam dari *sensing* Sensor1

berdasarkan besar amplitudo dari frekuensi.

5.2 Pengujian

Pengujian dilakukan dengan menggunakan dua buah metode yaitu pengujian fungsional dan pengujian eksperimental.

5.2.1 Pengujian Fungsional

Pengujian dilakukan dengan menjalankan aplikasi di *Command Line Interface*. Tampilan utama setelah pengguna memasukkan jumlah sensor node seperti ada pada Gambar 5.4.



```
lfunk@lfunk-VAIO MINGW64 ~/Desktop/DEMO
$ java -jar Tester.jar

Please insert total Sensor Node (not including basestation) :
5
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :
```

Gambar 5.4: Tampilan awal setelah memasukkan jumlah sensor node

Fitur - fitur aplikasi yang terdapat pada aplikasi adalah sebagai berikut:

- Check Online node

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 5.5.

```

MINGW64:/c/Users/lfunk/Desktop/DEMO
lfunk&LFunk-VAIO MINGW64 ~/Desktop/DEMO
$ java -jar Tester.jar

Please insert total Sensor Node (not including basestation) :
5
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :

ONLINE NODE :
> BaseStation ONLINE Time: 08:20:58.342
> Sensor1 ONLINE Time: 08:20:58.342
> Sensor2 ONLINE Time: 08:20:58.342
> Sensor3 ONLINE Time: 08:20:58.342
> Sensor4 ONLINE Time: 08:20:58.342
> Sensor5 ONLINE Time: 08:20:58.342
- CHECKING DONE -

OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :

```

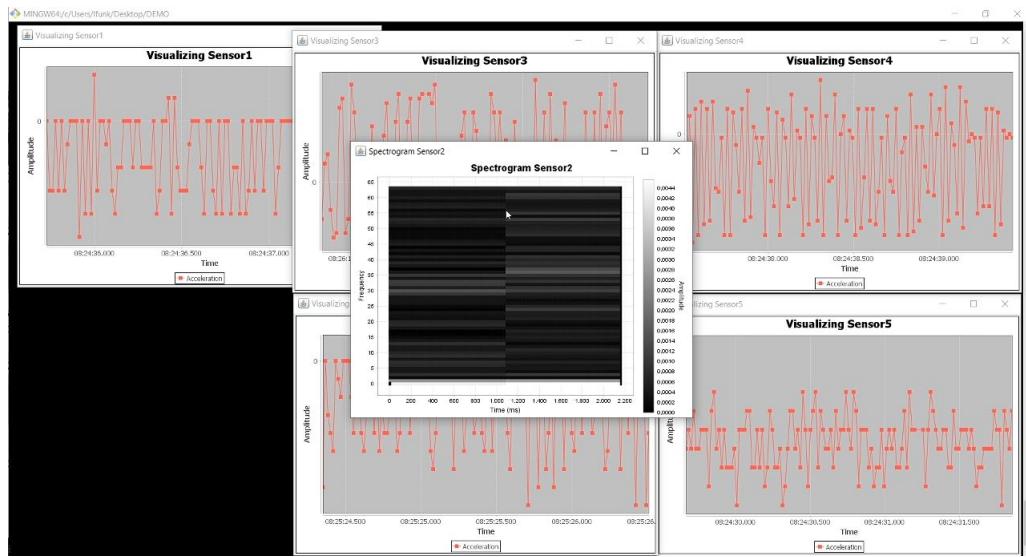
Gambar 5.5: Pengujian "Check Online node"

- Sense

Fitur ini berfungsi untuk memberikan perintah kepada sensor node - sensor node yang sedang menyalakan untuk memulai *sensing*. Saat pengguna memasukkan angka "2", aplikasi akan menampilkan pula grafik hasil *sensing* dan grafik spectrogram hasil ekstraksi fitur yang akan terus diperbaharui. Tampilan awal awal fungsi ini dijalankan seperti pada Gambar 5.6 dan tampilan setelah grafik diperbaharui seperti pada Gambar 5.7



Gambar 5.6: Tampilan awal setelah menjalankan fungsi "Sense"



Gambar 5.7: Tampilan setelah mendapatkan hasil ekstraksi fitur dari sensor node

Saat fungsi Sense berjalan, aplikasi tidak akan menerima input selain masukan angka "3" untuk memberhentikan *sensing*. Tampilan jika pengguna memasukkan input lain dapat dilihat pada Gambar 5.8

```

OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :
2
SENSING..
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :
E
IN SENSING STATE.. TO STOP USE OPTION '3'
WRONG INPUT
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :
2
IN SENSING STATE.. TO STOP USE OPTION '3'
WRONG INPUT
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :
4
IN SENSING STATE.. TO STOP USE OPTION '3'
WRONG INPUT
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :

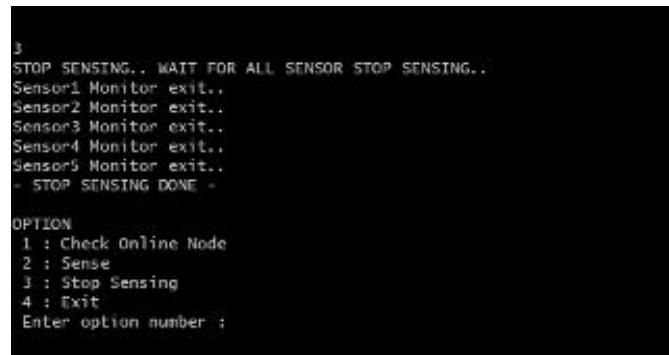
```

Gambar 5.8: Pengujian input selain angka "3" saat "Sense"

- Stop sensing

Fitur ini berfungsi untuk menghentikan sensor node yang sedang *sensing* dan menutup grafik - grafik yang ditampilkan. Pengguna harus memasukkan angka "3" saat sensor node sedang

sensing untuk menjalankan fungsi ini. Tampilan setelah fungsi ini dijalankan seperti pada Gambar 5.9



```

3
STOP SENSING.. WAIT FOR ALL SENSOR STOP SENSING..
Sensor1 Monitor exit..
Sensor2 Monitor exit..
Sensor3 Monitor exit..
Sensor4 Monitor exit..
Sensor5 Monitor exit..
- STOP SENSING DONE -

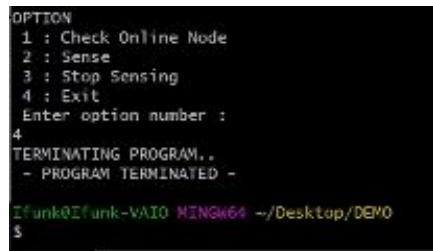
OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :

```

Gambar 5.9: Pengujian "Stop sensing"

- Exit

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 5.10



```

OPTION
1 : Check Online Node
2 : Sense
3 : Stop Sensing
4 : Exit
Enter option number :
4
TERMINATING PROGRAM..
- PROGRAM TERMINATED -

Ifunk0@funk-VAIO MINGW64 ~/Desktop/DEMO
$ 

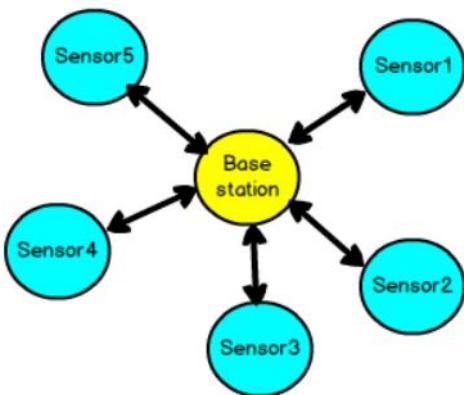
```

Gambar 5.10: Pengujian "Exit"

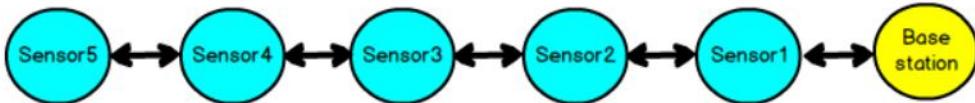
5.2.2 Pengujian Eksperimental

Pengujian Eksperimental ini dilakukan di atap rumah penguji di jalan Pasirkaliki Bandung dan di jalan raya Pasirkaliki Bandung. Sensor node yang digunakan untuk 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 melihat kemampuan sensor node mengekstraksi fitur di atap rumah dan di jalan raya dengan topologi *star* dan topologi *tree*. Pengujian ini mengukur sumbu *x* dari akselerometer untuk *sample* sebesar 128 dalam 2 detik, *window* sebesar 64, *overlap* sebesar 0%, dan *sampling rate* sebesar 64*Sample/second*. Pengujian ini menggunakan *Window function Rectangle*, *Hanning*, dan *Hamming*. Besar *sample* dan *window function* tersebut dipilih untuk melihat kemampuan aplikasi ini dalam melakukan ekstraksi fitur di sensor node dengan besar *sample* dan *window* tersebut. Dengan *sampling rate* setengah dari besar *sample* maka setiap frekuensi getaran yang didapat adalah setengah dari frekuensi getaran yang didapat.

Topologi yang digunakan untuk pengujian ini adalah topologi *star* (Gambar 5.11) dan *tree* (Gambar 5.12). Topologi *star* dan *tree* dipilih karena kedua topologi tersebut sudah mencakup 2 jalur komunikasi WSN (*single hop* dan *multi hop*).



Gambar 5.11: Topologi star yang digunakan



Gambar 5.12: Topologi tree yang digunakan

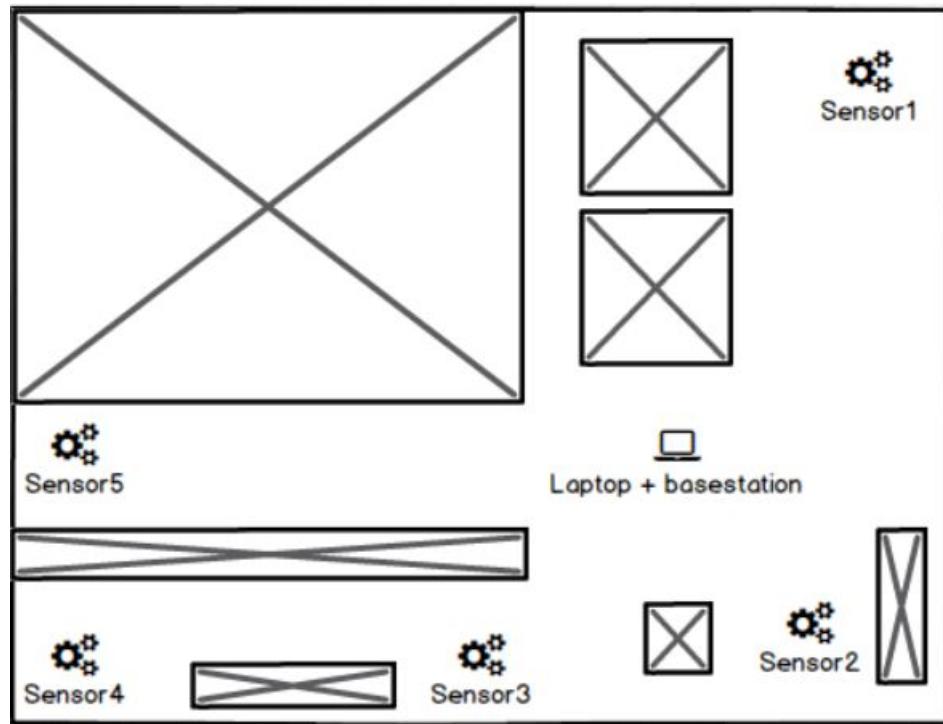
Skenario - skenario dalam pengujian ini berfokus dalam menguji setiap topologi dan *window function* yang digunakan. Setiap skenario bertujuan melihat kemampuan ekstraksi fitur setiap sensor node. Langkah pengujian pada setiap skenario dalam pengujian ini adalah sebagai berikut:

1. Sensor node diatur dengan topologi dan *window function* tertentu.
2. Sensor node diletakkan ditempat yang berjauhan dan sama pada setiap skenario.
3. Mengecek status *online* setiap sensor node lewat komputer.
4. Memberi perintah *sense* ke setiap sensor node lewat komputer.
5. Hasil tampilan dari hasil ekstraksi fitur di simpan

Hasil dari pengujian ini berupa grafik amplitudo akselerasi dari hasil *sensing* dan grafik spectrogram hasil ekstraksi fitur dari setiap sensor node.

Pengujian di Atap Rumah

Pada pengujian ini sensor diletakkan seperti pada Gambar 5.13. Setiap sensor node diberi jarak satu sama lain untuk menunjukkan komunikasi setiap sensor node secara nirkabel dan untuk melihat hasil ekstraksi di lokasi - lokasi yang berbeda.



Gambar 5.13: Denah peletakan sensor node

Pengujian dilakukan saat pagi ke siang hari dengan cuaca berawan. Hasil grafik dari pengujian dapat di lihat pada lampiran B.

- Topologi Star

Tabel 5.1: Tabel hasil getaran yang tertangkap di Atap dengan Topologi Star

Id Sensor	frekuensi dominan saat 0ms (Hz)	frekuensi dominan saat 1000ms (Hz)	window function
Sensor1	0	0.5	Rectangle
Sensor2	0	0.5	Rectangle
Sensor3	9	9	Rectangle
Sensor4	0	0.5	Rectangle
Sensor5	0	0.5	Rectangle
Sensor1	24.5	27.5	Hanning
Sensor2	17.5	17.5	Hanning
Sensor3	2	0	Hanning
Sensor4	0	0.5	Hanning
Sensor5	0	0	Hanning
Sensor1	15	11	Hamming
Sensor2	0	0.5	Hamming
Sensor3	0	0	Hamming
Sensor4	0	0	Hamming
Sensor5	0	0	Hamming

- Topologi Tree

Berikut tabel dari hasil getaran yang terekam oleh sensor node di atap rumah dengan topologi tree.

Tabel 5.2: Tabel hasil getaran yang tertangkap di Atap dengan Topologi Tree

Id Sensor	frekuensi dominan saat 0ms (<i>Hz</i>)	frekuensi dominan saat 1000ms (<i>Hz</i>)	<i>window function</i>
Sensor1	0	0	<i>Rectangle</i>
Sensor2	0	0	<i>Rectangle</i>
Sensor3	1	0	<i>Rectangle</i>
Sensor4	0	0	<i>Rectangle</i>
Sensor5	0	0	<i>Rectangle</i>
Sensor1	0	0	<i>Hanning</i>
Sensor2	0	0	<i>Hanning</i>
Sensor3	31.5	31.5	<i>Hanning</i>
Sensor4	0	0	<i>Hanning</i>
Sensor5	0	0	<i>Hanning</i>
Sensor1	0	0	<i>Hamming</i>
Sensor2	0	0	<i>Hamming</i>
Sensor3	22	10	<i>Hamming</i>
Sensor4	0	0	<i>Hamming</i>
Sensor5	0	0	<i>Hamming</i>

Dari tabel 5.1 dan 5.2 terlihat bahwa sensor node berhasil merekam getaran dan frekuensi dari atap rumah dengan topologi tree. Hasil frekuensi dari sensor yang berbeda - beda disebabkan karena perbedaan tempat dan waktu saat melakukan *sensing* sehingga getaran yang ditangkap berbeda - beda. Getaran terbesar yang ditangkap oleh sensor di Atap adalah sebesar 31.5Hz dan getaran terendah adalah 0Hz .

Pengujian di Jalan Raya

Pada pengujian ini sensor node diletakkan di pinggir jalan untuk merekam getaran - getaran saat ada kendaraan yang melewatkannya. Hasil dari pengujian dapat di lihat pada lampiran C.

- Topologi Star

Pengujian dilakukan dengan kondisi jalan raya cukup sepi. Berikut tabel dari hasil getaran yang terekam oleh sensor node di jalan raya dengan topologi *star*.

Tabel 5.3: Tabel hasil getaran yang tertangkap di Jalan Raya dengan Topologi Star

Id Sensor	frekuensi dominan saat 0ms (Hz)	frekuensi dominan saat 1000ms (Hz)	window function
Sensor1	0	0	Rectangle
Sensor2	0	9	Rectangle
Sensor3	0	0	Rectangle
Sensor4	0	26	Rectangle
Sensor5	0	0	Rectangle
Sensor1	0	18	Hanning
Sensor2	0	0	Hanning
Sensor3	15	11.5	Hanning
Sensor4	0	0	Hanning
Sensor5	0	0	Hanning
Sensor1	26.5	13	Hamming
Sensor2	0	0	Hamming
Sensor3	22	10	Hamming
Sensor4	0	0	Hamming
Sensor5	0	0	Hamming

- Topologi Tree

Tabel 5.4: Tabel hasil getaran yang tertangkap di Jalan Raya dengan Topologi Tree

Id Sensor	frekuensi dominan saat 0ms (Hz)	frekuensi dominan saat 1000ms (Hz)	window function
Sensor1	0	0	Rectangle
Sensor2	30.5	31.5	Rectangle
Sensor3	0	0	Rectangle
Sensor4	2	7.5	Rectangle
Sensor5	0	0	Rectangle
Sensor1	0	0	Hanning
Sensor2	0	0	Hanning
Sensor3	0	0	Hanning
Sensor4	19	10	Hanning
Sensor5	0	0	Hanning
Sensor1	0	0	Hamming
Sensor2	10.5	26.5	Hamming
Sensor3	0	11	Hamming
Sensor4	7	0	Hamming
Sensor5	0	0	Hamming

Dari tabel 5.3 dan 5.4 terlihat bahwa sensor node berhasil merekam getaran dan frekuensi dari jalan raya dengan topologi star dan tree. Hasil frekuensi dari sensor yang berbeda - beda disebabkan karena perbedaan tempat dan waktu saat melakukan *sensing* sehingga getaran yang ditangkap berbeda - beda. Getaran terbesar yang terekam di jalan raya adalah 31.5Hz dan getaran terendah adalah 0Hz .

Kesimpulan dari Pengujian

Dari hasil eksperimen, aplikasi berhasil melakukan ekstraksi fitur di atap dan jalan raya dengan topologi *star* dan *tree*. Getaran terendah yang ditangkap sensor di atap dan jalan raya adalah getaran dengan frekuensi $0Hz$ dan getaran terbesarnya dengan frekuensi $31.5Hz$. Dari pengujian ini terlihat bahwa hasil ekstraksi fitur tidak terpengaruh oleh topologi dari WSN yang dipakai namun dari tempat dan letak sensor node melakukan *sensing*.

5.2.3 Masalah dalam Pengujian

Berikut masalah - masalah yang dihadapi saat penguji melakukan pengujian:

1. Sensor node yang digunakan untuk pengujian terbatas, sehingga sensor node harus dipakai secara bergantian dengan mahasiswa lain yang mengambil topik skripsi yang berkaitan dengan sensor node.
2. *Memory* minim pada sensor node yang menghambat proses ekstraksi fitur yang membutuhkan *memory* yang besar untuk menyimpan *sample* dan hasil dari *ekstraksi fitur*
3. Saat pengujian sensor node dapat mati secara tiba - tiba, karena sumber *power* dari sensor node (baterai) dapat habis saat sensor node melakukan *sensing*
4. Cuaca hujan yang menghambat pengujian, karena pengujian dilakukan di ruangan terbuka.
5. Pengujian dilakukan saat adanya pandemi virus *Covid19*, sehingga adanya keterbatasan untuk keluar rumah dan penggunaan sensor node secara bersama dengan mahasiswa lain yang memilih topik skripsi yang berkaitan demi mematuhi peraturan PSBB (Pembatasan Sosial Berskala Besar) dari pemerintah.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

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

1. Aplikasi ekstraksi fitur dengan algoritma STFT pada *Wireless Sensor Network* telah berhasil dibangun.
2. Berdasarkan pengujian, aplikasi ini berhasil melakukan ekstraksi fitur pada WSN bertopologi *star* dan *tree* dengan jumlah sensor node sebanyak 6.

6.2 Saran

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

1. Aplikasi ini hanya menggunakan algoritma *Cooley-Tukey FFT* dalam algoritma *Short Time Fourier Transform*. Terdapat algoritma *Fast Fourier Transform* lainnya yang mungkin lebih efisien dibandingkan algoritma *Cooley-Tukey* yang dapat diimplementasikan dalam algoritma STFT.
2. Aplikasi ini baru menggunakan 3 *window function* pada algoritma STFT, yaitu *Rectangle*, *Hanning*, dan *Hamming*. Terdapat *window function* lainnya yang dapat diterapkan pada algoritma STFT yang mungkin dapat menghasilkan hasil ekstraksi fitur lebih baik.

DAFTAR REFERENSI

- [1] Kazem Sohraby, T. Z., Daniel Minoli (2007) *Wireless Sensor Networks: Technology, Protocols, and Applications*. A John Wiley and Sons, Ltd.
- [2] Holger Karl, A. W. (2005) *Protocols and Architectures For Wireless Sensor Networks*. A John Wiley and Sons, Ltd.
- [3] Divya Sharma, K. S., Sandeep Verma (2013) Network topologies in wireless sensor networks: A review. *International Journal of Electronics and Communication Technology*, **4**, 93–97.
- [4] Abbas Jamalipour, J. Z. (2009) *Wireless Sensor Networks A Networking Perspective*. A John Wiley and Sons, Ltd.
- [5] Dargie, W. dan Poellabauer, C. (2010) *Fundamentals Of Wireless Sensor Network Theory And Practice*. A John Wiley and Sons, Ltd.
- [6] Pavel Ripka, A. T. (2007) *Moderns Sensors Handbook*. ISTE ,Ltd.
- [7] Isabelle Guyon, M. N., Steve Gunn (2006) *Feature Extraction: Foundations and Applications*. Springer.
- [8] Smith, S. W. (1999) *The Scientist and Engineer's Guide to Digital Signal Processing*, second edition. California Technical Publishing.
- [9] Priyanka S. Pariyal, D. M. G., Dhara M. Koyani (2016) Comparison based analysis of different fft architectures. *I.J. Image, Graphics and Signal Processing*, **8**, 41–47.
- [10] Burrus, C. S. (2008) *Fast Fourier Transform*. C. Sidney BurrusDaniel Williamson.
- [11] Alan V. Oppenheim, J. R., Ronald W. Schafer (1998) *Discrete Time Signal Processing*, 2 edition. Prentice-Hall, Inc.
- [12] Kalfika Yani, B. P., Achmad Rizal (2008) Analisis kinerja algoritma short time fourier transform (stft) untuk deteksi sinyal carrier frequency hopping spread spectrum (fhss) cdma. *Seminar Sistem Informasi Indonesia (SESINDO2008)*, Surabaya, Indonesia, 17 Desember 6. ITS , Surabaya.
- [13] Seo, N. (2008) Enee632 project5: Short time fourier transform. ENEE632 Project. 23 September 2019.
- [14] Application Note 243 (1994) *The Fundamental of Signal Analysis*. Hewlett-Packard Co. USA.
- [15] Liu, D. Y.-W. (2015) The short-time fourier transform. OpenCourseWare from National Tsing Hua University. 23 September 2019.

LAMPIRAN A

KODE PROGRAM

Kode A.1: Plotting.java

```
1 package UserApp;
2
3 /**
4  * 
5  * @author Ivan Kristanto <ivankristanto12@gmail.com>
6  */
7
8 public class Plotting {
9
10    public String plotSense;
11    public String plotTime;
12
13    public boolean makePlot;
14    public static String senseResultDefaultPath;
15
16    public void plotRealTime(String time, String senseResult) {
17        plotSense = senseResult;
18        plotTime = time;
19    }
20}
21}
```

Kode A.2: Tester.java

```
1 package UserApp;
2
3 import java.awt.event.WindowEvent;
4 import java.io.BufferedReader;
5 import java.io.File;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.nio.file.Files;
9 import java.text.SimpleDateFormat;
10 import java.util.Date;
11 import java.util.HashMap;
12 import java.util.Scanner;
13
14 import org.apache.tools.ant.BuildException;
15 import org.apache.tools.ant.DefaultLogger;
16 import org.apache.tools.ant.Project;
17 import org.apache.tools.ant.ProjectHelper;
18
19 import com.virtenio.commander.io.DataConnection;
20 import com.virtenio.commander.toolsets.preon32.Preon32Helper;
21
22 /**
23  * 
24  * @author Ivan Kristanto <ivankristanto12@gmail.com>
25  * 
26  */
27
28 public class Tester {
29
30    private static Thread senseThread;
31    private static BufferedReader in;
32    private static DataConnection conn;
33    private static boolean isSensing;
34    private static int totalSensor;
35
36    private static HashMap<String, Visualizing> vMap;
37
38    private static HashMap<String, double[][]> sensorAmplitudeResult;
39
40    public static void main(String[] args) throws Exception {
41
42        // init BaseStation
43        Tester tester = new Tester();
44        tester.context_set("context.set.1");
45        tester.time_synchronize();
46
47        Preon32Helper nodeHelper = new Preon32Helper("COM7", 115200);
48        conn = nodeHelper.runModule("basestation");
49        in = new BufferedReader(conn.getInputStream());
50        conn.flush();
```

```

51     System.out.println("\n\n");
52
53     // Input
54     Scanner sc = new Scanner(System.in);
55     System.out.println("Please insert total Sensor Node (not including basestation):");
56     totalSensor = sc.nextInt();
57
58     while (true) {
59         System.out.println(
60             "OPTION\n1: Check_Online_Node\n2: Sense\n3: Stop_Sensing\n4: Exit\nEnter option number:");
61         int input = sc.nextInt();
62         if (isSensing) {
63             if (input == 1 || input == 2 || input == 4) {
64                 System.out.println("INSENSING STATE..TO_STOP_USE_OPTION'3'");
65                 input = 0;
66             } else {
67                 conn.write(input);
68             }
69         } else {
70             conn.write(input);
71         }
72         byte[] buffer = new byte[1024];
73         if (input == 1) {
74             System.out.println("ONLINE_NODE:");
75             Thread.sleep(1000);
76             while (in.available() > 0) {
77                 in.read(buffer);
78                 conn.flush();
79                 String s = new String(buffer);
80                 String[] res = s.split(" ");
81                 for (int i = 0; i < res.length; i++) {
82                     if (res[i].length() != 0) {
83                         if (res[i].charAt(0) != '@') {
84                             String[] temp = res[i].split("#");
85                             if (temp.length > 1)
86                                 System.out.println(
87                                     ">" + temp[0] + "Time:" + formatTimetoString(Long.parseLong(temp[1])));
88                         }
89                     }
90                 }
91             }
92             System.out.println("CHECKING_DONE\n");
93         } else if (input == 2) {
94             System.out.println("SENSING..");
95             isSensing = true;
96             vMap = new HashMap<String, Visualizing>();
97             sensorAmplitudeResult = new HashMap<String, double[][]>();
98
99             for (int i = 1; i <= totalSensor; i++) {
100                 vMap.put("Sensor" + i, new Visualizing("Sensor" + i));
101                 vMap.get("Sensor" + i).showed = false;
102                 vMap.get("Sensor" + i).showPlot();
103                 vMap.get("Sensor" + i).canPlot = false;
104                 sensorAmplitudeResult.put("Sensor" + i, new double[128][128]);
105             }
106
107             if (senseThread == null) {
108                 senseThread = new Thread() {
109                     public void run() {
110                         while (true) && isSensing) {
111                             byte[] buffer = new byte[1024];
112                             try {
113                                 if (in.available() > 0) {
114                                     in.read(buffer);
115                                     conn.flush();
116                                     String s = new String(buffer);
117
118                                     String[] res = s.split(" ");
119                                     for (int i = 0; i < res.length; i++) {
120                                         if (res[i].trim().length() != 0) {
121                                             if (res[i].charAt(0) != '@' && res[i].charAt(0) != 'A'
122                                                 && res[i].charAt(0) != 'd') {
123                                                 String[] str = res[i].split(" ");
124                                                 String save = formatTimetoString(Long.parseLong(str[2])) + " "
125                                                 + str[3];
126                                                 saveSenseResult(str[0], save);
127
128                                                 int segment = str.length - 8;
129                                                 int idx = Integer.parseInt(str[1]);
130
131                                                 for (int k = 0; k < segment; k++) {
132                                                     sensorAmplitudeResult.get(str[0].trim())[k][idx] = Double
133                                                         .parseDouble(str[k + 4]);
134                                                 }
135                                             if (idx == 127) {
136                                                 double[] resTime = new double[128];
137                                                 for (int j = 0; j < 128; j++) {
138                                                     System.out.println();
139                                                     resTime[j] = j * Double.parseDouble(str[str.length - 1]);
140                                                 }
141                                                 double deltaTime = Double.parseDouble(str[str.length - 1]);
142                                                 double fs = Double.parseDouble(str[str.length - 2]);
143                                                 double overlap = Double.parseDouble(str[str.length - 3]);
144                                                 int windowSize = Integer.parseInt(str[str.length - 4]);
145                                                 new Spectrogram("Spectrogram_" + str[0],
146                                                     sensorAmplitudeResult.get(str[0]), resTime, fs,
147                                                     windowSize, overlap, deltaTime);
148                                             }
149                                         }
150                                     }
151                                 }
152                             }
153                         }
154                     }
155                 };
156                 senseThread.start();
157             }
158         }
159     }
160 }
```

```

150                                     // plotting
151                                     vMap.get(str[0].trim()).plotMaker.plotRealTime(str[2], str[3]);
152                                     vMap.get(str[0].trim()).canPlot = true;
153                                 }
154                             }
155                         }
156                     }
157                 }
158             } catch (NumberFormatException e) {
159             } catch (IOException e) {
160             }
161         }
162     };
163     senseThread.start();
164 }
165 //
166 }
167 }
168
169 // Stop Sensing
170 } else if (input == 3) {
171     if (isSensing) {
172         System.out.println("STOP_SENSING..WAIT_FOR_ALL_SENSOR_STOP_SENSING..");
173         for (int i = 1; i <= totalSensor; i++) {
174             Visualizing temp = vMap.get("Sensor" + i);
175             temp.frame.dispatchEvent(new WindowEvent(temp.frame, WindowEvent.WINDOW_CLOSING));
176         }
177         isSensing = false;
178         senseThread = null;
179         System.out.println("-STOP_SENSING_DONE-");
180     } else {
181         System.out.println("NOT_IN_SENSING_STATE..");
182     }
183
184 } else if (input == 4) {
185     System.out.println("TERMINATING_PROGRAM..");
186     isSensing = false;
187     Thread.sleep(1000);
188     System.out.println("..PROGRAM_TERMINATED..");
189     System.exit(0);
190 } else {
191     System.out.println("WRONG_INPUT");
192 }
193
194 }
195 }
196 }
197
198 // console build ant
199 private static DefaultLogger getConsoleLogger() {
200     DefaultLogger consoleLogger = new DefaultLogger();
201     consoleLogger.setErrorPrintStream(System.err);
202     consoleLogger.setOutputPrintStream(System.out);
203     consoleLogger.setLogLevel(Project.MSG_INFO);
204     return consoleLogger;
205 }
206
207 // ant build time synchronize
208 private void time_synchronize() throws Exception {
209     DefaultLogger consoleLogger = getConsoleLogger();
210     File buildFile = new File("C:\\Users\\Ifunk\\Desktop\\SKRIPSI\\Program_Skripsi\\Sandbox\\build.xml");
211     Project antProject = new Project();
212     antProject.setUserProperty("ant.file", buildFile.getAbsolutePath());
213     antProject.addBuildListener(consoleLogger);
214     try {
215         antProject.fireBuildStarted();
216         antProject.init();
217         ProjectHelper helper = ProjectHelper.getProjectHelper();
218         antProject.addReference("ant.ProjectHelper", helper);
219         helper.parse(antProject, buildFile);
220         String target = "cmd.time.synchronize";
221         antProject.executeTarget(target);
222         antProject.fireBuildFinished(null);
223     } catch (BuildException e) {
224         e.printStackTrace();
225     }
226 }
227
228 // set context basestation
229 private void context_set(String target) throws Exception {
230     DefaultLogger consoleLogger = getConsoleLogger();
231     File buildFile = new File("C:\\Users\\Ifunk\\Desktop\\SKRIPSI\\Program_Skripsi\\Sandbox\\buildUser.xml");
232     Project antProject = new Project();
233     antProject.setUserProperty("ant.file", buildFile.getAbsolutePath());
234     antProject.addBuildListener(consoleLogger);
235     try {
236         antProject.fireBuildStarted();
237         antProject.init();
238         ProjectHelper helper = ProjectHelper.getProjectHelper();
239         antProject.addReference("ant.ProjectHelper", helper);
240         helper.parse(antProject, buildFile);
241         antProject.executeTarget(target);
242         antProject.fireBuildFinished(null);
243     } catch (BuildException e) {
244         e.printStackTrace();
245     }
246 }
247
248 // save data

```

```

249 |     private static void saveSenseResult(String sensorName, String msg) {
250 |         File resFolder = new File("SenseResult");
251 |         if (!resFolder.exists()) {
252 |             resFolder.mkdir();
253 |         }
254 |
255 |         vMap.get(sensorName).plotMaker.senseResultDefaultPath = resFolder.getAbsolutePath();
256 |
257 |         Date date = new Date(System.currentTimeMillis());
258 |         String path = resFolder.getAbsolutePath() + "\\ " + sensorName + "-"
259 |             + new SimpleDateFormat("ddMMyyyy").format(date) + ".txt";
260 |         try {
261 |             FileWriter fileWriter = new FileWriter(path, true);
262 |             BufferedWriter buffWriter = new BufferedWriter(fileWriter);
263 |             File dataFile = new File(path);
264 |
265 |             buffWriter.append(msg + "\n");
266 |             buffWriter.close();
267 |             fileWriter.close();
268 |         } catch (Exception e) {
269 |         }
270     }
271 }
272 private static String formatTimetoString(long timeInMillis) {
273     Date date = new Date(timeInMillis);
274     SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss,SSS");
275     return simpleDateFormat.format(date);
276 }
277 }
278 }
```

Kode A.3: Visualizing.java

```

1 package UserApp;
2
3 import java.awt.BasicStroke;
4 import java.awt.BorderLayout;
5 import java.awt.Color;
6 import java.awt.Font;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ActionListener;
9 import java.awt.event.WindowAdapter;
10 import java.awt.event.WindowEvent;
11 import java.io.IOException;
12 import java.util.Date;
13
14 import javax.swing.BorderFactory;
15 import javax.swing.JFrame;
16 import javax.swing.JPanel;
17 import javax.swing.Timer;
18
19 import org.jfree.chart.ChartPanel;
20 import org.jfree.chart.ChartUtilities;
21 import org.jfree.chart.JFreeChart;
22 import org.jfree.chart.axis.DateAxis;
23 import org.jfree.chart.axis.NumberAxis;
24 import org.jfree.chart.plot.XYPlot;
25 import org.jfree.chart.renderer.xy.XYItemRenderer;
26 import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
27 import org.jfree.data.time.Millisecond;
28 import org.jfree.data.time.TimeSeries;
29 import org.jfree.data.time.TimeSeriesCollection;
30
31 /**
32 * 
33 * @author Ivan Kristanto <ivankristanto12@gmail.com>
34 * 
35 */
36 public class Visualizing extends JPanel {
37
38     public volatile boolean canPlot;
39     public boolean showed;
40
41     public Plotting plotMaker;
42     public JFrame frame;
43     public JFreeChart chart;
44     public XYPlot plot;
45     public String sensorId;
46
47     private TimeSeries senseResult;
48
49     public Visualizing(String sensorId) {
50         super(new BorderLayout());
51         this.plotMaker = new Plotting();
52         this.sensorId = sensorId;
53         showed = false;
54
55         this.senseResult = new TimeSeries("Acceleration");
56         this.senseResult.setMaximumItemAge(10000);
57
58         TimeSeriesCollection dataset = new TimeSeriesCollection();
59         dataset.addSeries(this.senseResult);
60
61         DateAxis domain = new DateAxis("Time");
62         NumberAxis range = new NumberAxis("Amplitude");
63         domain.setTickLabelFont(new Font("SansSerif", Font.PLAIN, 12));
64         range.setTickLabelFont(new Font("SansSerif", Font.PLAIN, 12));
65         domain.setLabelFont(new Font("SansSerif", Font.PLAIN, 14));
```

```

66     range.setLabelFont(new Font("SansSerif", Font.PLAIN, 14));
67
68     XYItemRenderer renderer = new XYLineAndShapeRenderer(true, true);
69     renderer.setSeriesPaint(0, Color.RED);
70     renderer.setSeriesPaint(1, Color.GREEN);
71
72     renderer.setSeriesStroke(0, new BasicStroke(3f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL));
73     renderer.setSeriesStroke(1, new BasicStroke(3f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL));
74     plot = new XYPlot(dataset, domain, range, renderer);
75     domain.setAutoRange(true);
76     domain.setLowerMargin(0.0);
77     domain.setUpperMargin(0.0);
78     domain.setTickLabelsVisible(true);
79
80     range.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
81
82     chart = new JFreeChart("Visualizing_" + sensorId, new Font("SansSerif", Font.BOLD, 24), plot, true);
83
84     ChartUtilities.applyCurrentTheme(chart);
85
86     ChartPanel chartPanel = new ChartPanel(chart, true);
87     chartPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createEmptyBorder(4, 4, 4, 4),
88                 BorderFactory.createLineBorder(Color.black)));
89     add(chartPanel);
90 }
91
92
93     private void addSenseObservation(long timeMilis, double senseRes) {
94         this.senseResult.addOrUpdate(new Millisecond(new Date(timeMilis)), senseRes);
95     }
96
97
98     public void showPlot() {
99         if (!showed) {
100             this.showed = true;
101             frame = new JFrame("Visualizing_" + sensorId);
102             Visualizing panel = this;
103             frame.getContentPane().add(panel, BorderLayout.CENTER);
104             frame.setBounds(200, 120, 700, 500);
105             frame.setVisible(true);
106
107             panel.new DataGenerator(0).start();
108             frame.addWindowListener(new WindowAdapter() {
109                 public void windowClosing(WindowEvent e) {
110                     System.out.println(sensorId + "_Monitor_exit..");
111                 }
112             });
113         }
114     }
115 }
116
117 class DataGenerator extends Timer implements ActionListener {
118
119     DataGenerator(int interval) {
120         super(interval, null);
121         addActionListener(this);
122     }
123
124     public void actionPerformed(ActionEvent event) {
125         if (canPlot) {
126             canPlot = false;
127             addSenseObservation(Long.parseLong(plotMaker.plotTime), Double.parseDouble(plotMaker.plotSense));
128         }
129     }
130 }
131 }
```

Kode A.4: Spectrogram.java

```

1 package UserApp;
2
3 import java.awt.Color;
4 import java.awt.Dimension;
5 import java.awt.Paint;
6 import javax.swing.JFrame;
7 import org.jfree.chart.ChartPanel;
8 import org.jfree.chart.JFreeChart;
9 import org.jfree.chart.axis.AxisLocation;
10 import org.jfree.chart.axis.NumberAxis;
11 import org.jfree.chart.plot.XYPlot;
12 import org.jfree.chart.renderer.PaintScale;
13 import org.jfree.chart.renderer.xy.XYBlockRenderer;
14 import org.jfree.chart.title.PaintScaleLegend;
15 import org.jfree.data.xy.DefaultXYZDataset;
16 import org.jfree.data.xy.XYDataset;
17 import org.jfree.data.xy.XYZDataset;
18 import org.jfree.ui.RectangleEdge;
19 import org.jfree.ui.RectangleInsets;
20
21
22 public class Spectrogram {
23
24     private static double maxAmp;
25     private static int length;
26     private static double[][] resAmplitude;
27     private static double[] resTime;
28     private static double fs;
29     private static int windowSize;
```

```

30    private static double overlap;
31    private static String title;
32    private static double deltaTime;
33
34    public Spectrogram(String title, double[][] resA, double[] resTime, double fs, int windowSize, double overlap,
35                        double deltaTime) {
36        this.windowSize = windowSize;
37        this.deltaTime = deltaTime;
38        this.overlap = overlap;
39        this.title = title;
40        this.resAmplitude = resA;
41        this.resTime = resTime;
42        this.length = resA[0].length;
43        this.maxAmp = 0;
44        this.fs = fs;
45        for (int i = 0; i < resA.length; i++) {
46            for (int j = 0; j < resA[0].length; j++) {
47                if (this.maxAmp < resA[i][j]) {
48                    this.maxAmp = resA[i][j];
49                }
50            }
51        }
52    }
53
54    JFrame f = new JFrame(title);
55    ChartPanel chartPanel = new ChartPanel(createChart(createDataset())) {
56        @Override
57        public Dimension getPreferredSize() {
58            return new Dimension(640, 480);
59        }
60    };
61    chartPanel.setMouseZoomable(true, false);
62    f.add(chartPanel);
63    f.pack();
64    f.setLocationRelativeTo(null);
65    f.setVisible(true);
66}
67
68    private static JFreeChart createChart(XYDataset dataset) {
69        NumberAxis xAxis = new NumberAxis("Time_(ms)");
70        NumberAxis yAxis = new NumberAxis("Frequency");
71        XYPlot plot = new XYPlot(dataset, xAxis, yAxis, null);
72        XYBlockRenderer r = new XYBlockRenderer();
73        SpectrumPaintScale ps = new SpectrumPaintScale(0, maxAmp);
74
75        r.setPaintScale(ps);
76        r.setBlockWidth(deltaTime);
77        plot.setRenderer(r);
78        JFreeChart chart = new JFreeChart(title, JFreeChart.DEFAULT_TITLE_FONT, plot, false);
79        NumberAxis scaleAxis = new NumberAxis("Amplitude");
80        scaleAxis.setAxisLinePaint(Color.white);
81        scaleAxis.setTickMarkPaint(Color.white);
82        PaintScaleLegend legend = new PaintScaleLegend(ps, scaleAxis);
83
84        legend.setSubdivisionCount(resAmplitude.length);
85
86        legend.setAxisLocation(AxisLocation.TOP_OR_RIGHT);
87        legend.setPadding(new RectangleInsets(10, 10, 10, 10));
88        legend.setStripWidth(20);
89        legend.setPosition(RectangleEdge.RIGHT);
90        legend.setBackgroundPaint(Color.WHITE);
91        chart.addSubtitle(legend);
92        chart.setBackgroundPaint(Color.white);
93        return chart;
94    }
95
96    private static XYZDataset createDataset() {
97        DefaultXYZDataset dataset = new DefaultXYZDataset();
98        int idx = 0;
99        for (int i = 0; i < resAmplitude[0].length; i++) {
100            double[][] data = new double[3][length];
101            for (int j = 0; j < resAmplitude[0].length / 2; j++) {
102                data[0][j] = resTime[i];
103                data[1][j] = j;
104                data[2][j] = resAmplitude[idx][j];
105                dataset.addSeries("Series" + i, data);
106            }
107            if (overlap == 0 && windowSize != 0) {
108                if ((i % (windowSize - 1)) == 0 && i > 1) {
109                    idx++;
110                }
111            } else {
112                if ((i % ((windowSize * overlap) + 1)) == 0 && i > 1) {
113                    idx++;
114                }
115            }
116        }
117
118        return dataset;
119    }
120
121    private static class SpectrumPaintScale implements PaintScale {
122
123        private final double lowerBound;
124        private final double upperBound;
125
126        public SpectrumPaintScale(double lowerBound, double upperBound) {
127            this.lowerBound = lowerBound;
128            this.upperBound = upperBound;

```

```

129     }
130
131     @Override
132     public double getLowerBound() {
133         return lowerBound;
134     }
135
136     @Override
137     public double getUpperBound() {
138         return upperBound;
139     }
140
141     @Override
142     public Paint getPaint(double value) {
143         float scaledValue = (float) (value / (getUpperBound() - getLowerBound()));
144         float scaledH = scaledValue;
145         return Color.getHSBColor(0, 0, scaledH);
146     }
147 }
148 }
```

Kode A.5: Accelerometer.java

```

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

Kode A.6: Complex.java

```

1 package SensorNode;
2
3 public class Complex {
4     public double re;
5     public double im;
6
7     public Complex(double r, double i) {
8         this.re = r;
9         this.im = i;
10    }
11
12    public Complex tambah(Complex x) {
13        Complex res = new Complex(this.re + x.re, this.im + x.im);
14        return res;
15    }
16
17    public Complex kurang(Complex x) {
18        Complex res = new Complex(this.re - x.re, this.im - x.im);
19        return res;
20    }
21
22
23    public Complex kali(Complex x) {
24        Complex res = new Complex(this.re * x.re - this.im * x.im, this.re * x.im + this.im * x.re);
25        return res;
26    }
27
28    public double absolute() {
29        return Math.sqrt(this.re * this.re + this.im * this.im );
30    }
31
32    @Override
33    public String toString() {
34        return this.re+"_"+this.im;
```

```
35| }
36| }
```

Kode A.7: FFT.java

```

1 package SensorNode;
2
3 public class FFT {
4
5     public int length;
6
7     public FFT(int length) {
8         this.length = length;
9     }
10
11    public int bitReverse(int n, int bits) {
12        if (n == 0) {
13            return 0;
14        }
15
16        String temp = Integer.toBinaryString(n);
17        if (temp.length() < bits) {
18            int k = bits - temp.length();
19            for (int i = 0; i < k; i++) {
20                temp = "0" + temp;
21            }
22        }
23
24        int res = 0;
25        for (int i = temp.length() - 1; i >= 0; i--) {
26            if (temp.charAt(i) == '1') {
27                res += Math.pow(2, i);
28            }
29        }
30
31        return res;
32    }
33
34    public Complex[] fft(Complex[] input) {
35        int bits = (int) (Math.log(input.length) / Math.log(2));
36        Complex[] finalOrder = new Complex[input.length];
37        for (int i = 0; i < input.length; i++) {
38            int order = bitReverse(i, bits);
39            finalOrder[i] = input[order];
40        }
41
42        for (int N = 2; N <= finalOrder.length; N = N * 2) {
43            for (int i = 0; i < finalOrder.length; i += N) {
44                for (int k = 0; k < N / 2; k++) {
45
46                    Complex first = finalOrder[i + k];
47                    Complex second = finalOrder[i + k + (N / 2)];
48
49                    double w = (-2 * Math.PI * k) / (double) N;
50                    Complex exp = (new Complex(Math.cos(w), Math.sin(w)).kali(second));
51
52                    finalOrder[i + k] = first.tambah(exp);
53                    finalOrder[i + k + (N / 2)] = first.kurang(exp);
54                }
55            }
56        }
57        return finalOrder;
58    }
59
60}
```

Kode A.8: SenseController.java

```

1 package SensorNode;
2
3 import com.virtenio.vm.Time;
4
5 public class SenseController {
6     private Accelerometer sensor;
7     public static final int X_AXIS = 0;
8     public static final int Y_AXIS = 1;
9     public static final int Z_AXIS = 2;
10    public final int N = 128;
11    public long [] time;
12    public double fs;
13    public double [] senseResult;
14    public double deltaTime;
15
16    public SenseController(Accelerometer sensor) {
17        this.sensor = sensor;
18        this.senseResult = new double[N];
19    }
20
21    private float[] sensing() {
22        try {
23            return sensor.sense();
24        } catch (Exception e) {
25            System.out.println("Sense_Error");
26        }
27        return null;
28    }
29}
```

```

30 public Complex[] createSample(int axis) throws InterruptedException {
31     Complex[] res = new Complex[N];
32     time = new long[N];
33     for (int i = 0; i < N; i++) {
34         senseResult[i] = this.sensing()[axis];
35         res[i] = new Complex(senseResult[i], 0);
36         time[i] = Time.currentTimeMillis();
37         Thread.sleep(16);
38     }
39     this.fs = N / ((time[N-1] - time[0])/1000); //sampling rate
40     this.deltaTime = time[1] - time[0];
41     return res;
42 }
43
44
45 }

```

Kode A.9: ShortTimeFourierTransform.java

```

1 package SensorNode;
2
3 public class ShortTimeFourierTransform implements Runnable {
4
5     public Complex[] input;
6     public Complex[] output;
7     public double[][] amplitude;
8     public FFT fft;
9     public int N = 128;
10    public double overlap = 0;
11    public int windowSize = 64;
12    public double fs;
13    public int segment;
14
15    public ShortTimeFourierTransform(FFT fft, Complex[] input, double fs) {
16        this.input = input;
17        this.fft = fft;
18        this.segment = (int) Math.floor((N - windowSize) / (windowSize - (overlap * windowSize))) + 1;
19        this.amplitude = new double[segment][N];
20        this.fs = fs;
21    }
22
23    // rectangle window
24    public void STFT() {
25        int idx = 0;
26        while (idx < segment) {
27            Complex[] windowedInput = new Complex[fft.length]; // panjang arr window sama dengan fft
28            for (int i = 0; i < windowSize; i++) {
29                int inputIdx;
30                if (overlap == 0) {
31                    inputIdx = i + (idx * (windowSize - 1));
32                } else {
33                    inputIdx = (int) (i + (idx * (windowSize - 1) * overlap));
34                }
35
36                //Pemilihan window
37                windowedInput[inputIdx] = rectangularWindow(input[inputIdx]);
38                windowedInput[inputIdx] = hannWindow(input[inputIdx], i);
39                windowedInput[inputIdx] = hammWindow(input[inputIdx], i);
40            }
41        }
42
43        for (int i = 0; i < windowedInput.length; i++) {
44            if (windowedInput[i] == null) {
45                windowedInput[i] = new Complex(0, 0);
46            }
47        }
48
49        output = this.fft.fft(windowedInput);
50        System.out.println("selesai");
51        for (int i = 0; i < output.length; i++) {
52            amplitude[idx][i] = output[i].absolute();
53        }
54        idx++;
55    }
56 }
57
58 @Override
59 public void run() {
60     this.STFT();
61 }
62
63 // w[n] = 1
64 public Complex rectangularWindow(Complex input) {
65     return input.kali(new Complex(1, 0));
66 }
67
68 // w[n] = 0.5 - 0.5 * cos (2 pi * i /N)
69 public Complex hannWindow(Complex input, int idx) {
70     double multiplier = 0.5 - 0.5 * Math.cos(2 * Math.PI * idx / N);
71     return input.kali(new Complex(multiplier, 0));
72 }
73
74 // w[n] = 0.54 - 0.46 * cos (2 pi * i /N)
75 public Complex hammWindow(Complex input, int idx) {
76     double multiplier = 0.54 - 0.46 * Math.cos(2 * Math.PI * idx / N);
77     return input.kali(new Complex(multiplier, 0));
78 }
79

```

80 | }
81 | }

Kode A.10: SNManager.java

```

1 package SensorNode;
2
3 import java.io.IOException;
4 import java.util.Random;
5
6 import com.virtenio.driver.device.at86rf231.AT86RF231;
7 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
8 import com.virtenio.misc.PropertyHelper;
9 import com.virtenio.preon32.node.Node;
10 import com.virtenio.radio.ieee_802_15_4.Frame;
11 import com.virtenio.radio.ieee_802_15_4.FrameIO;
12 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
13 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
14 import com.virtenio.vm.Time;
15
16 public class SNManager {
17
18     private static final Accelerometer sensor = new Accelerometer();
19     private static boolean isSensing = true;
20     private static SenseController sc = new SenseController(sensor);
21     private static Complex[] input;
22     private static Thread sensingThread;
23
24     // Setting Address
25     private static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
26     private static int[] node_list = new int[] { PropertyHelper.getInt("radio.panid", 0xABFE),
27         PropertyHelper.getInt("radio.panid", 0xDAAA), PropertyHelper.getInt("radio.panid", 0xDAAB),
28         PropertyHelper.getInt("radio.panid", 0xDAAC), PropertyHelper.getInt("radio.panid", 0xDAAD),
29         PropertyHelper.getInt("radio.panid", 0xDAAE) };
30
31     private static final String sensorId = "Sensor1";
32     private static int SENSOR_NODE_ADDRESS = node_list[1];
33     private static int baseStationAddr = node_list[0];
34     private static int[] nextNode = { node_list[2] };
35     private static int[] previousNode = { node_list[0] };
36
37     private static boolean sendAck;
38     private static boolean already;
39
40     public static void main(String[] args) throws Exception {
41         sendAck = false;
42         already = false;
43         sensor.init();
44         System.out.println(sensorId + " Waiting for task..");
45         runs();
46     }
47
48     public static void runs() {
49         try {
50             AT86RF231 t = Node.getInstance().getTransceiver();
51             t.open();
52             t.setAddressFilter(COMMON_PANID, SENSOR_NODE_ADDRESS, SENSOR_NODE_ADDRESS, false);
53             final RadioDriver radioDriver = new AT86RF231RadioDriver(t);
54             final FrameIO fio = new RadioDriverFrameIO(radioDriver);
55             receive(fio);
56
57         } catch (Exception e) {
58     }
59 }
60
61     public static void receive(final FrameIO fio) {
62         Thread thread = new Thread() {
63             public void run() {
64                 Frame frame = new Frame();
65                 while (true) {
66                     try {
67                         fio.receive(frame);
68                         byte[] content = frame.getPayload();
69                         String str = new String(content, 0, content.length);
70                         if (str.length() > 2) {
71                             if (str.substring(0, 4).equalsIgnoreCase("ACK2") && !sendAck) {
72                                 if (str.substring(4).equals(sensorId)) {
73                                     already = true;
74                                     Thread.sleep(20);
75                                 } else {
76                                     if (nextNode.length != 0) {
77                                         System.out.println("toNextNode1_" + str);
78                                         forwardMsgToNextNode(str, fio);
79                                     }
80                                 }
81                             } else if (str.substring(0, 3).equalsIgnoreCase("ACK")) {
82                                 if (str.substring(3).equals(sensorId) && !already) {
83                                     sendAck = true;
84                                     Thread.sleep(20);
85                                 } else {
86                                     if (nextNode.length != 0) {
87                                         System.out.println("toNextNode2_" + str);
88                                         forwardMsgToNextNode(str, fio);
89                                     }
90                                 }
91                             }
92                         }
93                     }
94                 }
95             }
96         }
97     }

```

```

94         // forward msg to previous
95         if (str.charAt(0) != '@') {
96             forwardMsgToPreviousNode(str, fio);
97         }
98         // online status
99         else if (str.substring(0, 2).equalsIgnoreCase("@1")) {
100             System.out.println("Online");
101             long curTime = Long.parseLong(str.substring(2));
102             Time.setCurrentTimeMillis(curTime);
103             curTime = Time.currentTimeMillis();
104             forwardMsgToNextNode("@1" + curTime, fio);
105             forwardMsgToPreviousNode("1" + sensorId + "_ONLINE#" + curTime, fio);
106
107     } // start sensing
108     else if (str.equalsIgnoreCase("@2")) {
109         System.out.println("Start_Sensing..");
110         forwardMsgToNextNode("@2", fio);
111         isSensing = true;
112
113         sensingThread = new Thread() {
114             public void run() {
115                 Thread computingThread = null;
116                 while (isSensing) {
117                     try {
118                         input = sc.createSample(sc.X_AXIS);
119                         FFT fft = new FFT(sc.N);
120
121                         ShortTimeFourierTransform stft = new ShortTimeFourierTransform(fft, input,
122                                         sc.fs);
123                         computingThread = new Thread(stft);
124                         computingThread.start();
125                         while (true) {
126                             if (!computingThread.isAlive()) {
127                                 Complex[] FFTcomputed = stft.output;
128                                 double[][] amplitude = stft.amplitude;
129                                 for (int j = 0; j < stft.N; j++) {
130                                     String res = "2" + sensorId + "_" + j + "_" + sc.time[j] + "_"
131                                         + sc.senseResult[j];
132                                     for (int i = 0; i < stft.segment; i++) {
133                                         res += "_" + amplitude[i][j];
134                                     }
135                                     res += "_" + stft.windowSize + "_" + stft.overlap + "_"
136                                         + stft.fs + "_" + sc.deltaTime;
137
138                                     sendAck = false;
139                                     already = false;
140                                     while (!sendAck) {
141                                         already = false;
142                                         System.out.println("send_A");
143                                         forwardMsgToPreviousNode("A" + sensorId, fio);
144                                         Thread.sleep(new Random().nextInt(50) + 10);
145                                     }
146                                     System.out.println("send_result");
147                                     while (!already) {
148                                         sendAck = false;
149                                         forwardMsgToPreviousNode(res, fio);
150                                         Thread.sleep(new Random().nextInt(50) + 10);
151                                     }
152                                     already = false;
153
154                                     System.out.println("sending_data_[ " + j + " ]..");
155                                     Thread.sleep(10);
156                                 }
157
158                                 forwardMsgToPreviousNode("4doneA" + sensorId, fio);
159
160                                 System.out.println("DONE");
161                                 break;
162                             }
163                         }
164                     } catch (InterruptedException e) {
165                         }
166                 }
167             };
168             sensingThread.start();
169
170     } // stop sensing
171     else if (str.equalsIgnoreCase("@3")) {
172         forwardMsgToNextNode("@3", fio);
173         isSensing = false;
174         System.out.println("Stop_Sensing..Waiting_for_task..");
175         System.exit(0);
176
177     } // exit
178     else if (str.equalsIgnoreCase("@4")) {
179         forwardMsgToNextNode("@4", fio);
180         System.out.println("Exiting..");
181         System.exit(0);
182     }
183
184     } catch (IOException e) {
185     } catch (InterruptedException e) {
186     }
187 }
188 }
189
190 thread.start();
191
192 }
```

```

193
194     public static void send(String msg, int src, int dest, FrameIO fio) {
195         int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
196             | Frame.SRC_ADDR_16;
197         final Frame sentFrame = new Frame(frameControl);
198         sentFrame.setDestPanId(COMMON_PANID);
199         sentFrame.setDestAddr(dest);
200         sentFrame.setSrcAddr(src);
201         sentFrame.setPayload(msg.getBytes());
202         try {
203             fio.transmit(sentFrame);
204         } catch (Exception e) {
205         }
206     }
207
208     private static void forwardMsgToNextNode(String msg, FrameIO fio) {
209         for (int i = 0; i < nextNode.length; i++) {
210             send(msg, SENSOR_NODE_ADDRESS, nextNode[i], fio);
211         }
212     }
213
214     private static void forwardMsgToPreviousNode(String msg, FrameIO fio) {
215         for (int i = 0; i < previousNode.length; i++) {
216             send(msg, SENSOR_NODE_ADDRESS, previousNode[i], fio);
217         }
218     }
219 }

```

Kode A.11: BSManager.java

```

1 package BaseStation;
2
3 import java.io.IOException;
4 import java.io.OutputStream;
5 import java.util.HashMap;
6 import java.util.Random;
7
8 import com.virtenio.driver.device.at86rf231.AT86RF231;
9 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
10 import com.virtenio.driver.timer.NativeTimer;
11 import com.virtenio.driver.timer.Timer;
12 import com.virtenio.driver.timer.TimerException;
13 import com.virtenio.driver.usart.NativeUSART;
14 import com.virtenio.driver.usart.USART;
15 import com.virtenio.driver.usart.USARTException;
16 import com.virtenio.driver.usart.USARTParams;
17 import com.virtenio.misc.PropertyHelper;
18 import com.virtenio.preon32.examples.common.USARTConstants;
19 import com.virtenio.preon32.node.Node;
20
21 import com.virtenio.radio.ieee_802_15_4.Frame;
22 import com.virtenio.radio.ieee_802_15_4.FrameIO;
23 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
24 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
25 import com.virtenio.vm.Time;
26 import com.virtenio.vm.event.AsyncEvent;
27 import com.virtenio.vm.event.AsyncEventHandler;
28 import com.virtenio.vm.event.AsyncEvents;
29
30 public class BSManager {
31
32     private static USART usart;
33     private static OutputStream out;
34
35     // Setting Address
36     private static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
37     private static int[] node_list = new int[] { PropertyHelper.getInt("radio.panid", 0xABFE),
38         PropertyHelper.getInt("radio.panid", 0xDAAC), PropertyHelper.getInt("radio.panid", 0xDAAB),
39         PropertyHelper.getInt("radio.panid", 0xDAAE), PropertyHelper.getInt("radio.panid", 0xDAAD),
40         PropertyHelper.getInt("radio.panid", 0xDAAE) };
41
42     private static int currAddr = node_list[0];
43
44 //    private static int[] connectedNodeAddr = new int[] { node_list[1] };
45 //    private static int[] connectedNodeAddr = new int[] { node_list[1], node_list[2], node_list[3], node_list[4],
46 //        node_list[5] };
47
48     private static HashMap<String, Integer> nodeAddrMap;
49
50     private static boolean already;
51     private static String ackAddr;
52     private static String sentResult;
53
54 /**
55 * @param args
56 * @throws Exception
57 */
58
59     public static void main(String[] args) throws Exception {
60         nodeAddrMap = new HashMap<String, Integer>();
61         for (int i = 1; i <= connectedNodeAddr.length; i++) {
62             nodeAddrMap.put("ASensor" + i, connectedNodeAddr[i - 1]);
63         }
64
65         usart = configUSART();
66         out = usart.getOutputStream();
67     }

```

```

68     ackAddr = "";
69     sentResult = "";
70     already = false;
71
72     new Thread() {
73         public void run() {
74             runs();
75         }
76     }.start();
77 }
78 /**
79 *
80 */
81
82 public static void runs() {
83
84     try {
85         AT86RF231 t = Node.getInstance().getTransceiver();
86         t.open();
87         t.setAddressFilter(COMMON_PANID, currAddr, currAddr, false);
88         final RadioDriver radioDriver = new AT86RF231RadioDriver(t);
89         final FrameIO fio = new RadioDriverFrameIO(radioDriver);
90
91         new Thread() {
92             public void run() {
93                 while (true) {
94                     String res;
95                     int input;
96                     try {
97                         input = usart.read();
98                         if (input == 1) {
99                             long curTime = Time.currentTimeMillis();
100                            res = new String("_BaseStation_ONLINE_" + curTime + "_");
101                            out.write(res.getBytes());
102                            usart.flush();
103                            for (int i = 0; i < connectedNodeAddr.length; i++) {
104                                send("@1", currAddr, currAddr, connectedNodeAddr[i], fio);
105                            }
106                        } else if (input == 2) {
107                            for (int i = 0; i < connectedNodeAddr.length; i++) {
108                                send("@2", currAddr, connectedNodeAddr[i], fio);
109                            }
110                        } else if (input == 3) {
111                            for (int i = 0; i < connectedNodeAddr.length; i++) {
112                                send("@3", currAddr, connectedNodeAddr[i], fio);
113                            }
114                            usart = configUSART();
115                            out = usart.getOutputStream();
116                            already = false;
117                            ackAddr = "";
118                            sentResult = "";
119                        } else if (input == 4) {
120                            for (int i = 0; i < connectedNodeAddr.length; i++) {
121                                send("@4", currAddr, connectedNodeAddr[i], fio);
122                            }
123                            Thread.sleep(50);
124                        }
125                    } catch (USARTException e) {
126                    } catch (IOException e) {
127                    } catch (InterruptedException e) {
128                    }
129                }
130            }.start();
131
132            new Thread() {
133                public void run() {
134                    receive(fio);
135                }
136            }.start();
137
138        } catch (Exception e) {
139            e.printStackTrace();
140        }
141    }
142
143 /**
144 *
145 * @param fio
146 */
147
148 public static void receive(final FrameIO fio) {
149     while (true) {
150         Frame frame = new Frame();
151         try {
152             fio.receive(frame);
153             byte[] content = frame.getPayload();
154             String str = new String(content, 0, content.length);
155             String res;
156             // done
157             if (str.trim().charAt(0) == '4') {
158                 if (str.substring(5).equals(ackAddr)) {
159                     already = false;
160                     ackAddr = "";
161                 }
162                 res = str.substring(1, 5);
163                 try {
164                     out.write(res.getBytes());
165                     usart.flush();
166                 } catch (USARTException e) {

```

```

167     }
168     // ASensor#
169     if (str.charAt(0) == 'A' && str.length() > 3 && str.charAt(1) == 'S') {
170         // first set ackAddr
171         if (ackAddr.equals("")) {
172             ackAddr = str;
173         }
174         // send ACKSensor#
175         if (ackAddr.length() != 0 && ackAddr.equals(str)) {
176             if (ackAddr.equals(str) && already) {
177                 already = false;
178                 ackAddr = "";
179             } else {
180                 if (!nodeAddrMap.containsKey(ackAddr)) {
181                     for (int i = 0; i < connectedNodeAddr.length; i++) {
182                         send("ACK" + ackAddr.substring(1), currAddr, connectedNodeAddr[i], fio);
183                     }
184                 } else {
185                     send("ACK" + ackAddr.substring(1), currAddr, nodeAddrMap.get(ackAddr), fio);
186                 }
187                 sentResult = "";
188             }
189         }
190     } else {
191         if (str.charAt(0) == 'I') {
192             res = "_" + str.substring(1) + "_";
193             try {
194                 out.write(res.getBytes());
195                 usart.flush();
196             } catch (USARTException e) {
197             }
198         }
199         // receive result
200         if (str.charAt(0) == '2') {
201             res = "_" + str.substring(1) + "_";
202             if (sentResult.equals("") && !already && res.substring(1, 8).equals(ackAddr.substring(1))) {
203                 sentResult = res;
204                 try {
205                     out.write(res.getBytes());
206                     usart.flush();
207                 } catch (USARTException e) {
208                 }
209             }
210             already = true;
211         }
212     }
213     // already send ACK2Sensor#
214     if (already) {
215         if (!nodeAddrMap.containsKey(ackAddr)) {
216             for (int i = 0; i < connectedNodeAddr.length; i++) {
217                 send("ACK2" + ackAddr.substring(1), currAddr, connectedNodeAddr[i], fio);
218                 Thread.sleep(10);
219             }
220         } else {
221             send("ACK2" + ackAddr.substring(1), currAddr, nodeAddrMap.get(ackAddr), fio);
222         }
223     }
224 }
225 } catch (IOException e) {
226 } catch (InterruptedException e) {
227 }
228 }
229 }
230 }
231 /**
232 *
233 * @param msg
234 * @param src
235 * @param dest
236 * @param fio
237 */
238 public static void send(String msg, int src, int dest, FrameIO fio) {
239     int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
240     | Frame.SRC_ADDR_16;
241     final Frame sentFrame = new Frame(frameControl);
242     sentFrame.setDestPanId(COMMON_PANID);
243     sentFrame.setDestAddr(dest);
244     sentFrame.setSrcAddr(src);
245     sentFrame.setPayload(msg.getBytes());
246     try {
247         fio.transmit(sentFrame);
248         Thread.sleep(50);
249     } catch (Exception e) {
250     }
251     System.out.println(msg);
252 }
253 /**
254 *
255 * @return
256 */
257 private static USART configUSART() {
258     USARTParams params = USARTConstants.PARAMS_115200;
259     NativeUSART usart = NativeUSART.getInstance(0);
260     try {
261         usart.close();
262         usart.open(params);
263     }
264 }
```

```
266|         return usart;
267|     } catch (Exception e) {
268|         return null;
269|     }
270| }
271| }
```

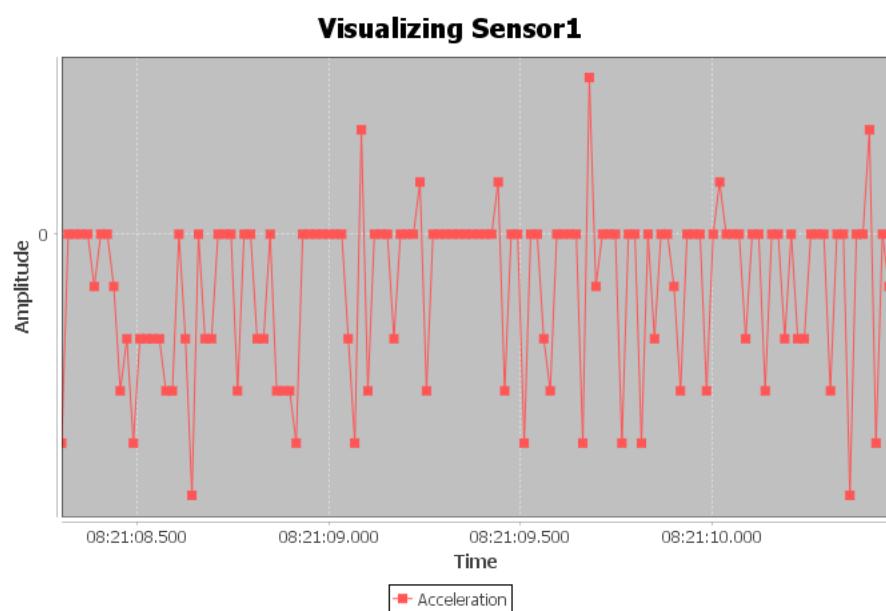

LAMPIRAN B

HASIL EKSPERIMEN DI ATAP

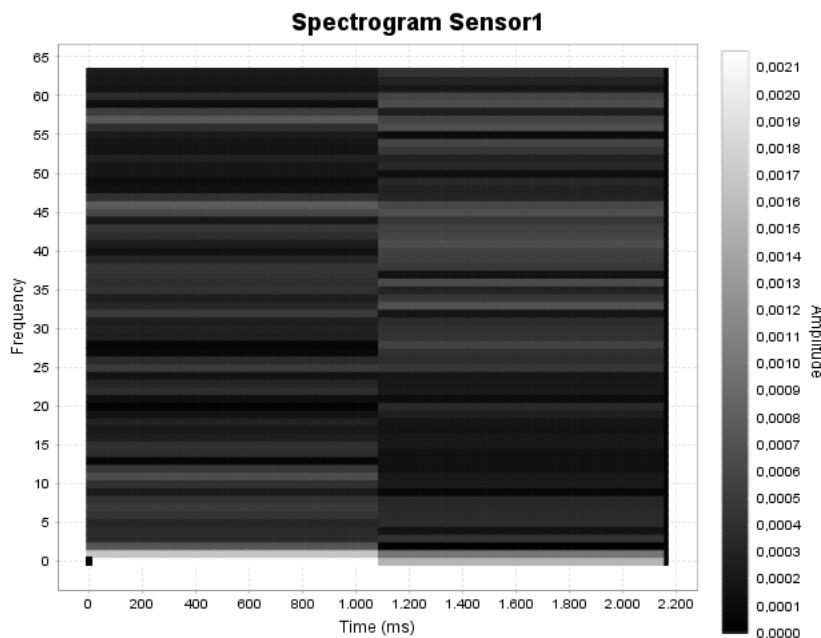
B.1 Topologi Star

B.1.1 Rectangle Window

- Sensor1

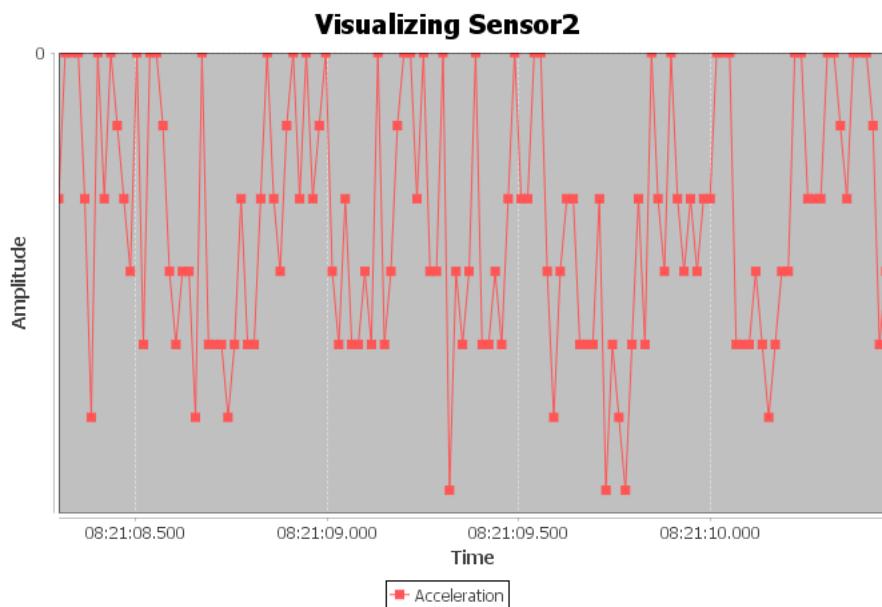


Gambar B.1: Hasil Sensing Sensor1 dengan topologi star dan window *Rectangle* di Atap

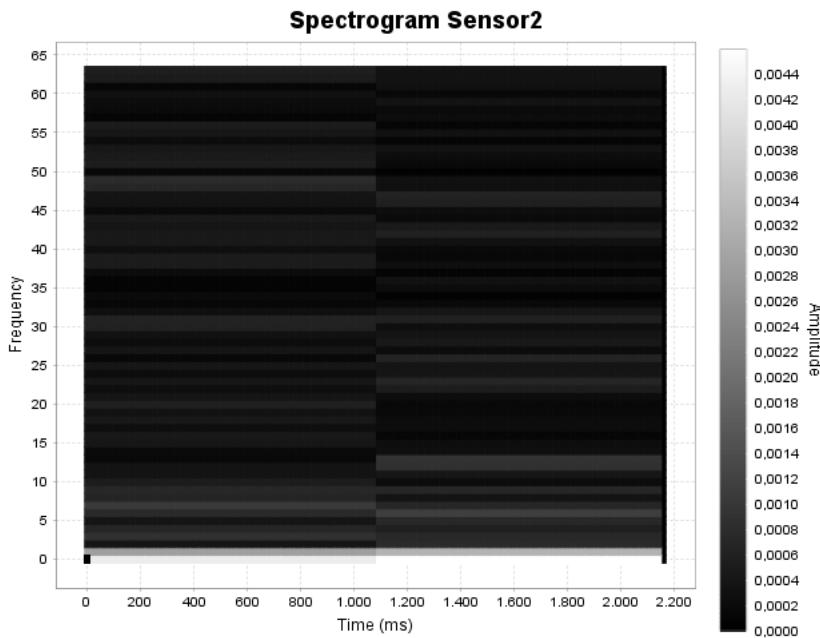


Gambar B.2: Hasil Spectrogram Sensor1 dengan topologi star dan window *Rectangle* di Atap

- Sensor2

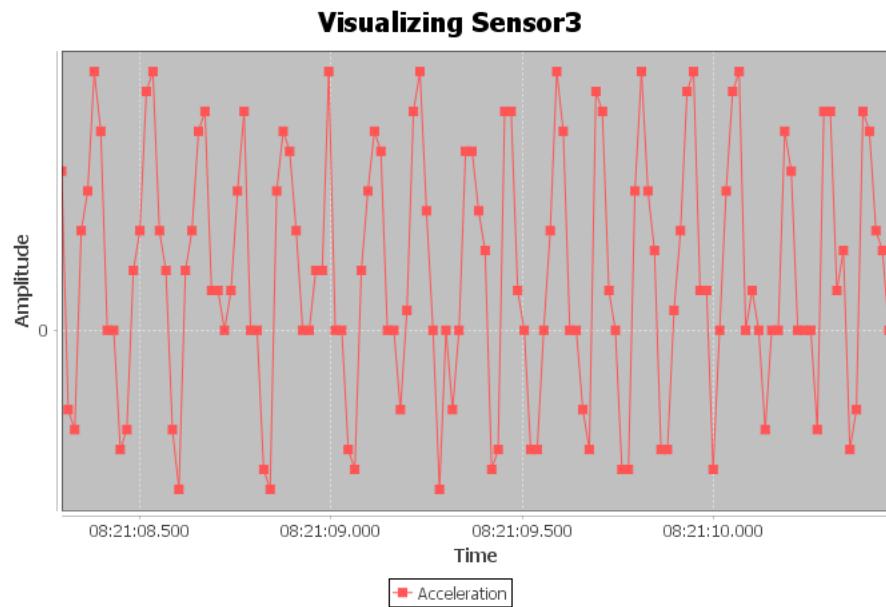


Gambar B.3: Hasil Sensing Sensor2 dengan topologi star dan window *Rectangle* di Atap

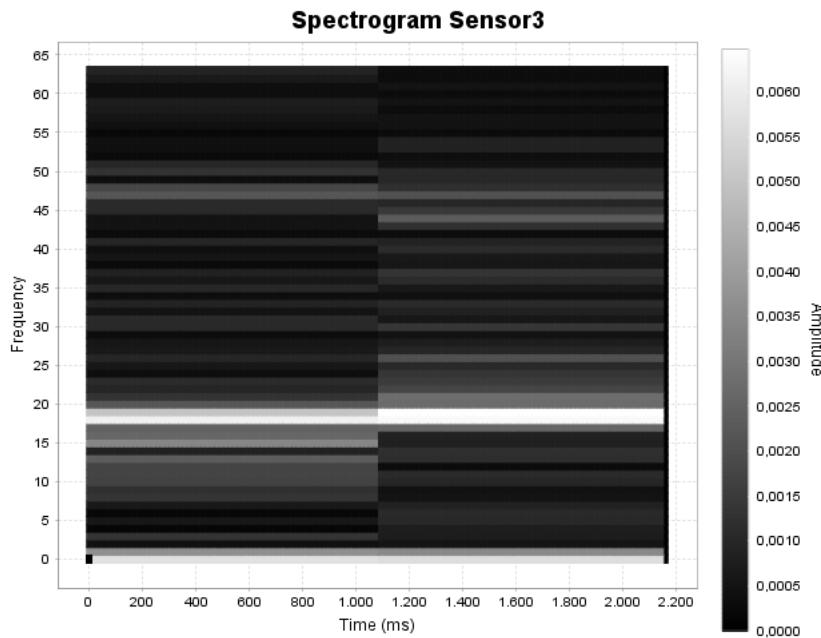


Gambar B.4: Hasil Spectrogram Sensor2 dengan topologi star dan window *Rectangle* di Atap

- Sensor3

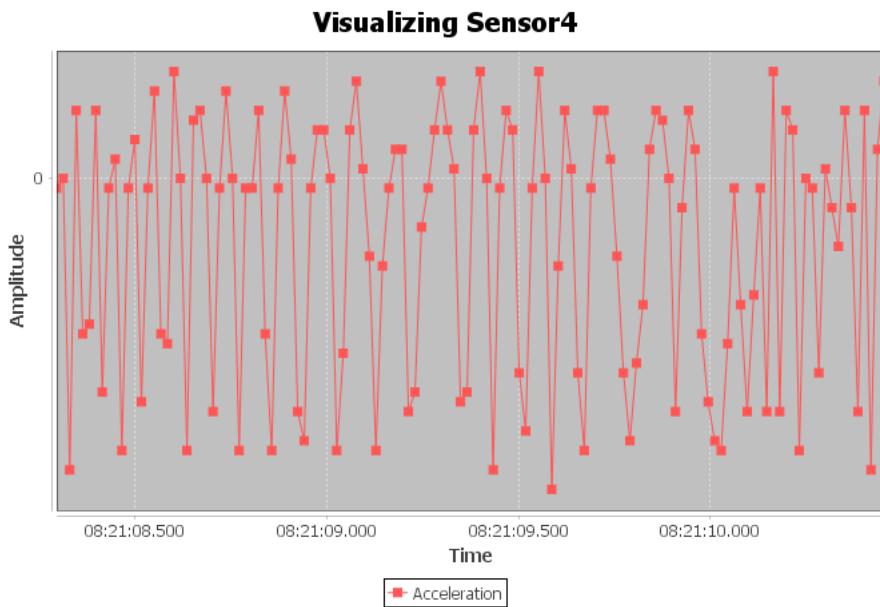


Gambar B.5: Hasil Sensing Sensor3 dengan topologi star dan window *Rectangle* di Atap

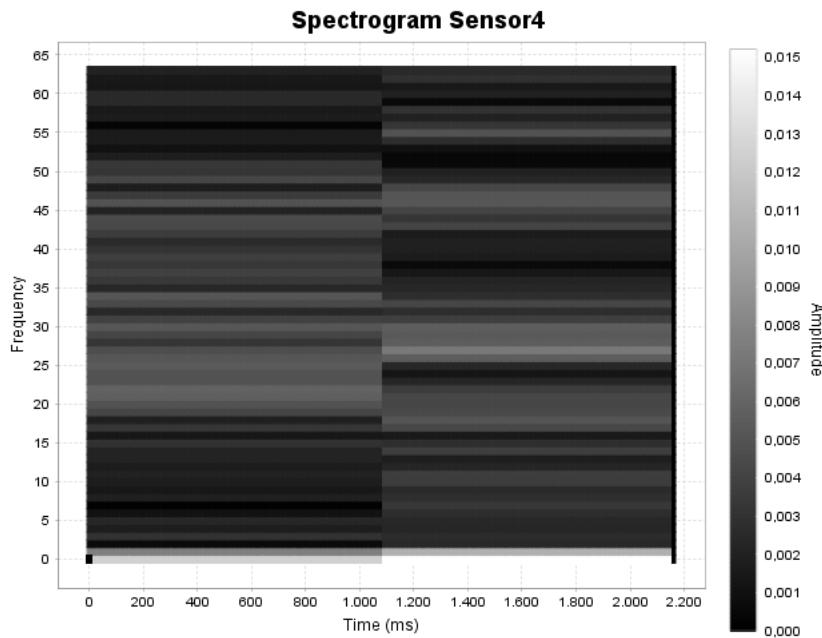


Gambar B.6: Hasil Spectrogram Sensor3 dengan topologi star dan window *Rectangle* di Atap

- Sensor4

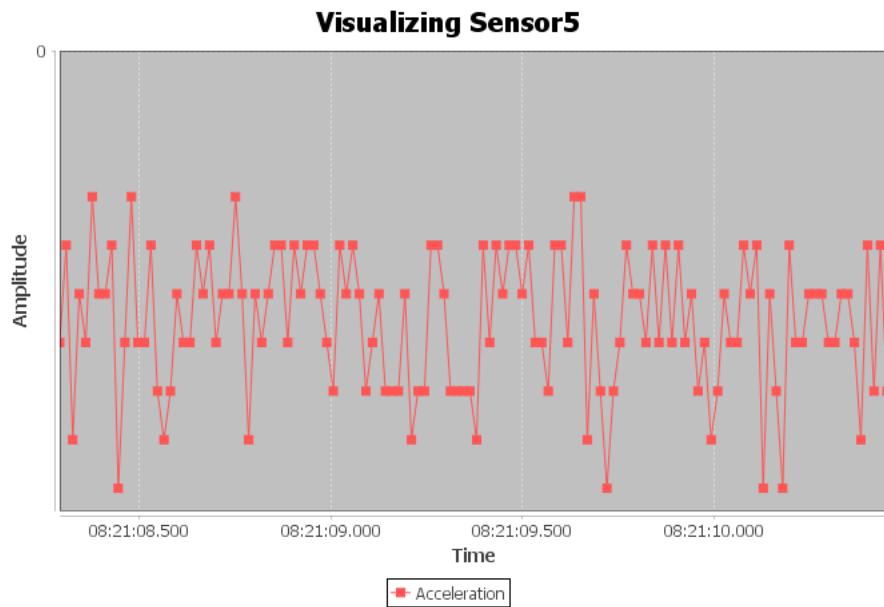


Gambar B.7: Hasil Sensing Sensor4 dengan topologi star dan window *Rectangle* di Atap

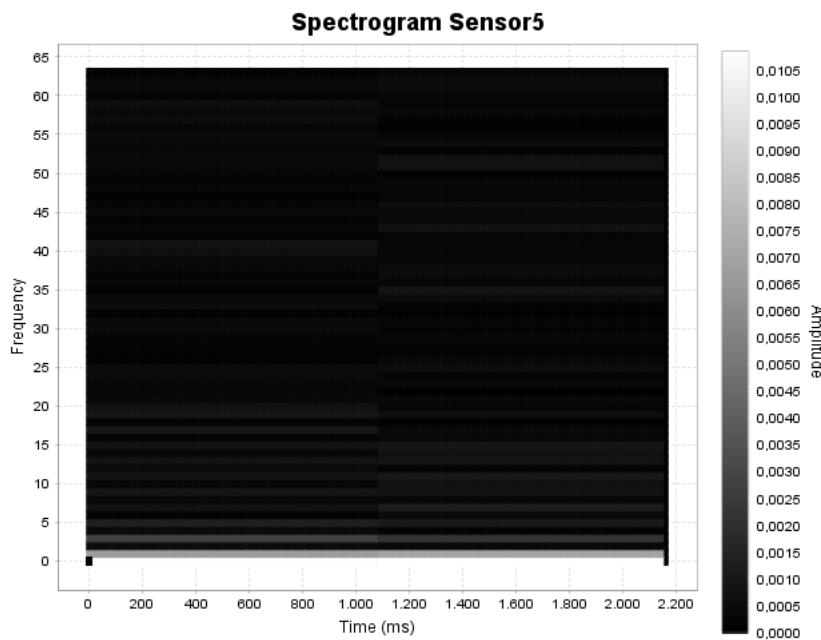


Gambar B.8: Hasil Spectrogram Sensor4 dengan topologi star dan window *Rectangle* di Atap

- Sensor5



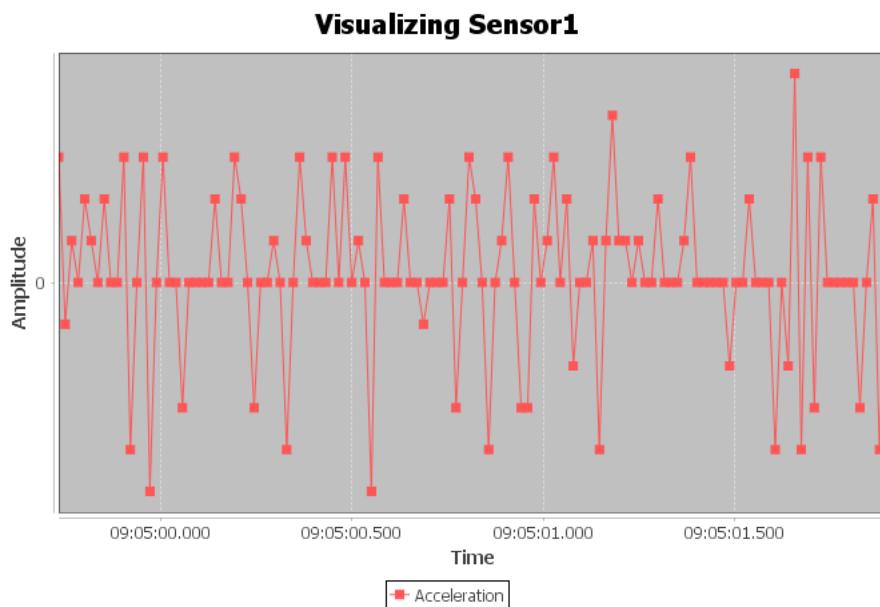
Gambar B.9: Hasil Sensing Sensor5 dengan topologi star dan window *Rectangle* di Atap



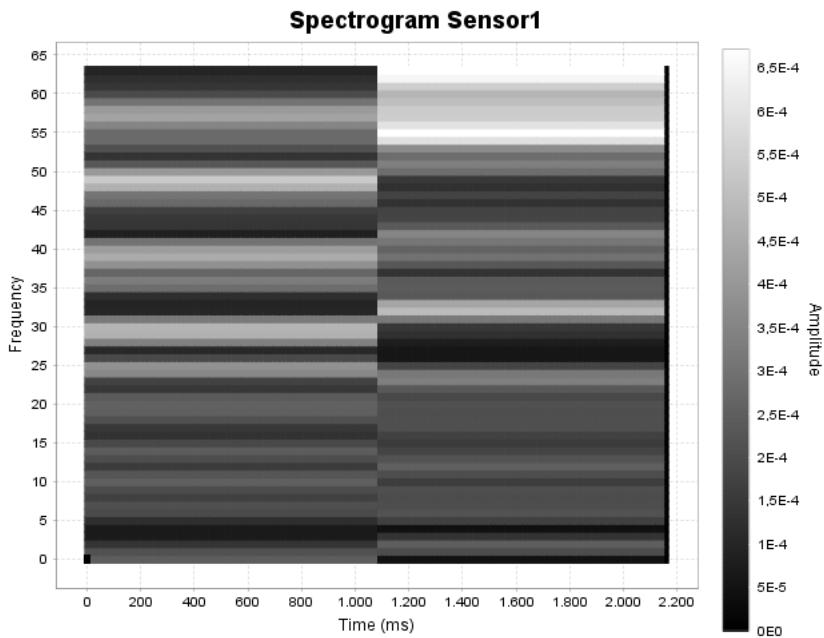
Gambar B.10: Hasil Spectrogram Sensor5 dengan topologi star dan window *Rectangle* di Atap

B.1.2 Hanning Window

- Sensor1

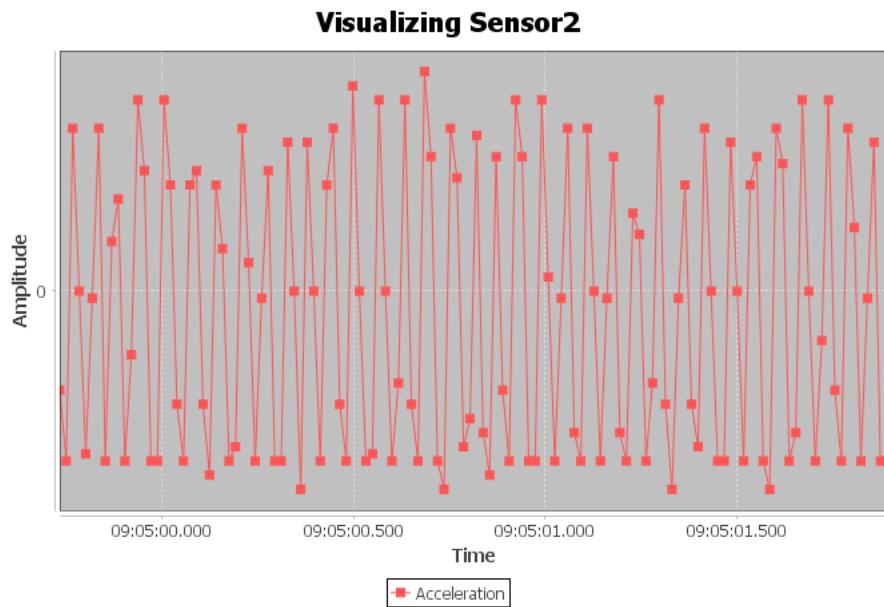


Gambar B.11: Hasil Sensing Sensor1 dengan topologi star dan window *Hanning* di Atap

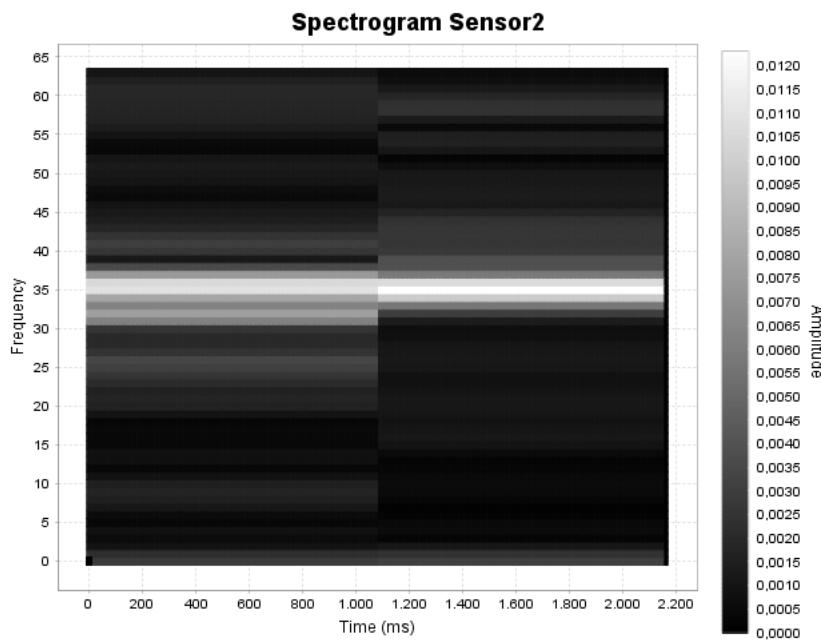


Gambar B.12: Hasil Spectrogram Sensor1 dengan topologi star dan window *Hanning* di Atap

- Sensor2

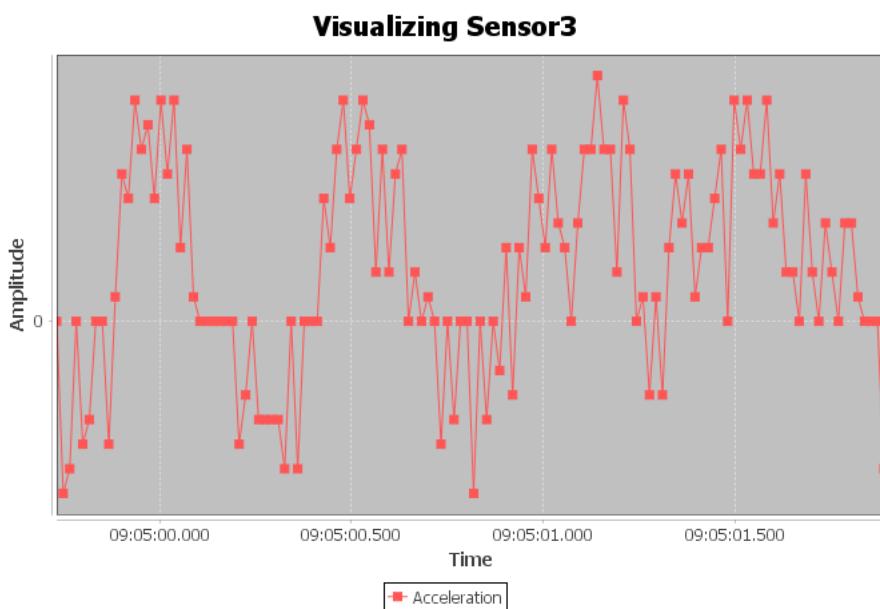


Gambar B.13: Hasil Sensing Sensor2 dengan topologi star dan window *Hanning* di Atap

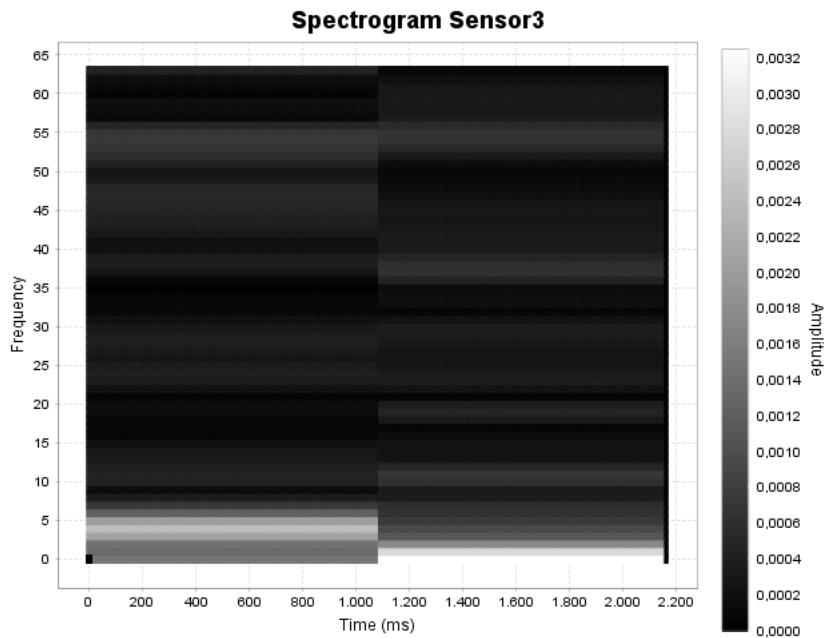


Gambar B.14: Hasil Spectrogram Sensor2 dengan topologi star dan window *Hanning* di Atap

- Sensor3

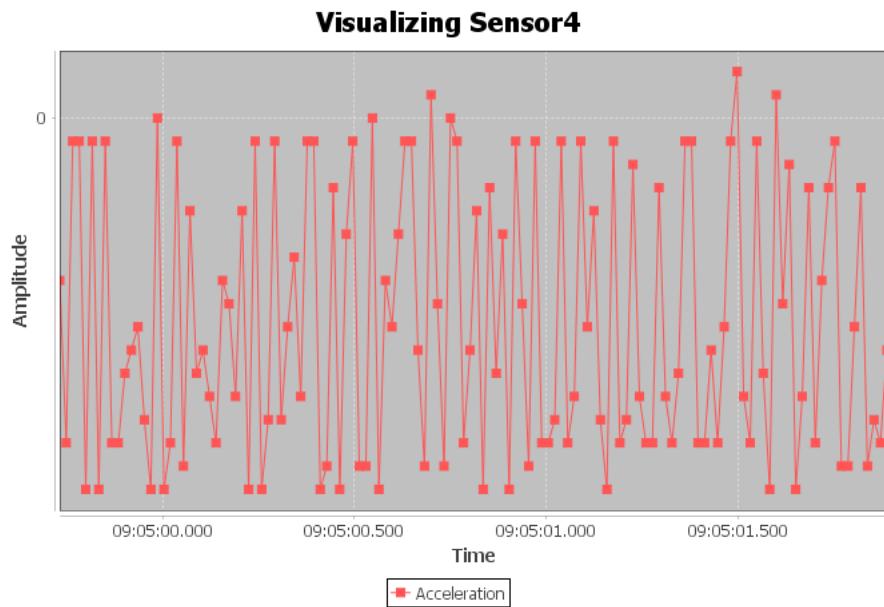


Gambar B.15: Hasil Sensing Sensor3 dengan topologi star dan window *Hanning* di Atap

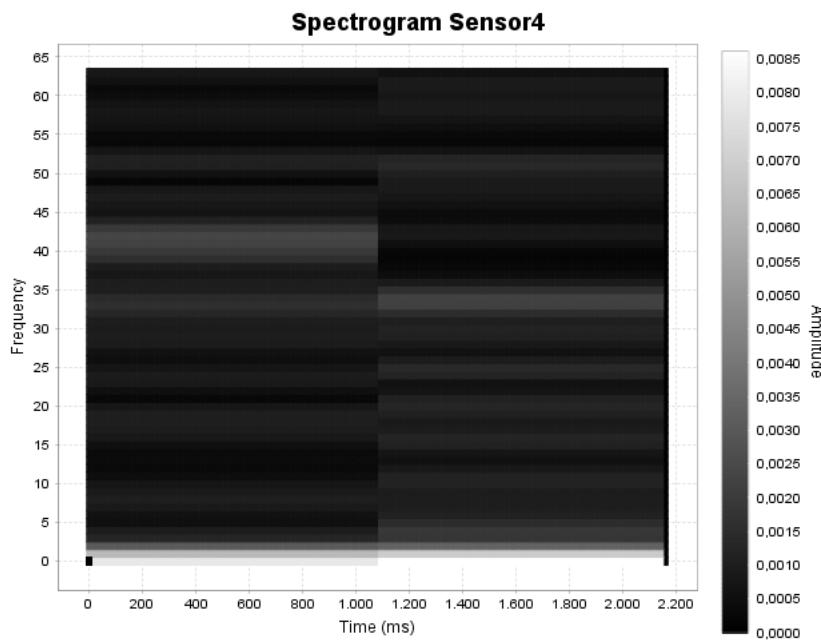


Gambar B.16: Hasil Spectrogram Sensor3 dengan topologi star dan window *Hanning* di Atap

- Sensor4

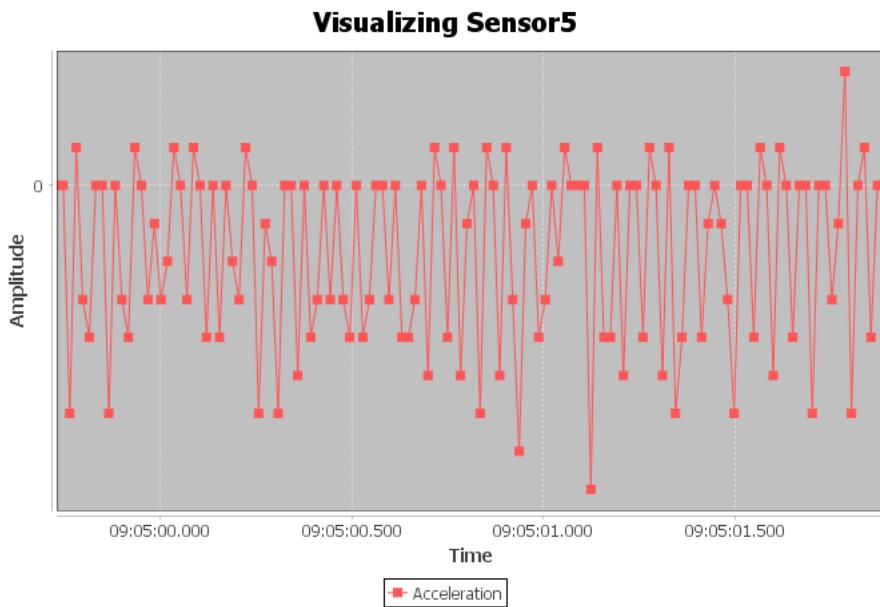


Gambar B.17: Hasil Sensing Sensor4 dengan topologi star dan window *Hanning* di Atap

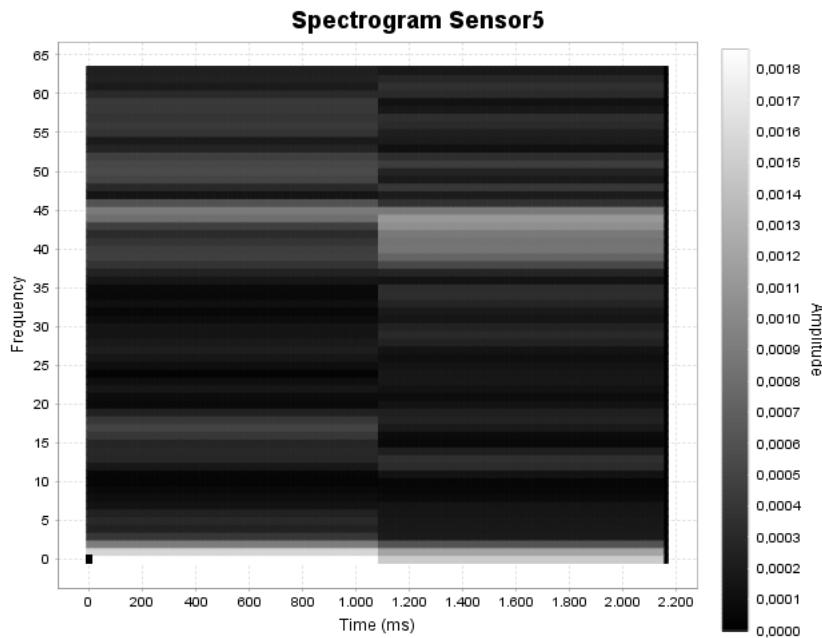


Gambar B.18: Hasil Spectrogram Sensor4 dengan topologi star dan window *Hanning* di Atap

- Sensor5



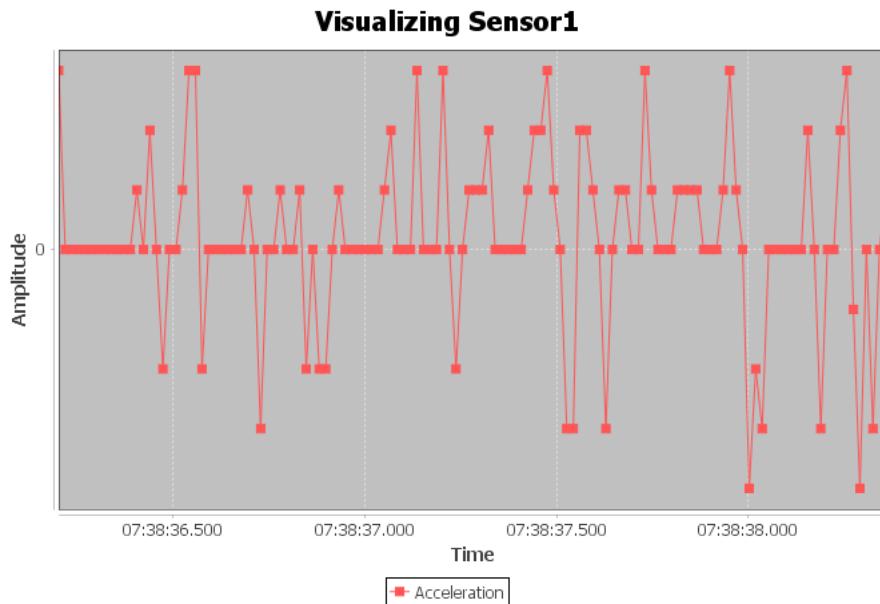
Gambar B.19: Hasil Sensing Sensor5 dengan topologi star dan window *Hanning* di Atap



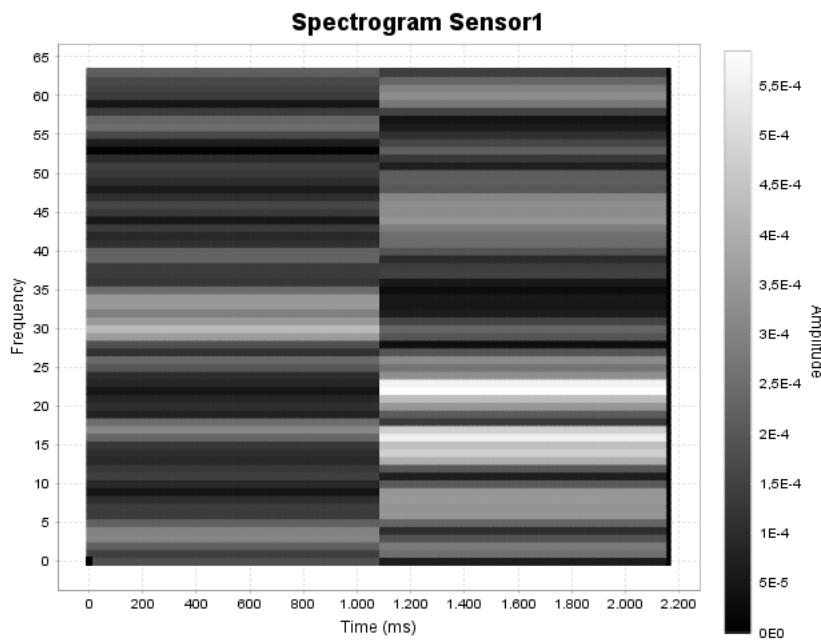
Gambar B.20: Hasil Spectrogram Sensor5 dengan topologi star dan window *Hanning* di Atap

B.1.3 Hamming Window

- Sensor1

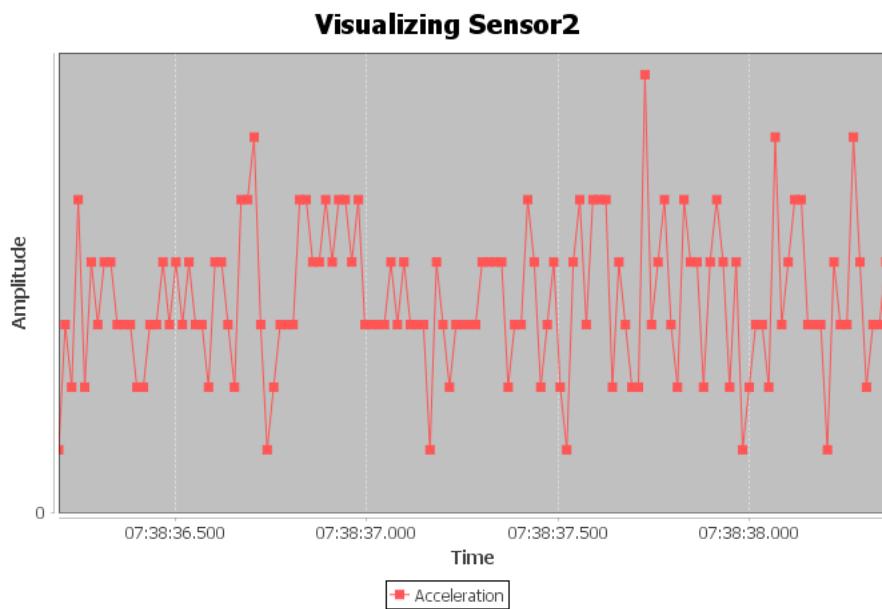


Gambar B.21: Hasil Sensing Sensor1 dengan topologi star dan window *Hamming* di Atap

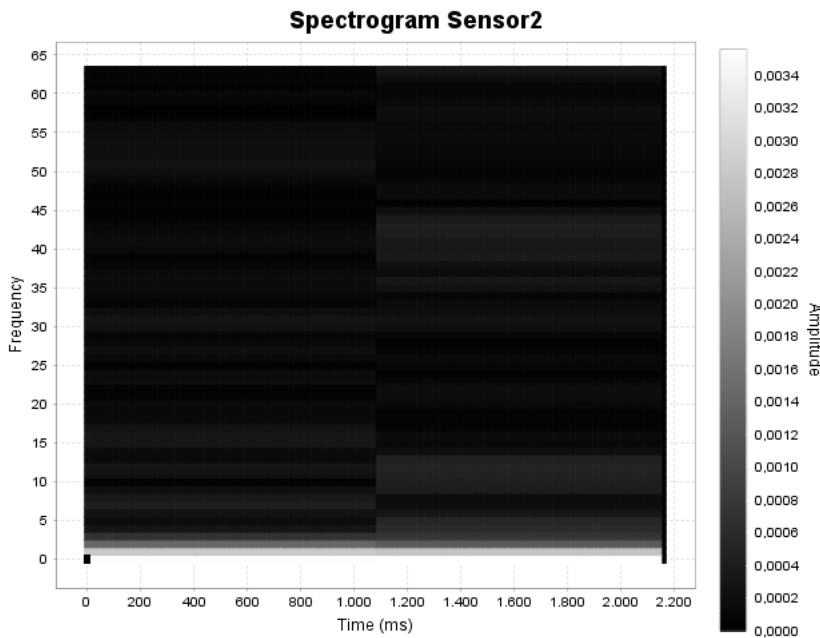


Gambar B.22: Hasil Spectrogram Sensor1 dengan topologi star dan window *Hamming* di Atap

- Sensor2

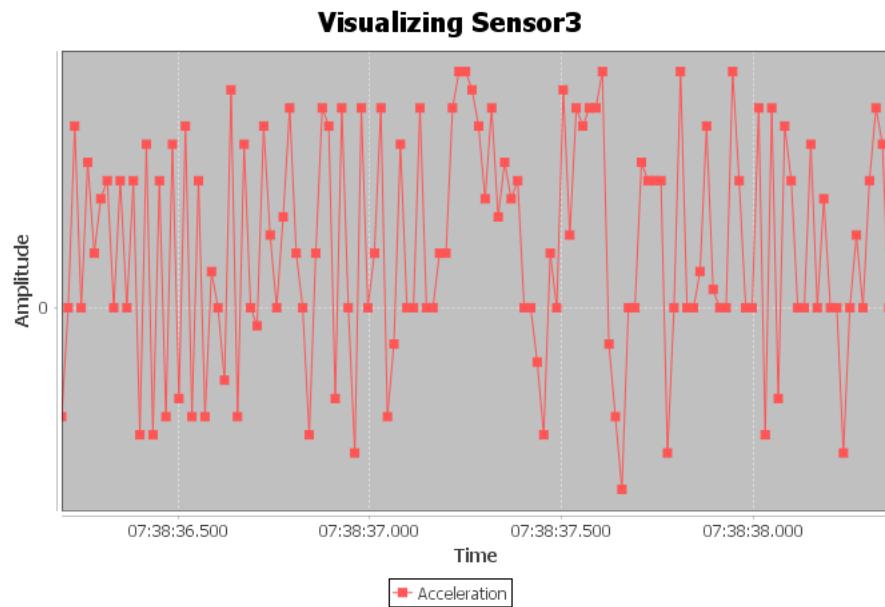


Gambar B.23: Hasil Sensing Sensor2 dengan topologi star dan window *Hamming* di Atap

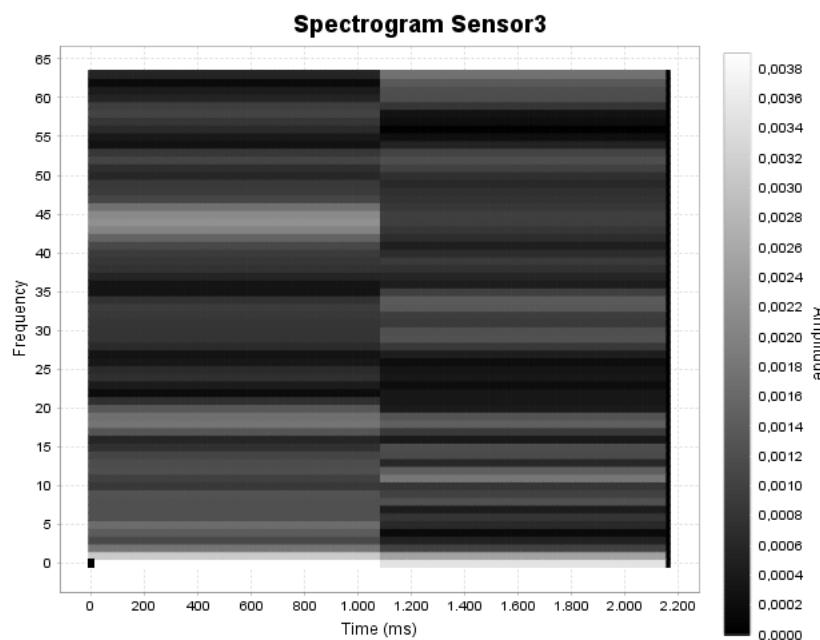


Gambar B.24: Hasil Spectrogram Sensor2 dengan topologi star dan window *Hamming* di Atap

- Sensor3

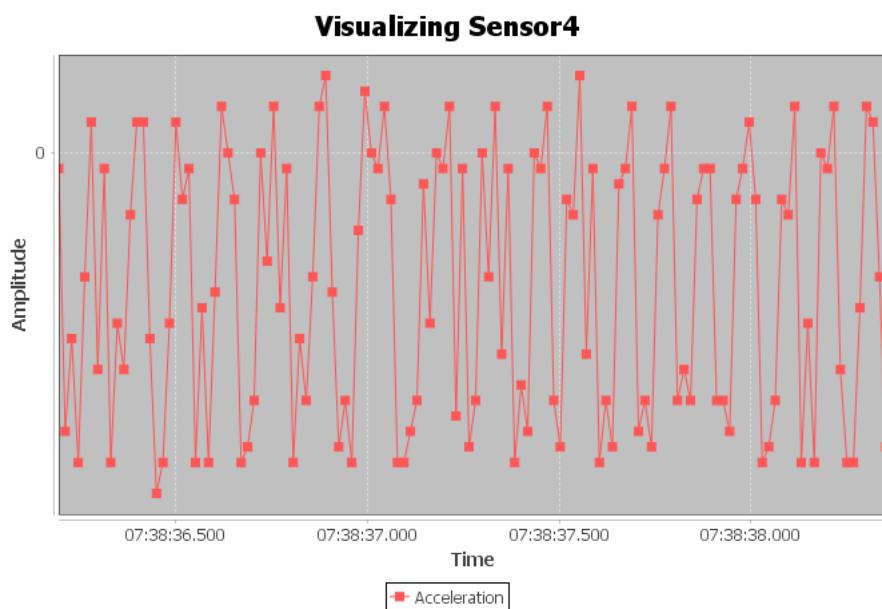


Gambar B.25: Hasil Sensing Sensor3 dengan topologi star dan window *Hamming* di Atap

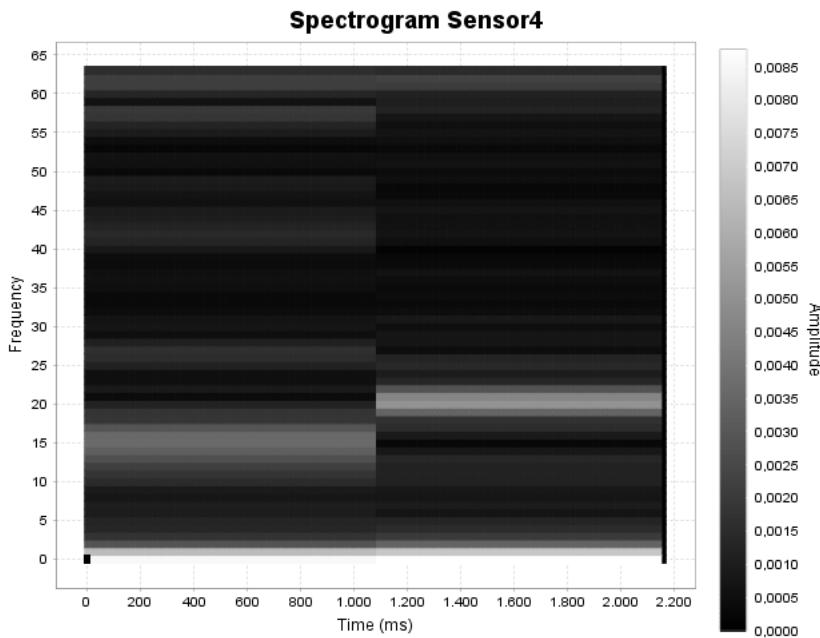


Gambar B.26: Hasil Spectrogram Sensor3 dengan topologi star dan window *Hamming* di Atap

- Sensor4

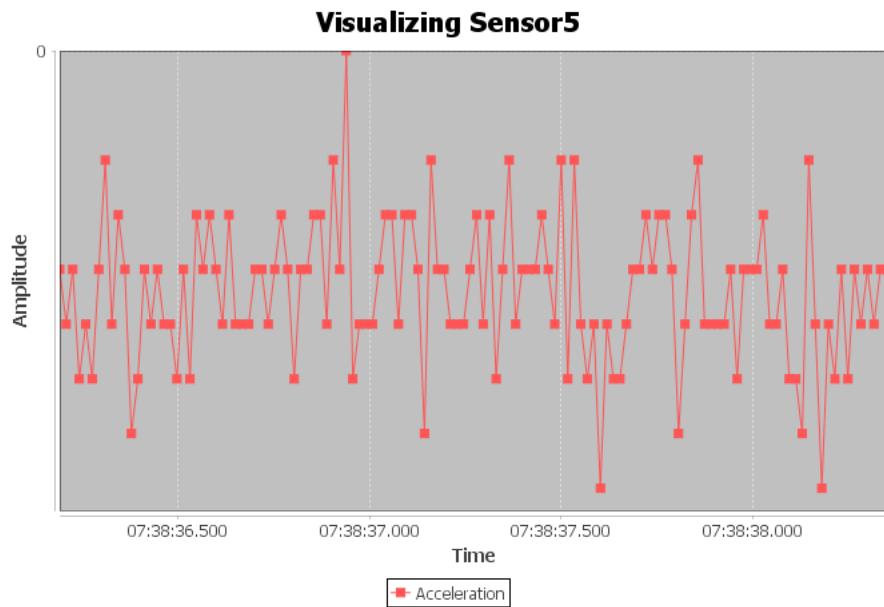


Gambar B.27: Hasil Sensing Sensor4 dengan topologi star dan window *Hamming* di Atap

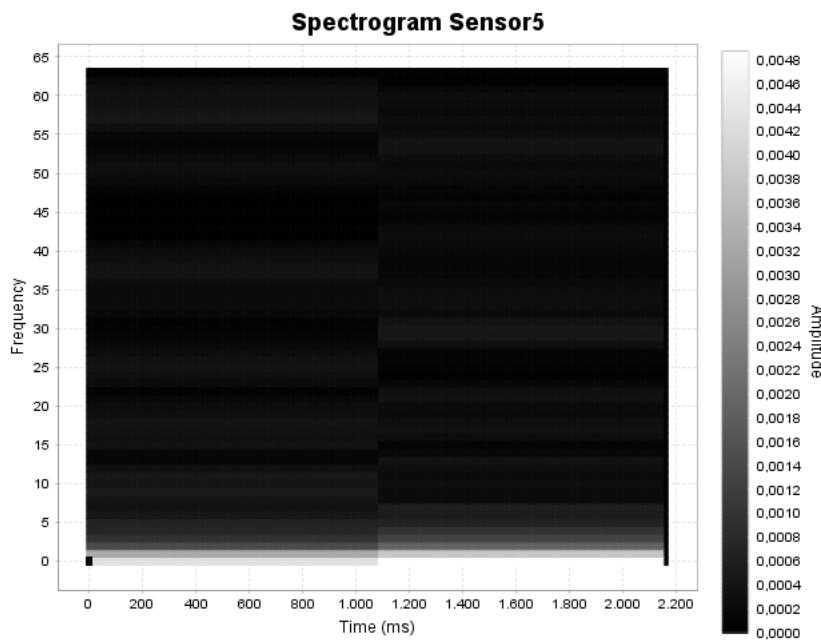


Gambar B.28: Hasil Spectrogram Sensor4 dengan topologi star dan window *Hamming* di Atap

- Sensor5



Gambar B.29: Hasil Sensing Sensor5 dengan topologi star dan window *Hamming* di Atap

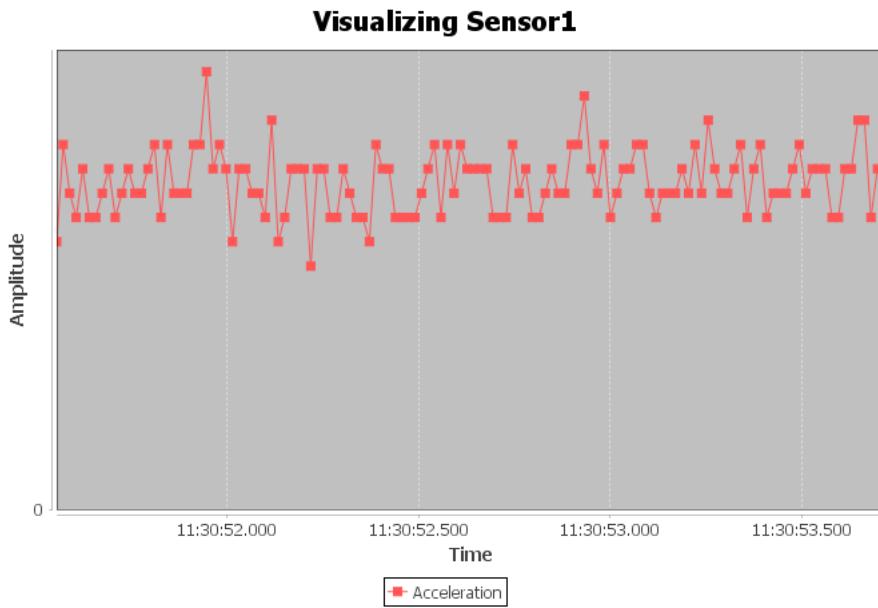


Gambar B.30: Hasil Spectrogram Sensor5 dengan topologi star dan window *Hamming* di Atap

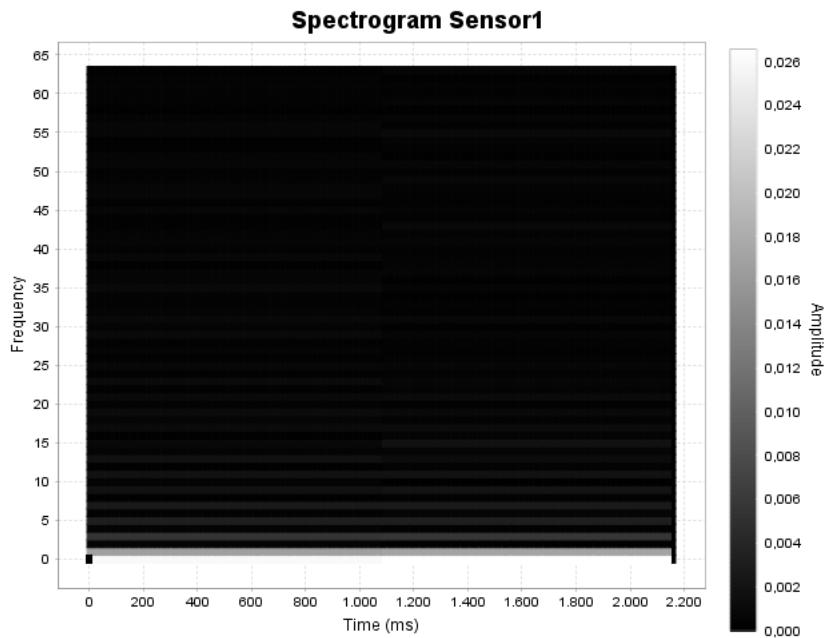
B.2 Topologi Tree

B.2.1 Rectangle Window

- Sensor1

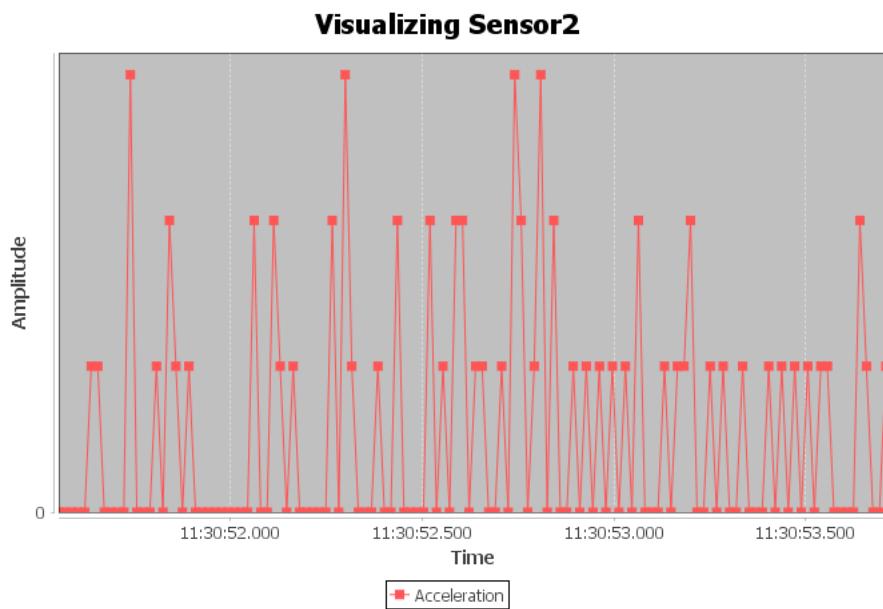


Gambar B.31: Hasil Sensing Sensor1 dengan topologi tree dan window *Rectangle* di Atap

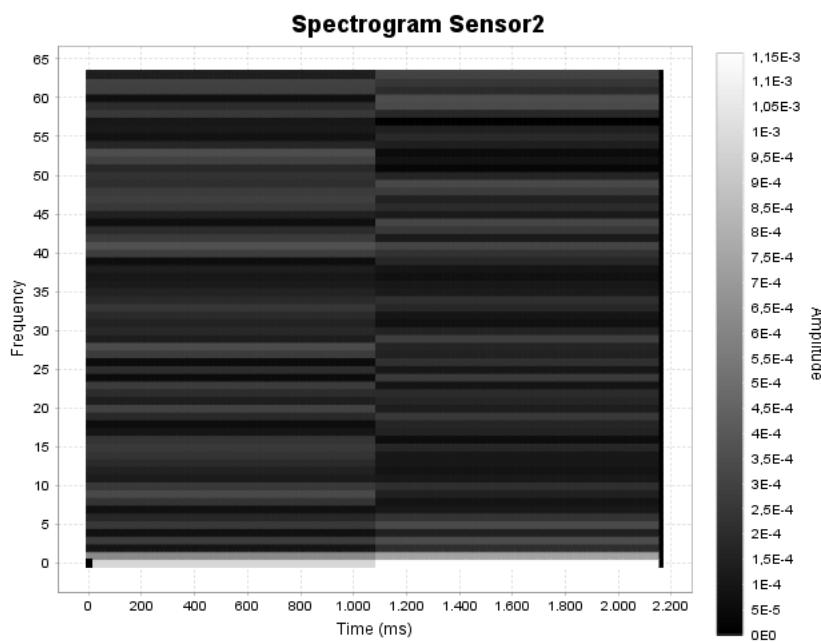


Gambar B.32: Hasil Spectrogram Sensor1 dengan topologi tree dan window *Rectangle* di Atap

- Sensor2

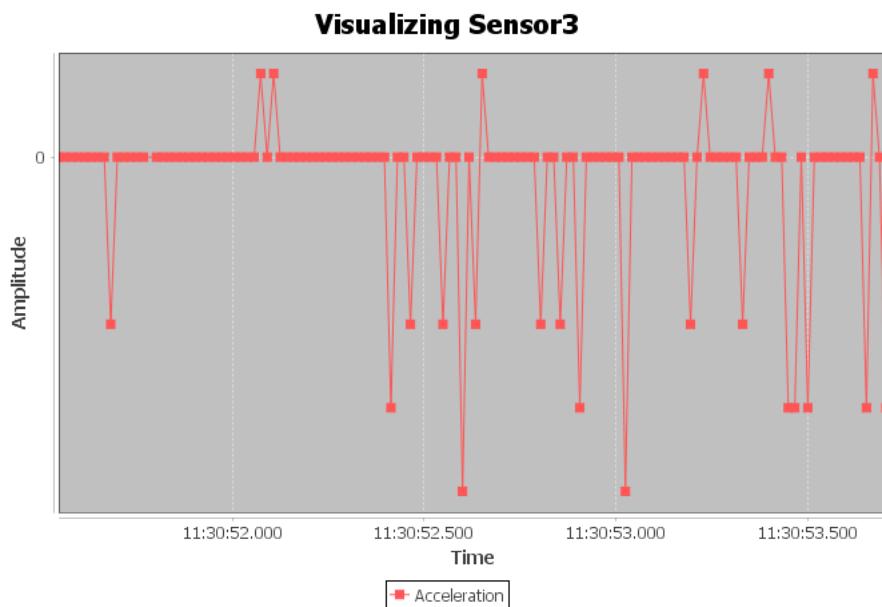


Gambar B.33: Hasil Sensing Sensor2 dengan topologi tree dan window *Rectangle* di Atap

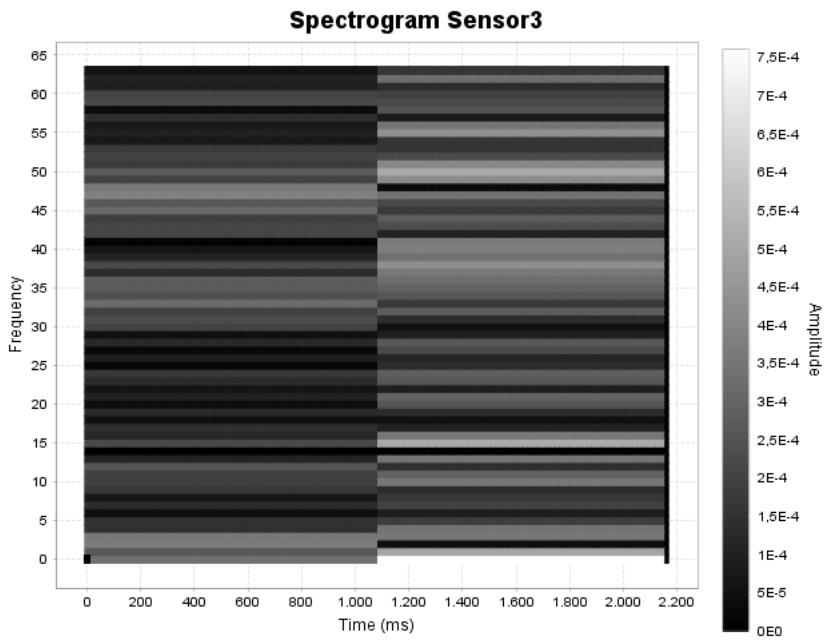


Gambar B.34: Hasil Spectrogram Sensor2 dengan topologi tree dan window *Rectangle* di Atap

- Sensor3

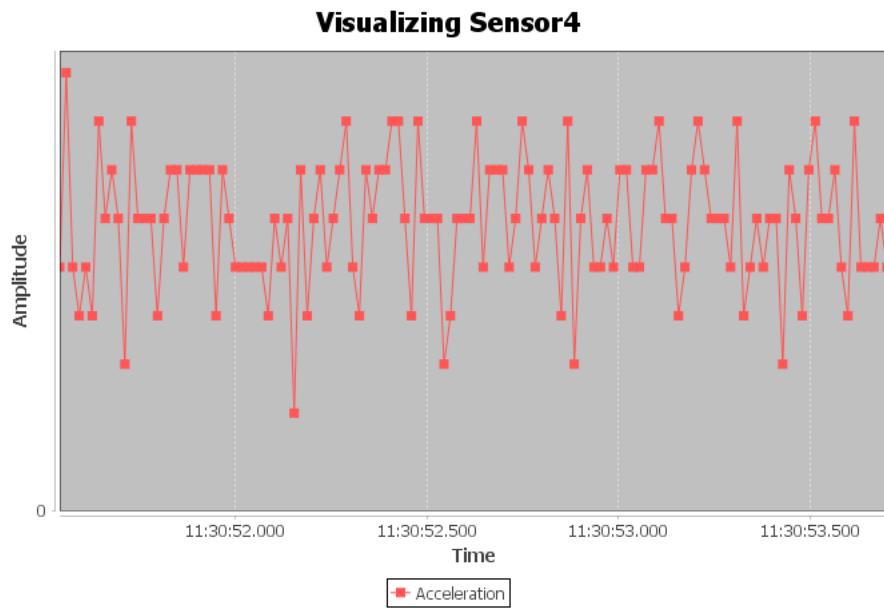


Gambar B.35: Hasil Sensing Sensor3 dengan topologi tree dan window *Rectangle* di Atap

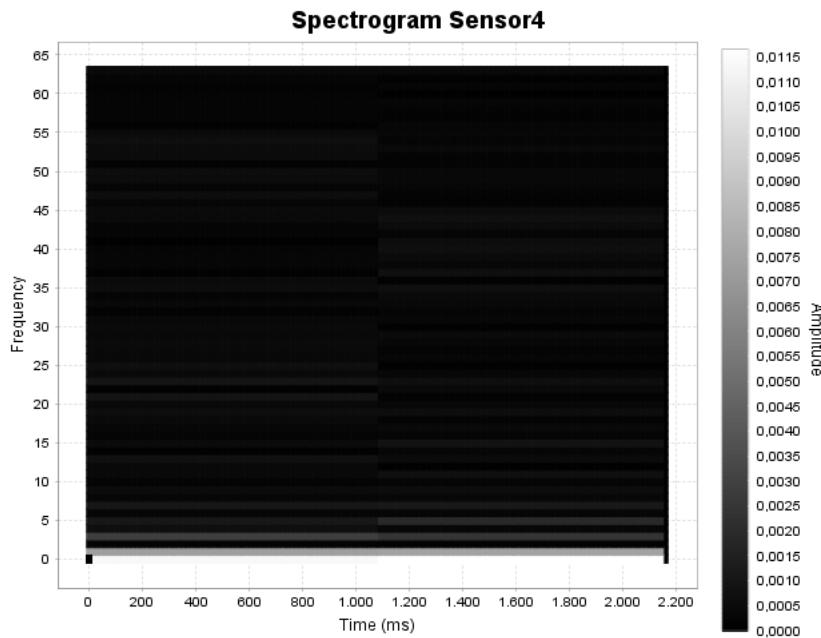


Gambar B.36: Hasil Spectrogram Sensor3 dengan topologi tree dan window *Rectangle* di Atap

- Sensor4

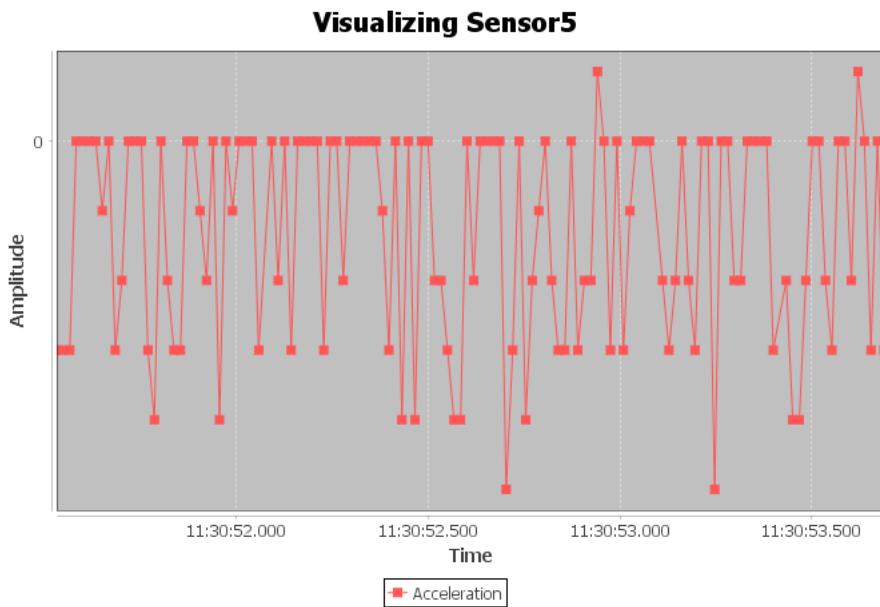


Gambar B.37: Hasil Sensing Sensor4 dengan topologi tree dan window *Rectangle* di Atap

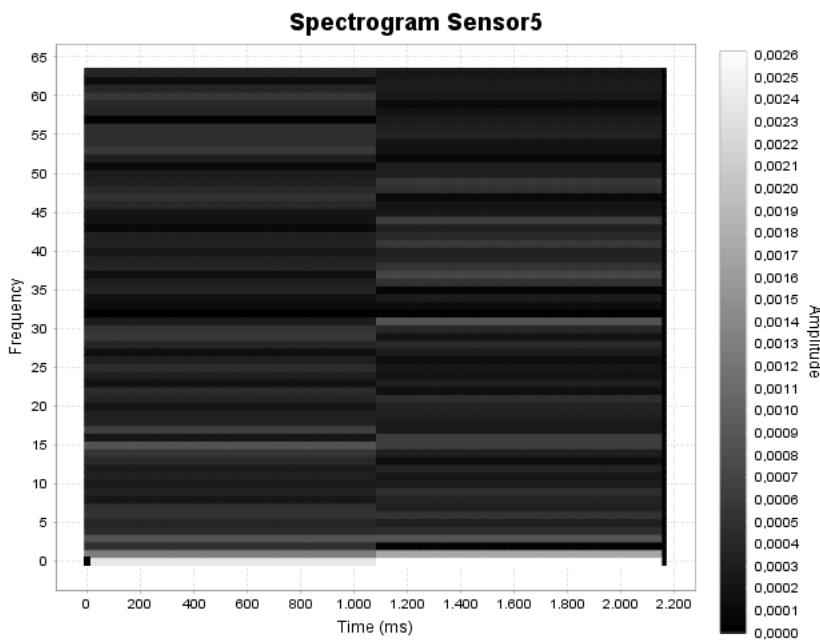


Gambar B.38: Hasil Spectrogram Sensor4 dengan topologi tree dan window *Rectangle* di Atap

- Sensor5



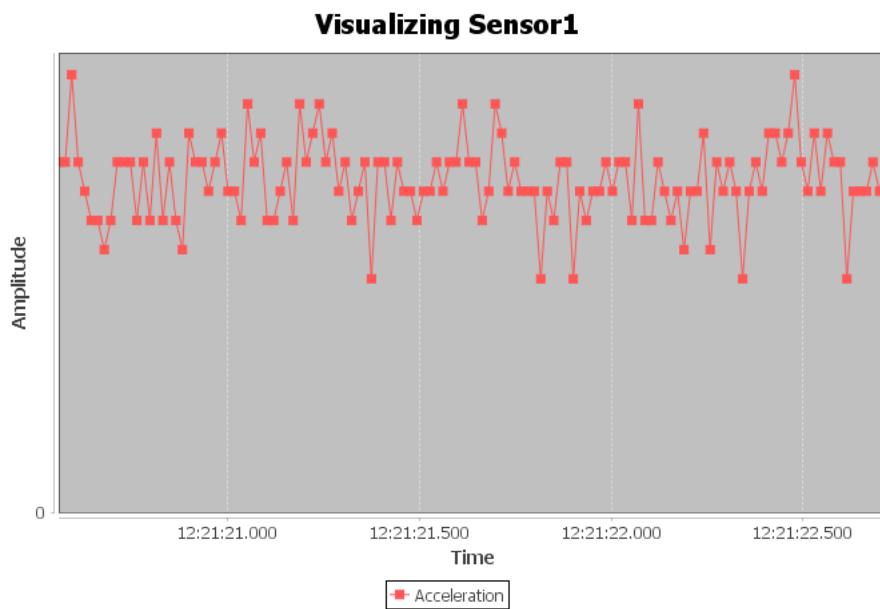
Gambar B.39: Hasil Sensing Sensor5 dengan topologi tree dan window *Rectangle* di Atap



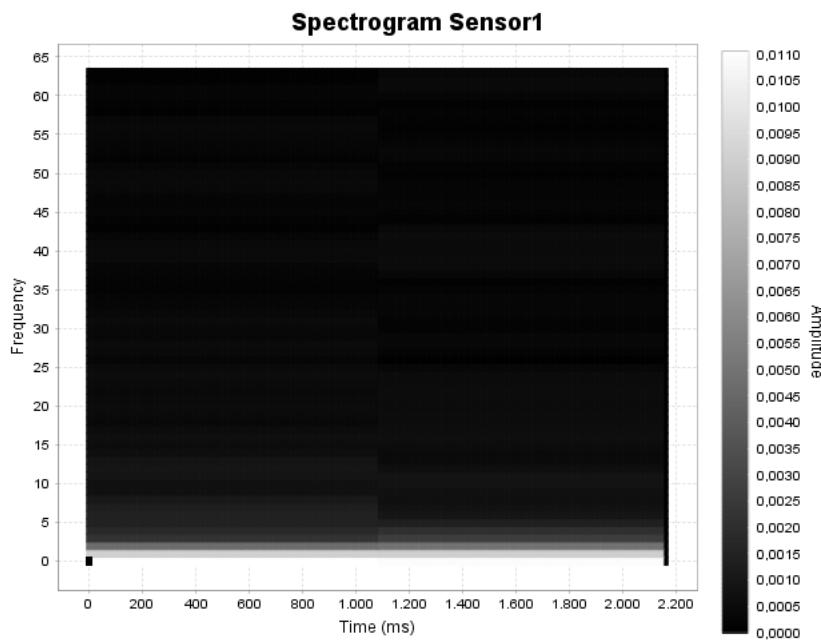
Gambar B.40: Hasil Spectrogram Sensor5 dengan topologi tree dan window *Rectangle* di Atap

B.2.2 Hanning Window

- Sensor1

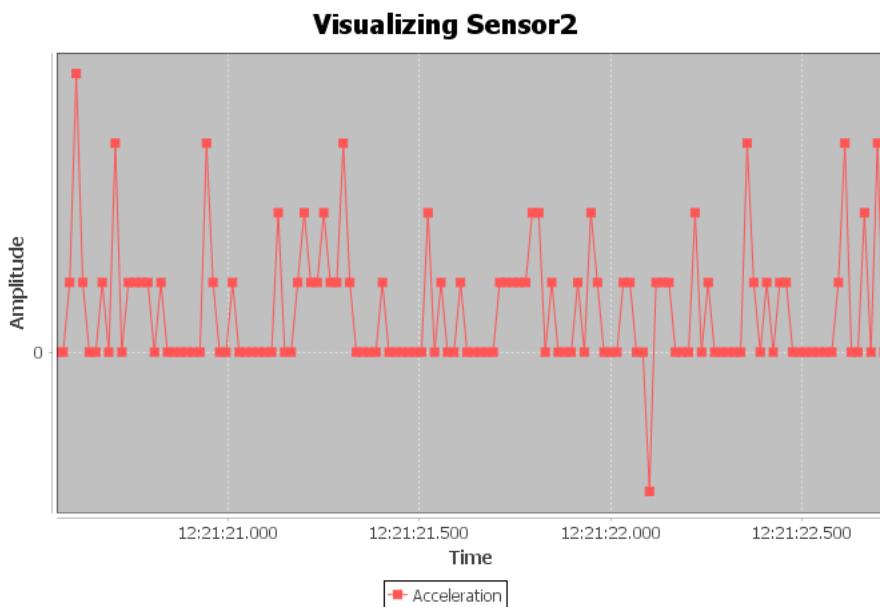


Gambar B.41: Hasil Sensing Sensor1 dengan topologi tree dan window *Hanning* di Atap

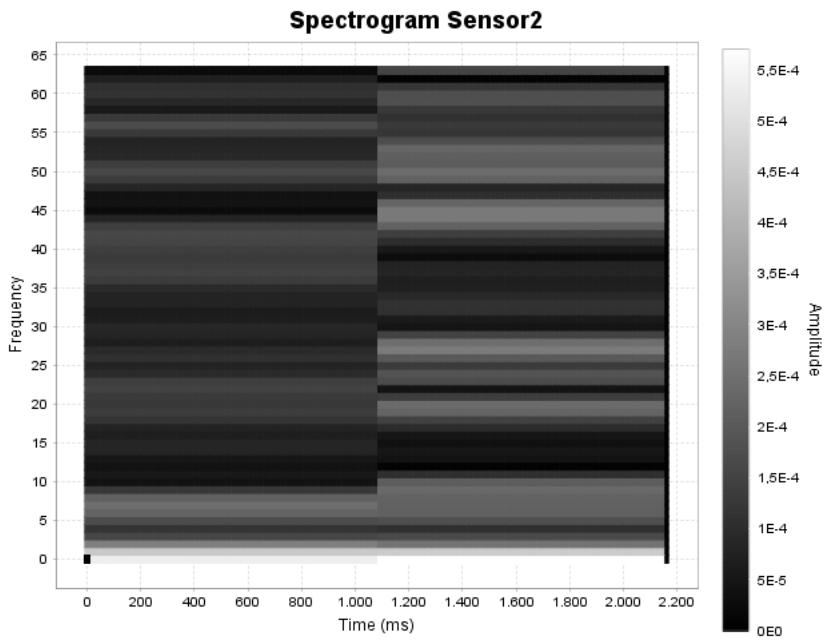


Gambar B.42: Hasil Spectrogram Sensor1 dengan topologi tree dan window *Hanning* di Atap

- Sensor2

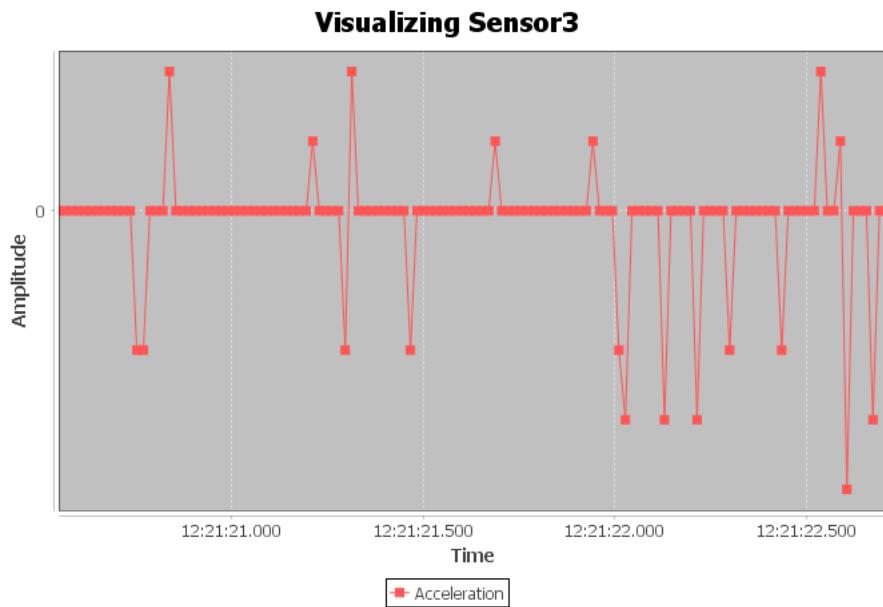


Gambar B.43: Hasil Sensing Sensor2 dengan topologi tree dan window *Hanning* di Atap

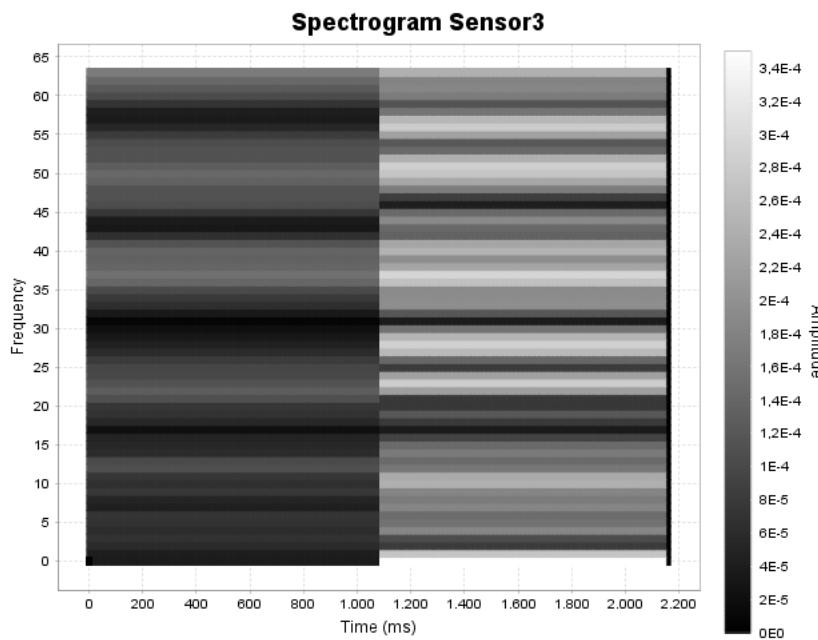


Gambar B.44: Hasil Spectrogram Sensor2 dengan topologi tree dan window *Hanning* di Atap

- Sensor3

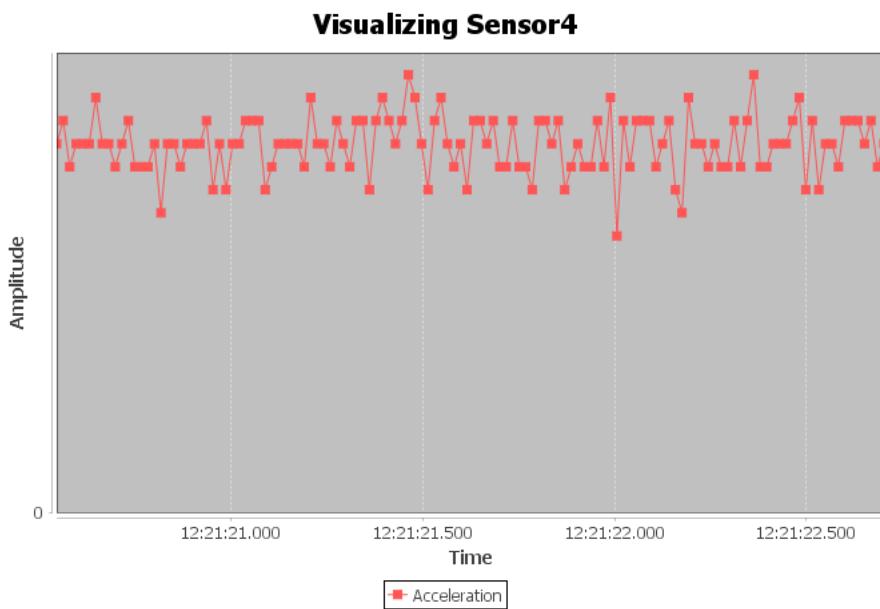


Gambar B.45: Hasil Sensing Sensor3 dengan topologi tree dan window *Hanning* di Atap

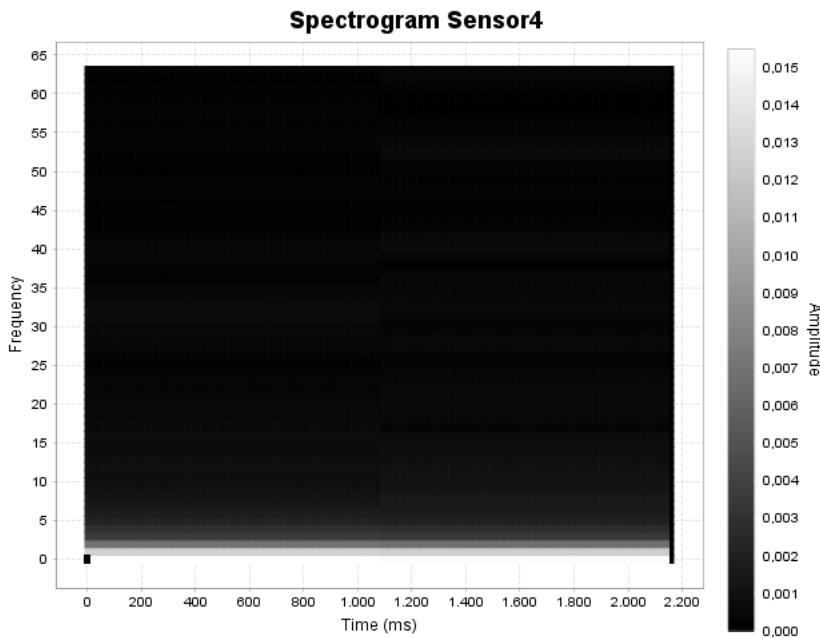


Gambar B.46: Hasil Spectrogram Sensor3 dengan topologi tree dan window *Hanning* di Atap

- Sensor4

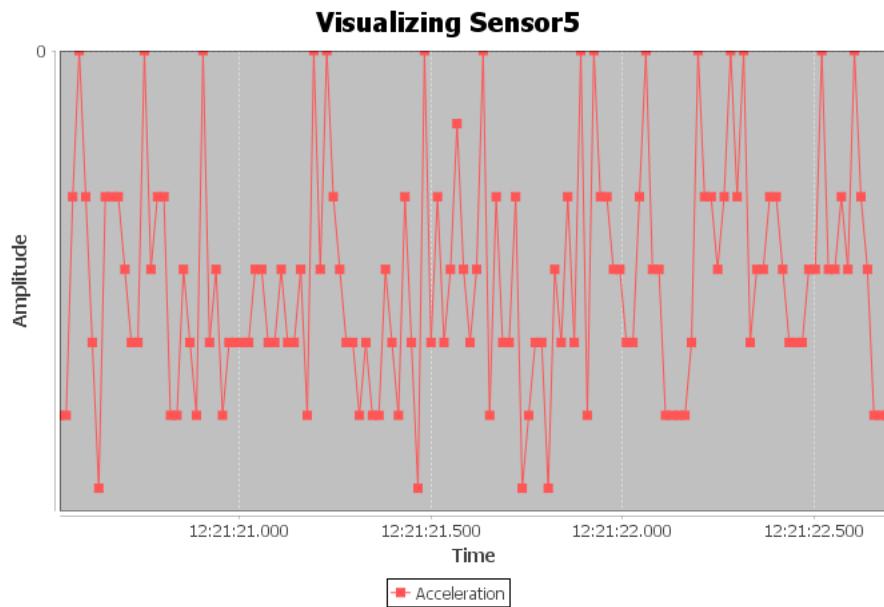


Gambar B.47: Hasil Sensing Sensor4 dengan topologi tree dan window *Hanning* di Atap

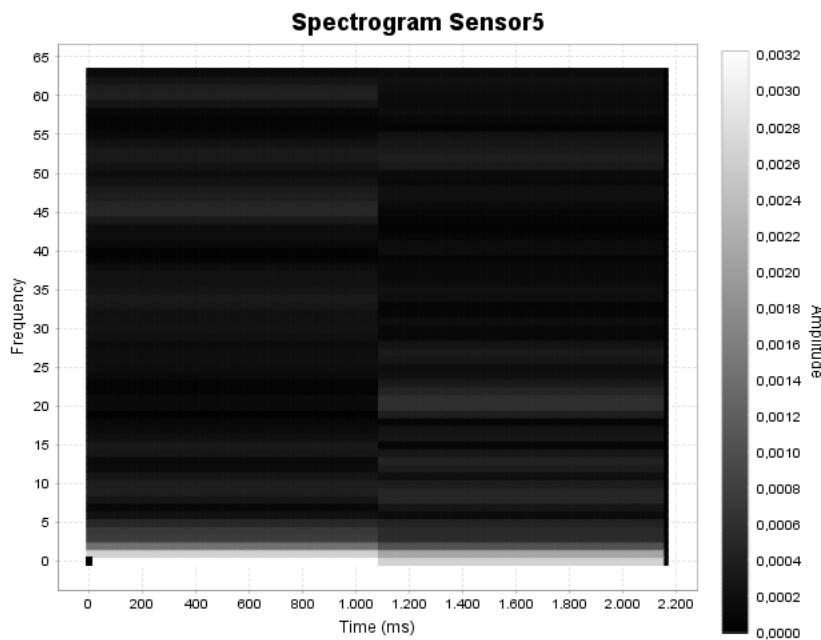


Gambar B.48: Hasil Spectrogram Sensor4 dengan topologi tree dan window *Hanning* di Atap

- Sensor5



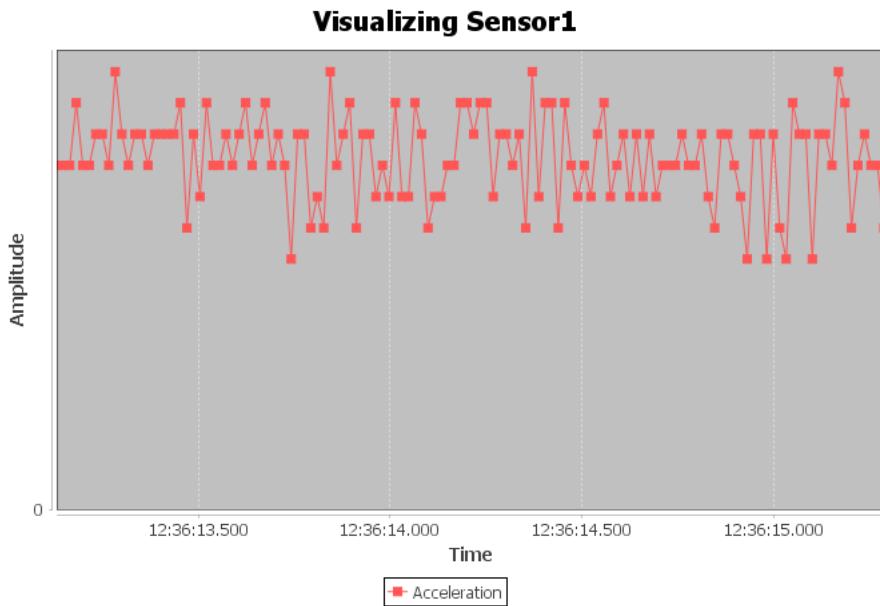
Gambar B.49: Hasil Sensing Sensor5 dengan topologi tree dan window *Hanning* di Atap



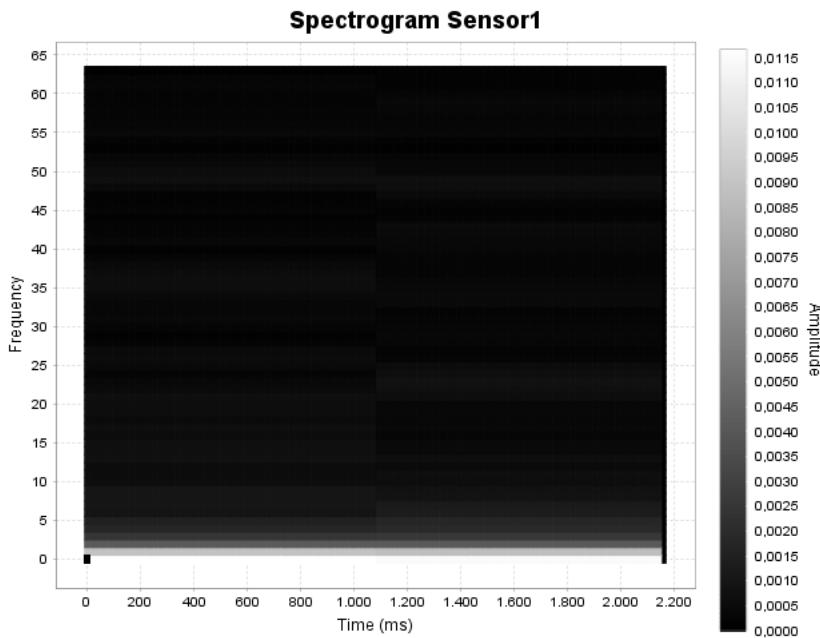
Gambar B.50: Hasil Spectrogram Sensor5 dengan topologi tree dan window *Hanning* di Atap

B.2.3 Hamming Window

- Sensor1

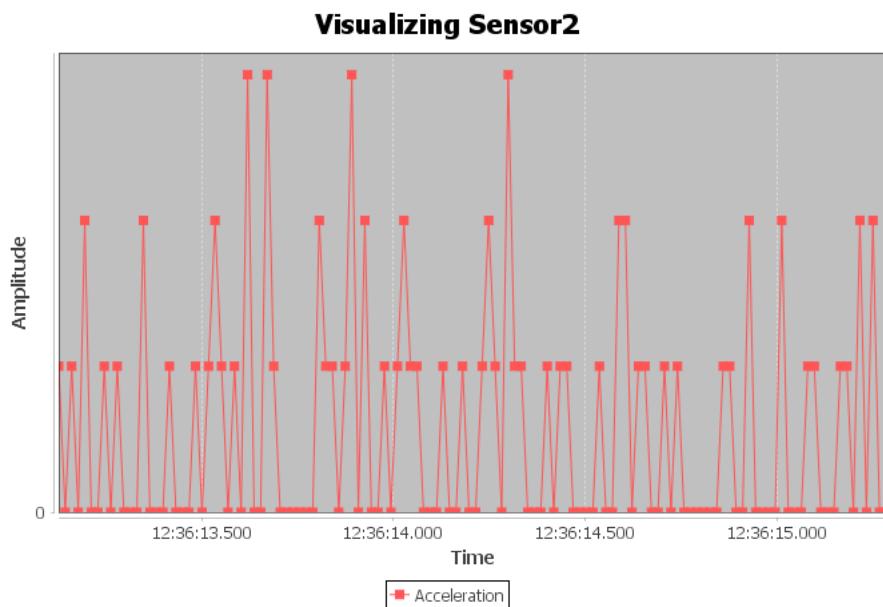


Gambar B.51: Hasil Sensing Sensor1 dengan topologi tree dan window *Hamming* di Atap

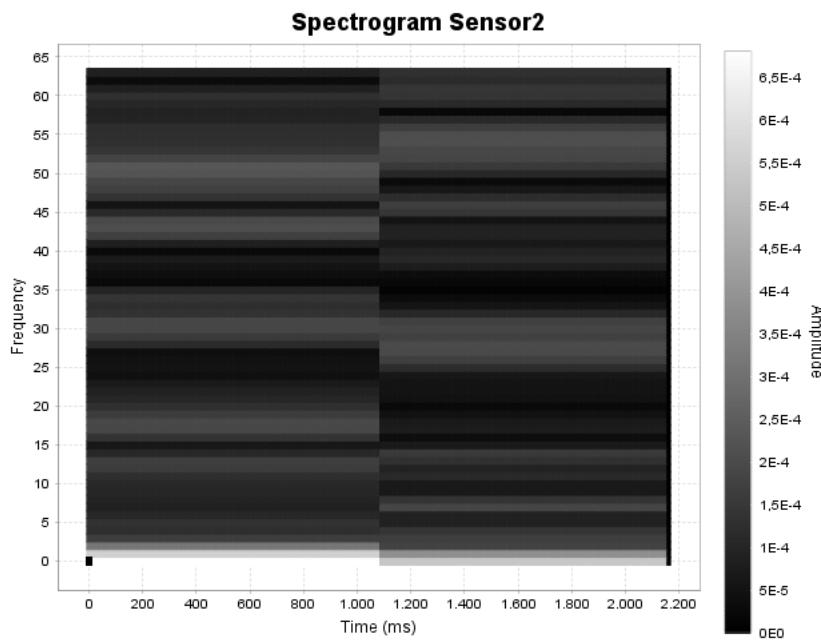


Gambar B.52: Hasil Spectrogram Sensor1 dengan topologi tree dan window *Hamming* di Atap

- Sensor2

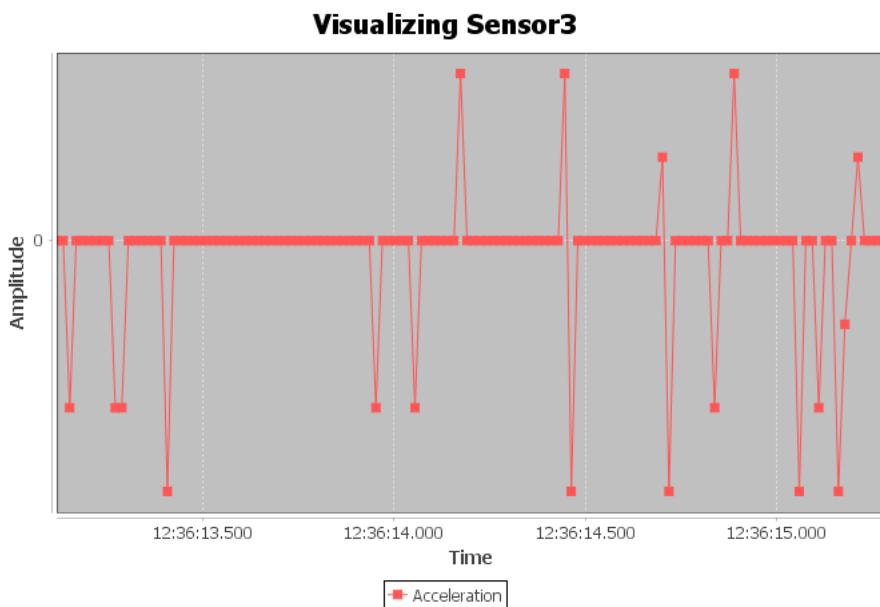


Gambar B.53: Hasil Sensing Sensor2 dengan topologi tree dan window *Hamming* di Atap

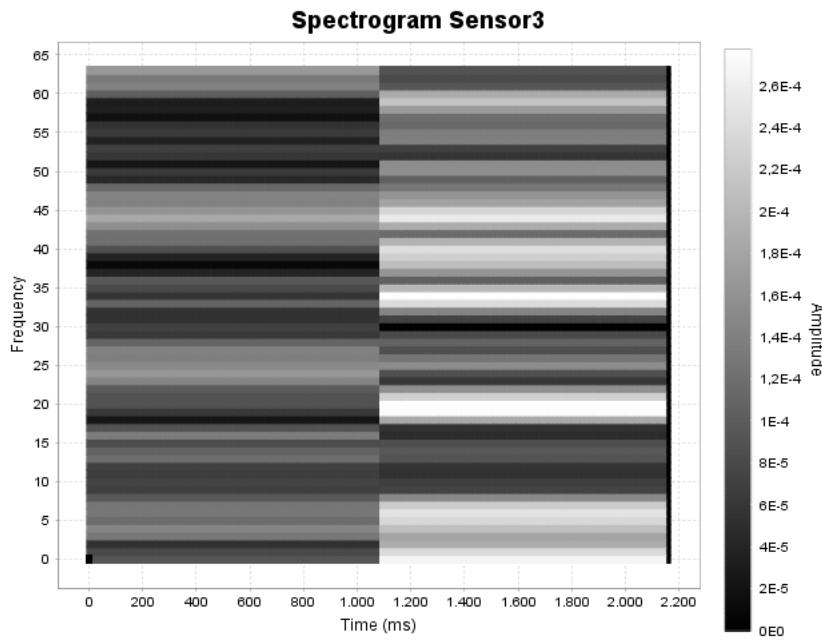


Gambar B.54: Hasil Spectrogram Sensor2 dengan topologi tree dan window *Hamming* di Atap

- Sensor3

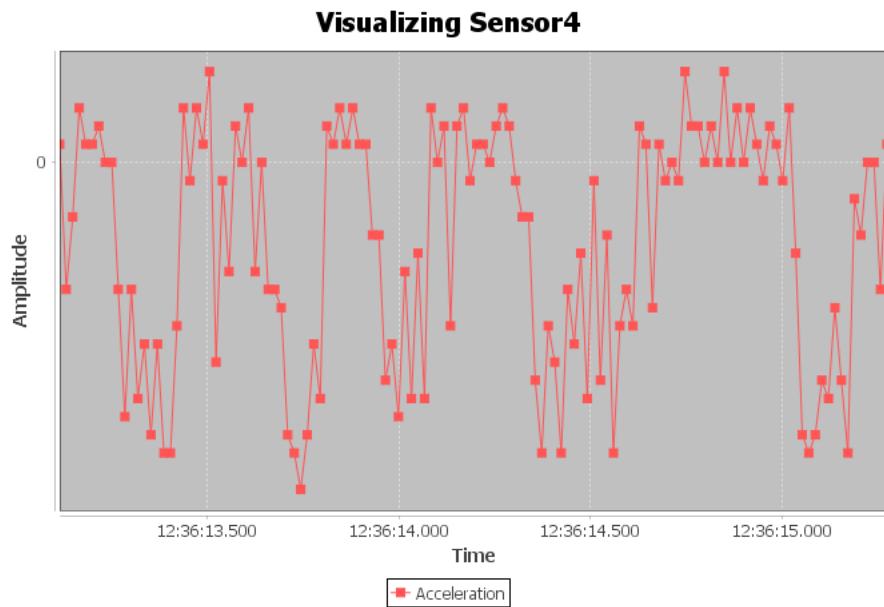


Gambar B.55: Hasil Sensing Sensor3 dengan topologi tree dan window *Hamming* di Atap

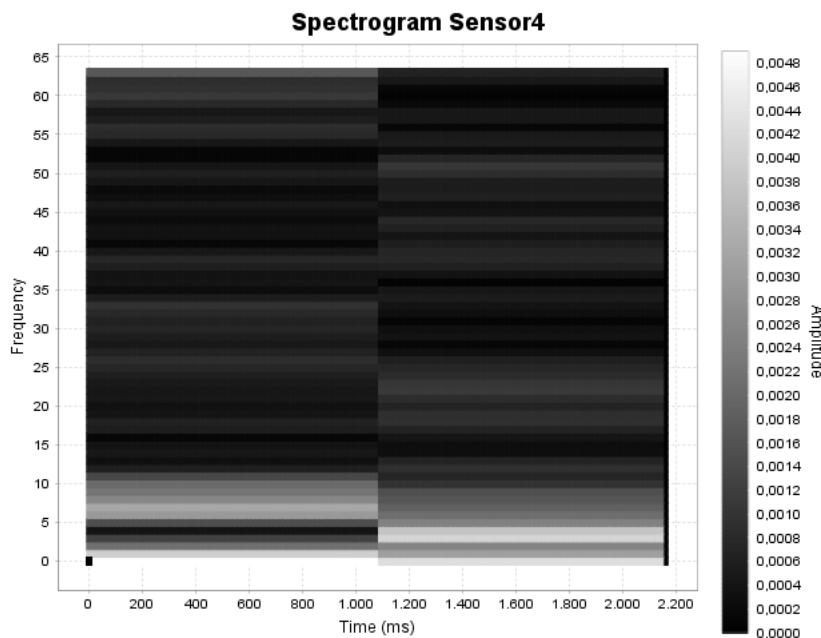


Gambar B.56: Hasil Spectrogram Sensor3 dengan topologi tree dan window *Hamming* di Atap

- Sensor4

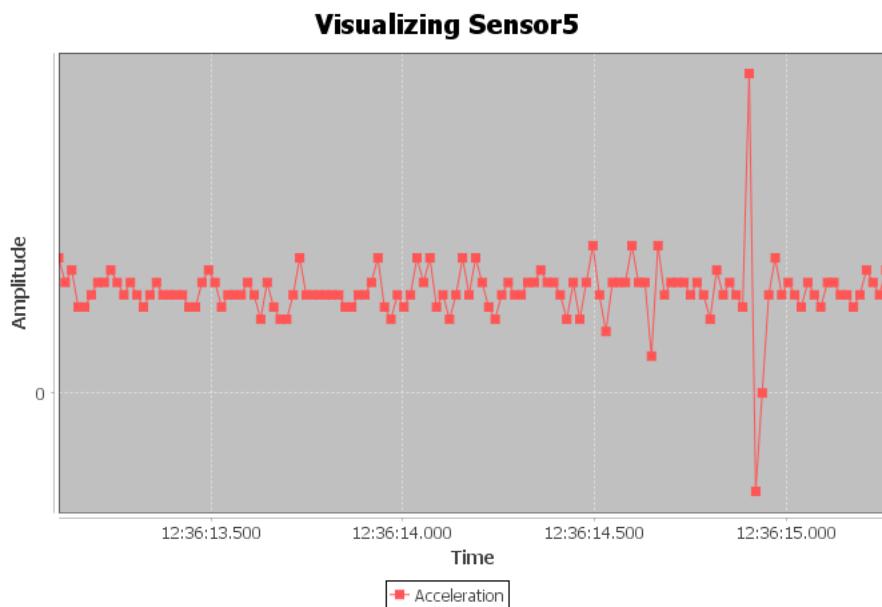


Gambar B.57: Hasil Sensing Sensor4 dengan topologi tree dan window *Hamming* di Atap

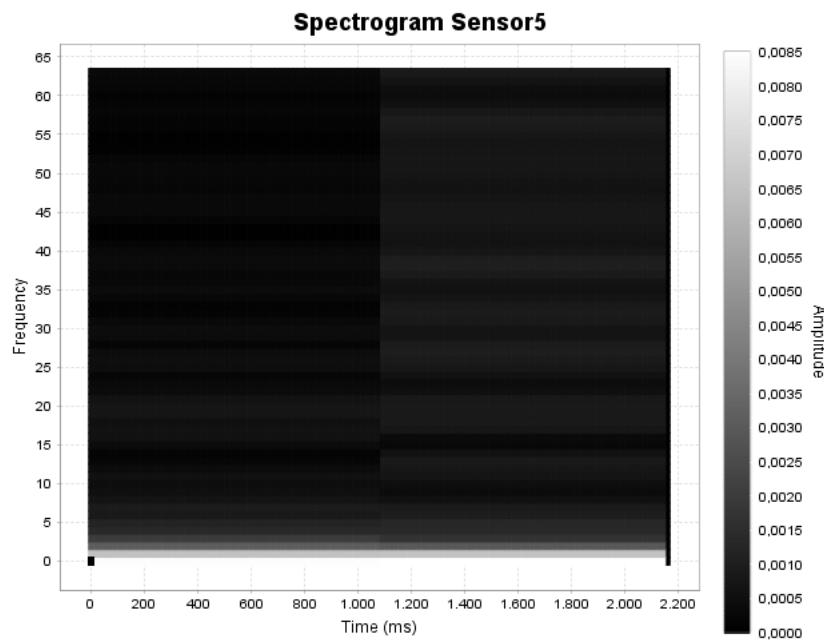


Gambar B.58: Hasil Spectrogram Sensor4 dengan topologi tree dan window *Hamming* di Atap

- Sensor5



Gambar B.59: Hasil Sensing Sensor5 dengan topologi tree dan window *Hamming* di Atap



Gambar B.60: Hasil Spectrogram Sensor5 dengan topologi tree dan window *Hamming* di Atap

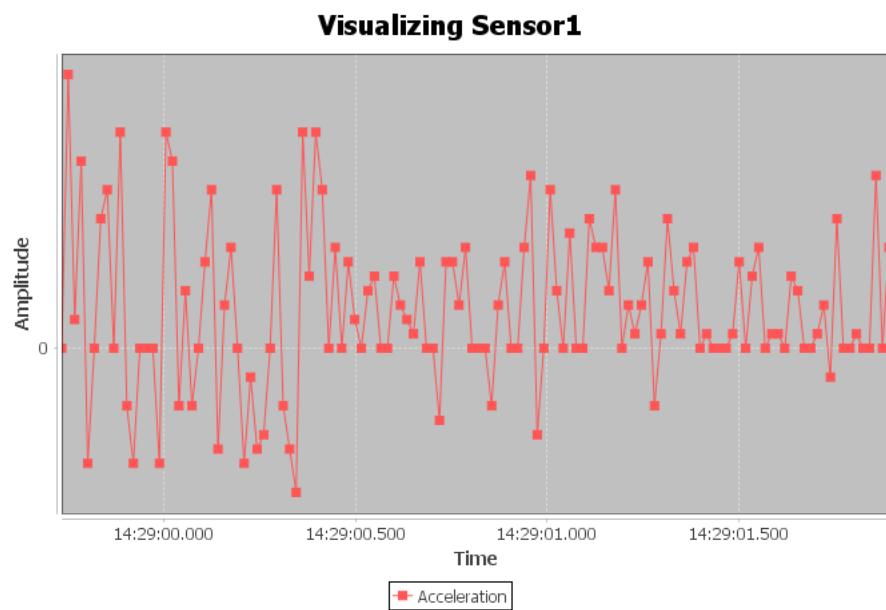
LAMPIRAN C

HASIL EKSPERIMEN DI JALAN

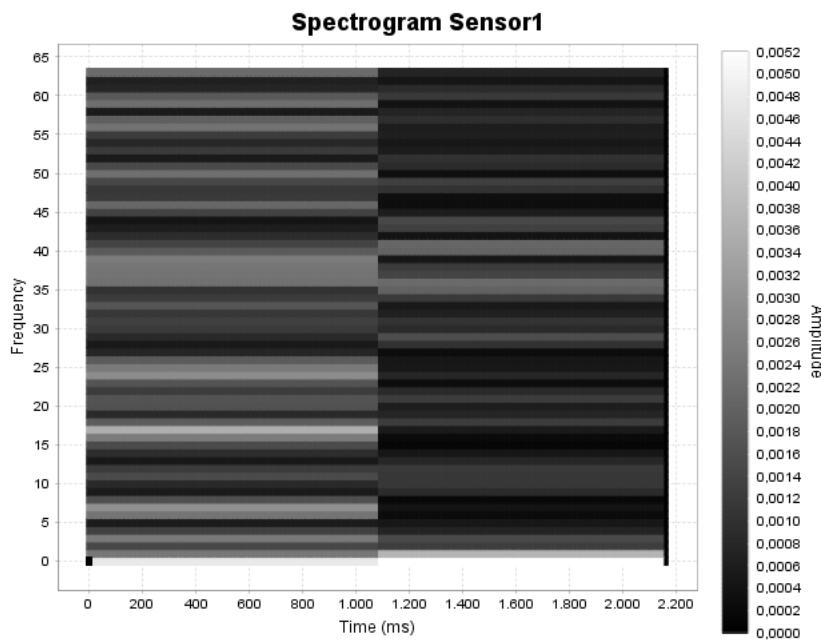
C.1 Topologi Star

C.1.1 Rectangle Window

- Sensor1

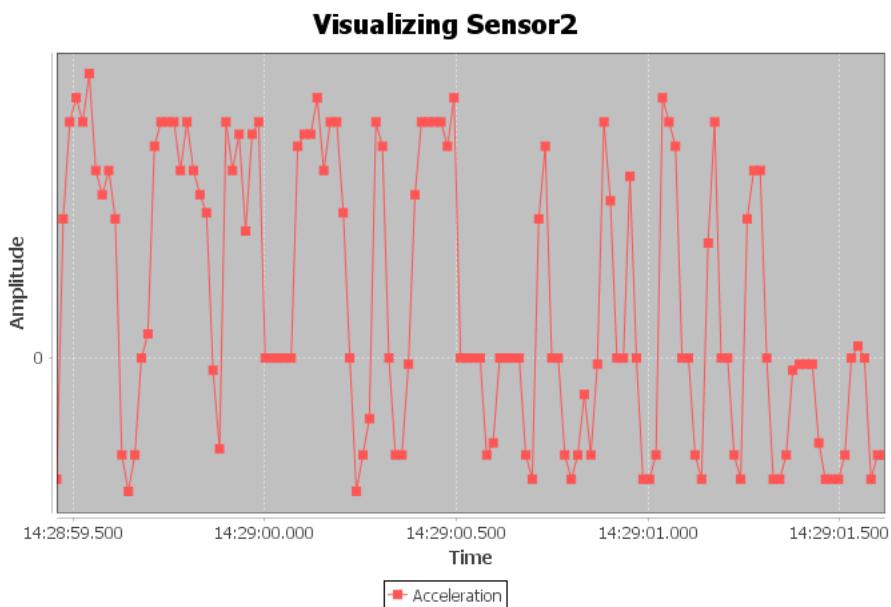


Gambar C.1: Hasil Sensing Sensor1 dengan topologi star dan window *Rectangle* di Jalan

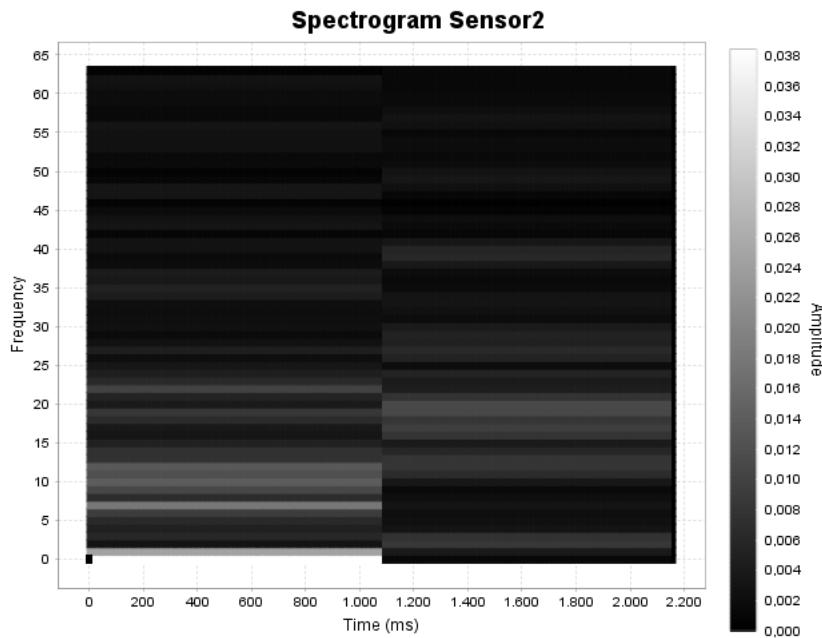


Gambar C.2: Hasil Spectrogram Sensor1 dengan topologi star dan window *Rectangle* di Jalan

- Sensor2

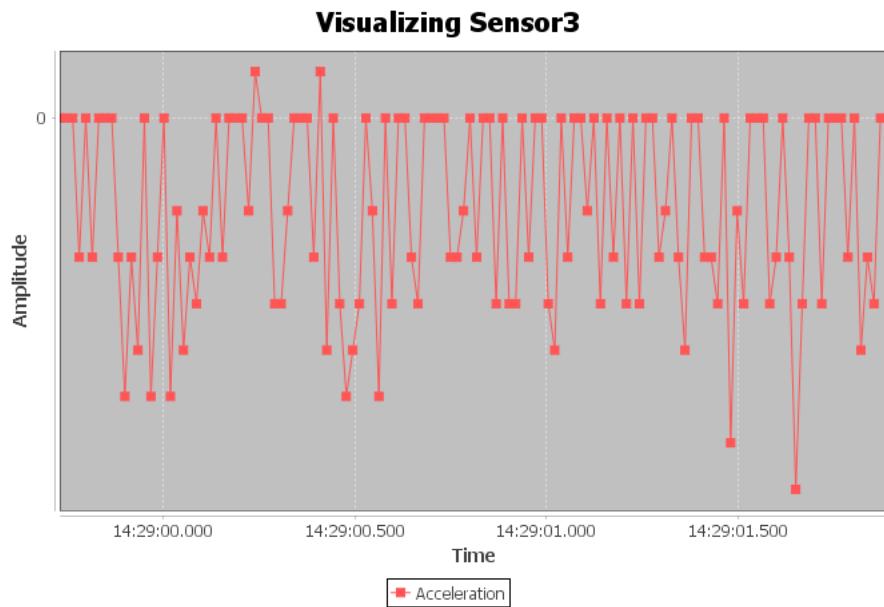


Gambar C.3: Hasil Sensing Sensor2 dengan topologi star dan window *Rectangle* di Jalan

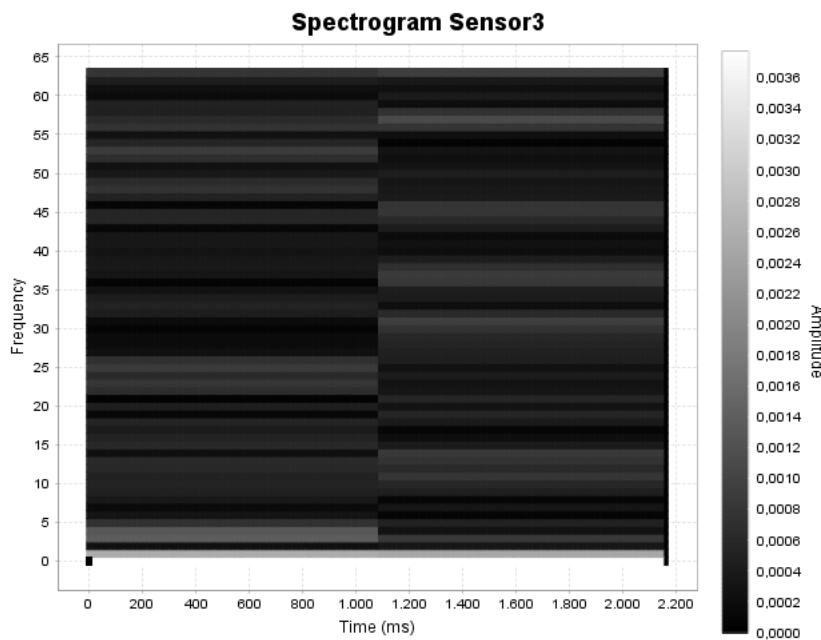


Gambar C.4: Hasil Spectrogram Sensor2 dengan topologi star dan window *Rectangle* di Jalan

- Sensor3

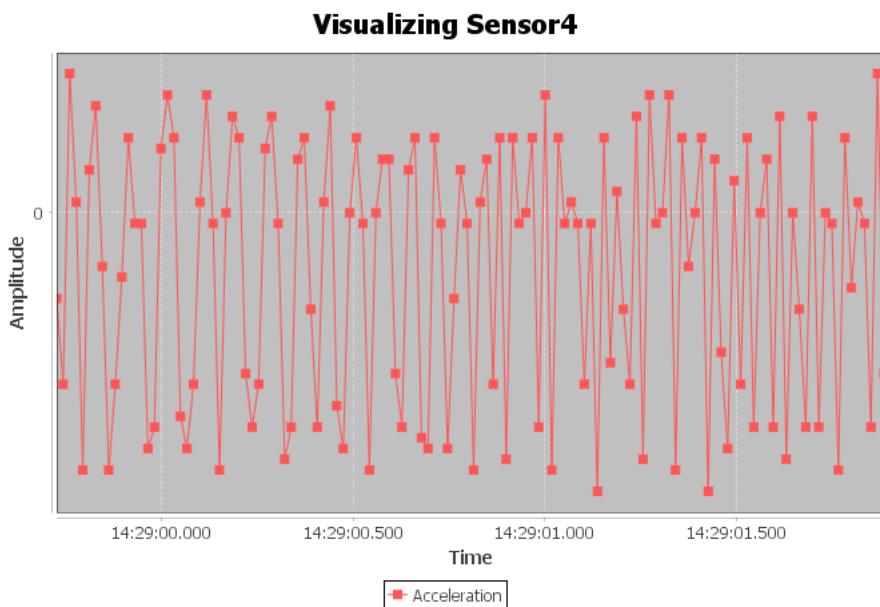


Gambar C.5: Hasil Sensing Sensor3 dengan topologi star dan window *Rectangle* di Jalan

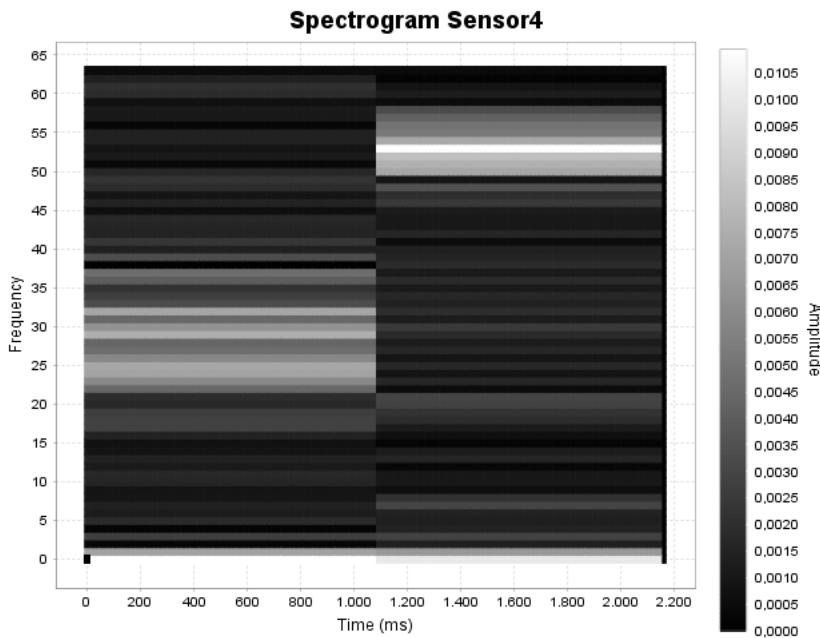


Gambar C.6: Hasil Spectrogram Sensor3 dengan topologi star dan window *Rectangle* di Jalan

- Sensor4

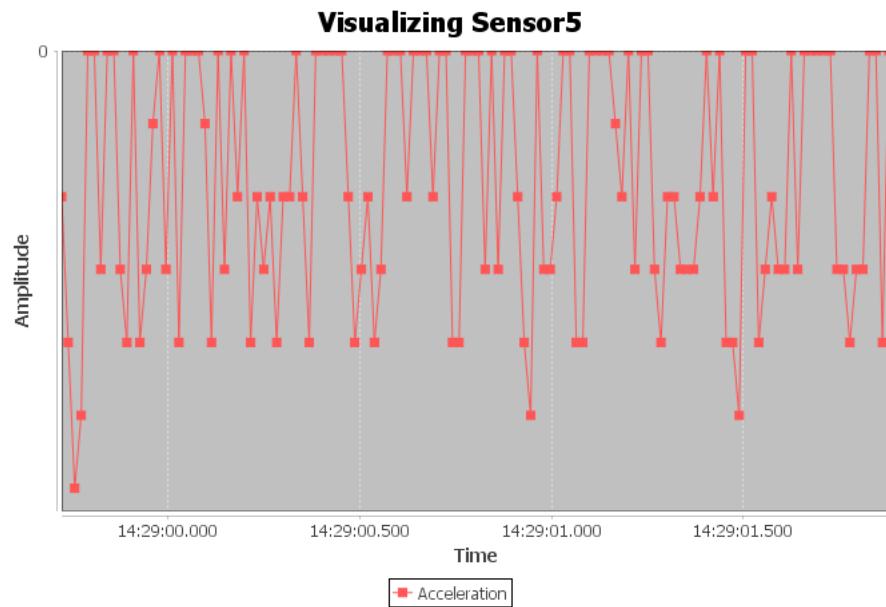


Gambar C.7: Hasil Sensing Sensor4 dengan topologi star dan window *Rectangle* di Jalan

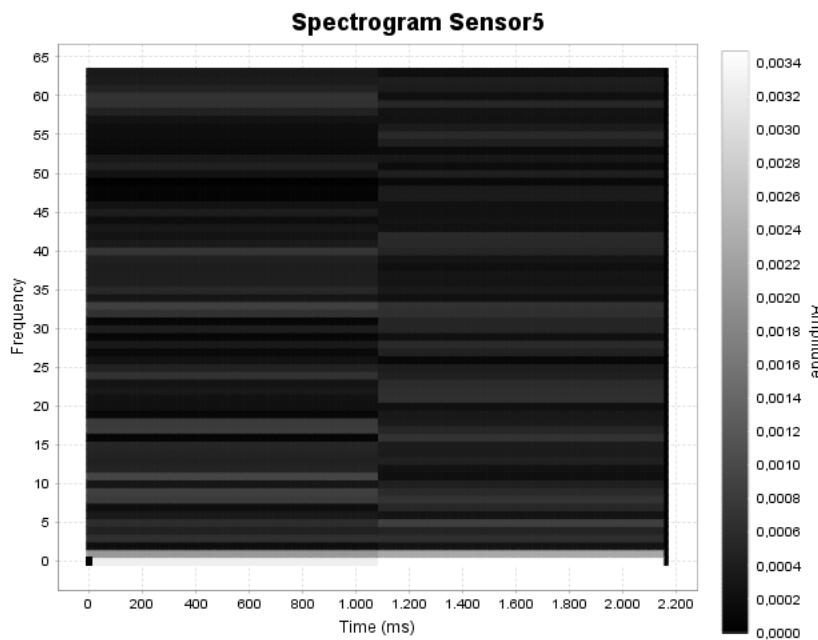


Gambar C.8: Hasil Spectrogram Sensor4 dengan topologi star dan window *Rectangle* di Jalan

- Sensor5



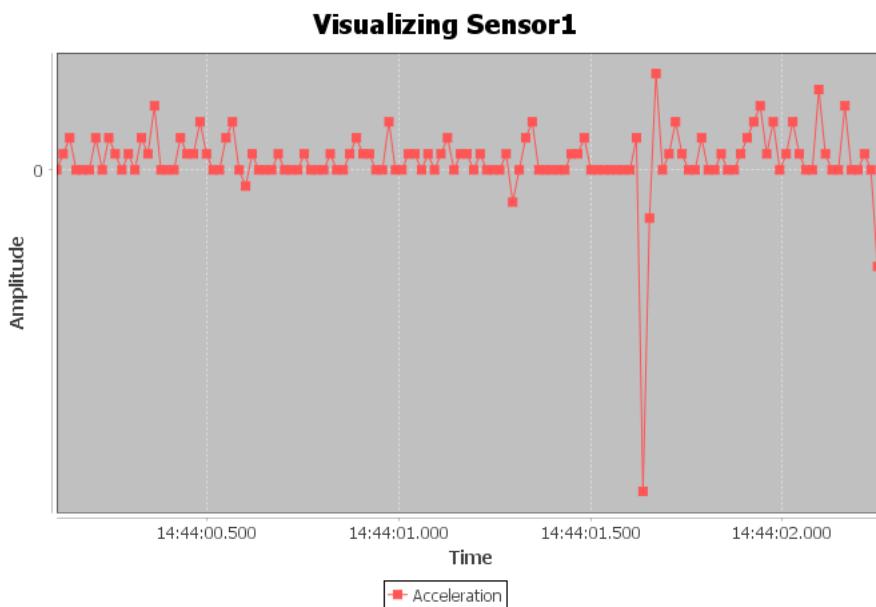
Gambar C.9: Hasil Sensing Sensor5 dengan topologi star dan window *Rectangle* di Jalan



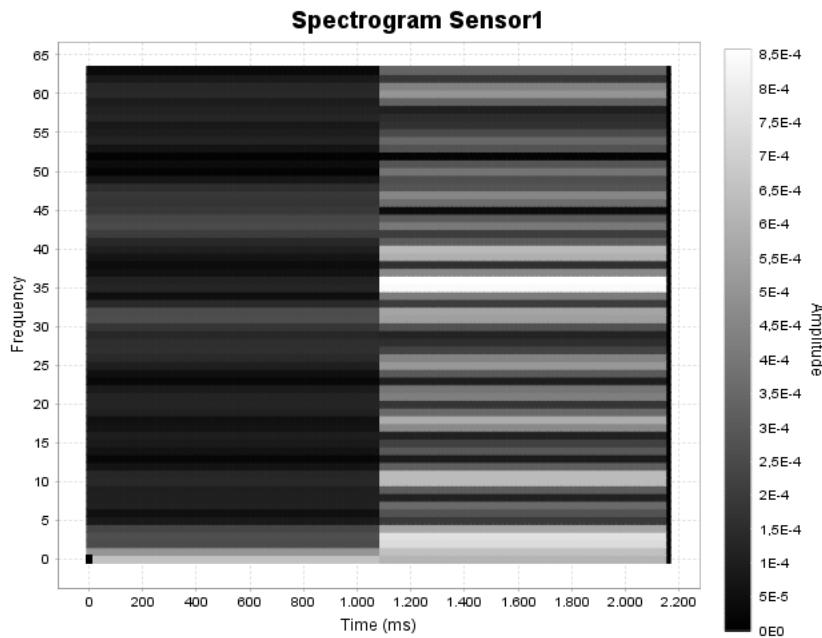
Gambar C.10: Hasil Spectrogram Sensor5 dengan topologi star dan window *Rectangle* di Jalan

C.1.2 Hanning Window

- Sensor1

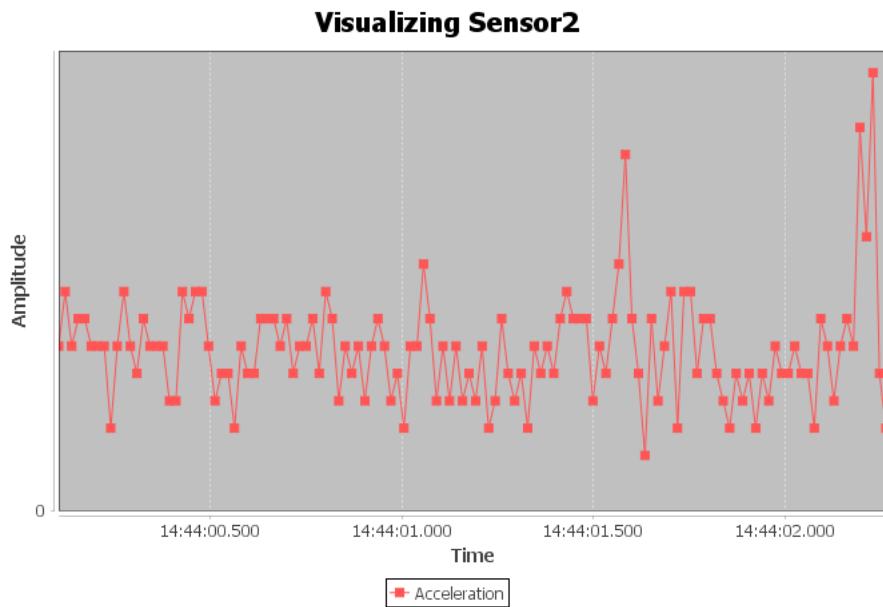


Gambar C.11: Hasil Sensing Sensor1 dengan topologi star dan window *Hanning* di Jalan

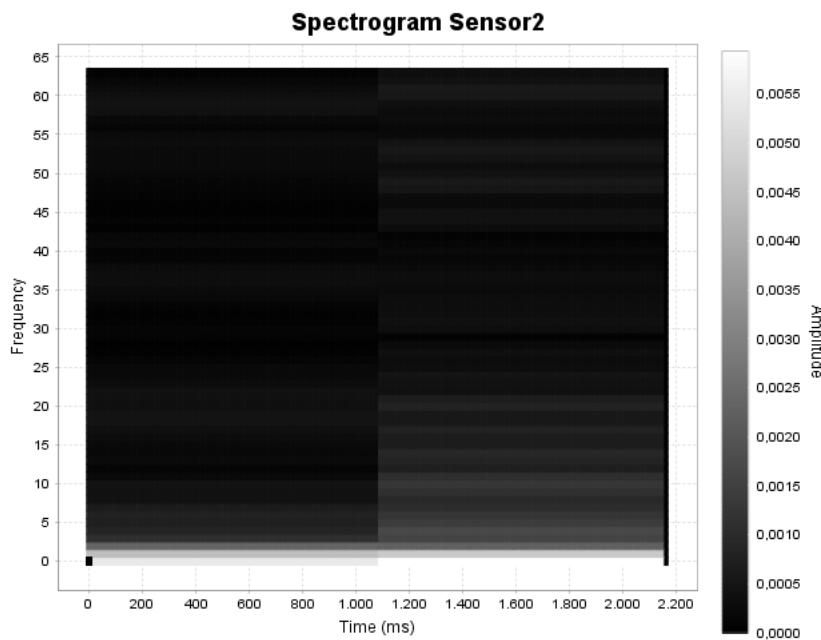


Gambar C.12: Hasil Spectrogram Sensor1 dengan topologi star dan window *Hanning* di Jalan

- Sensor2

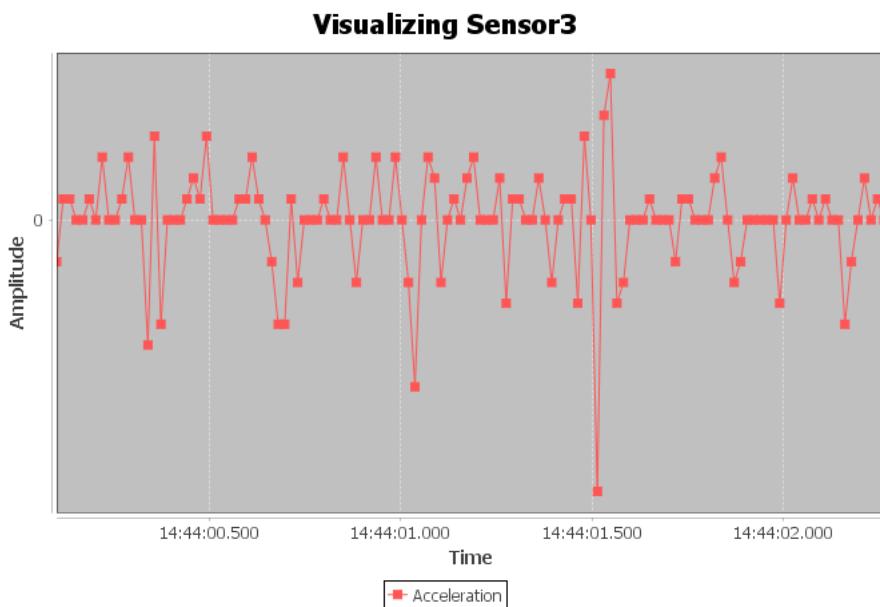


Gambar C.13: Hasil Sensing Sensor2 dengan topologi star dan window *Hanning* di Jalan

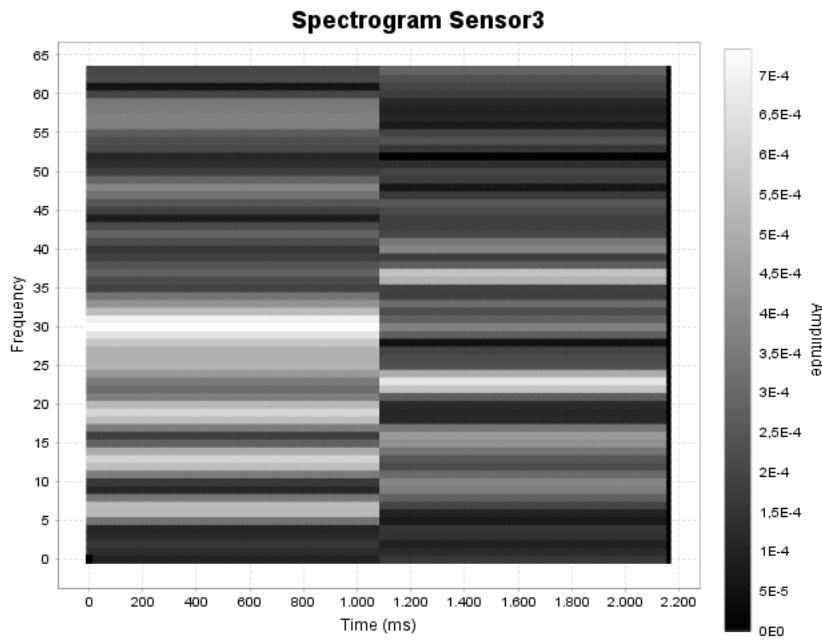


Gambar C.14: Hasil Spectrogram Sensor2 dengan topologi star dan window *Hanning* di Jalan

- Sensor3

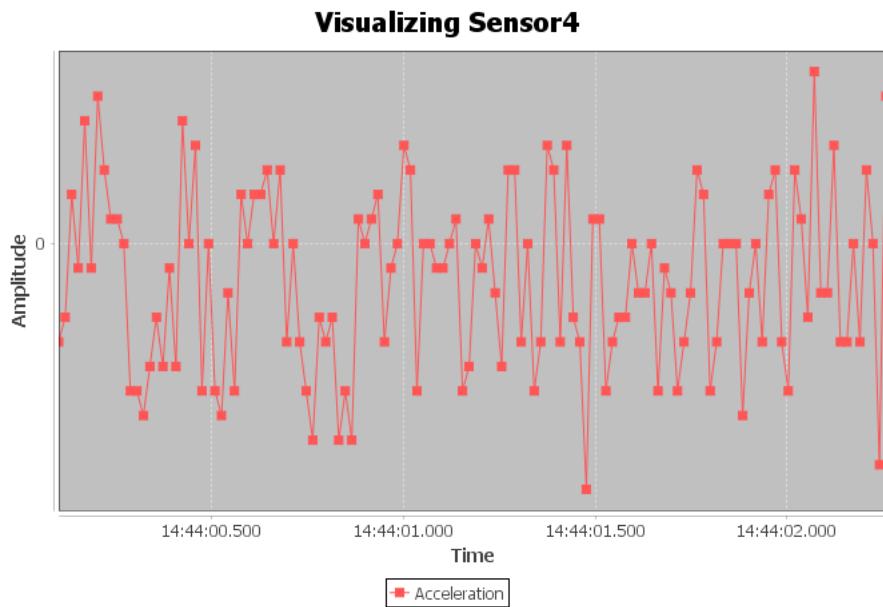


Gambar C.15: Hasil Sensing Sensor3 dengan topologi star dan window *Hanning* di Jalan

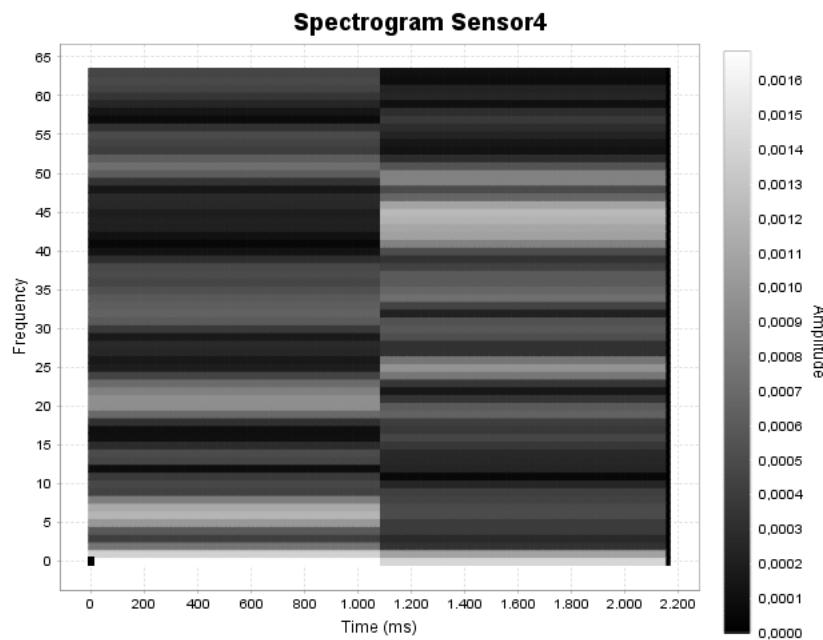


Gambar C.16: Hasil Spectrogram Sensor3 dengan topologi star dan window *Hanning* di Jalan

- Sensor4

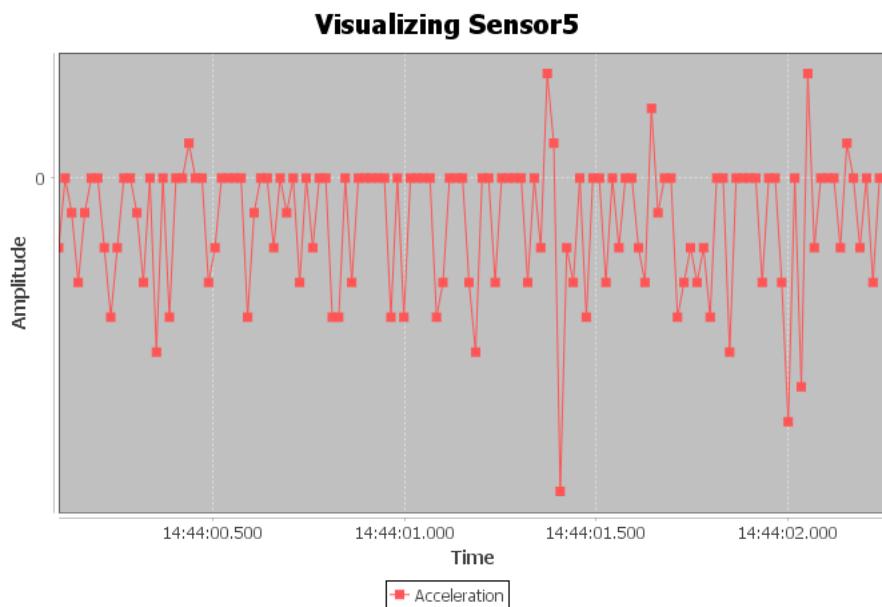


Gambar C.17: Hasil Sensing Sensor4 dengan topologi star dan window *Hanning* di Jalan

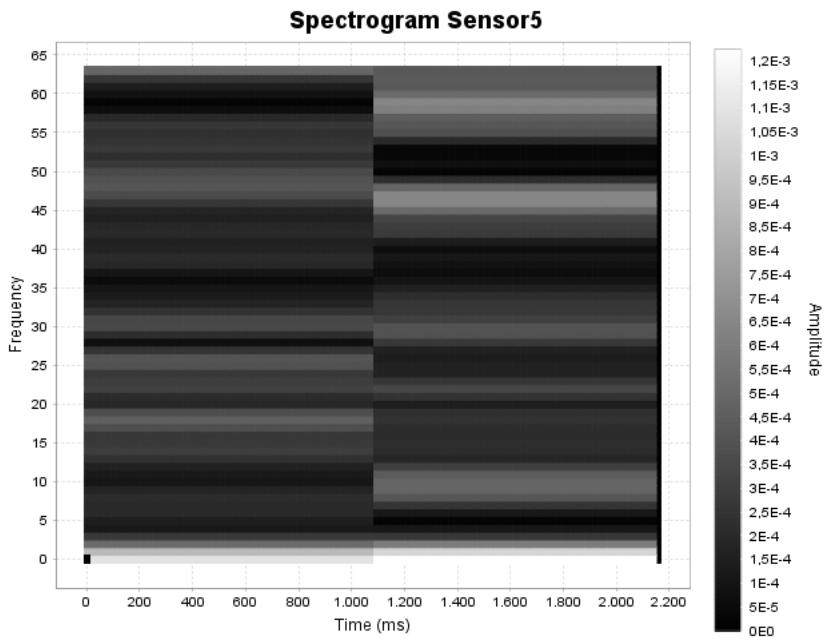


Gambar C.18: Hasil Spectrogram Sensor4 dengan topologi star dan window *Hanning* di Jalan

- Sensor5



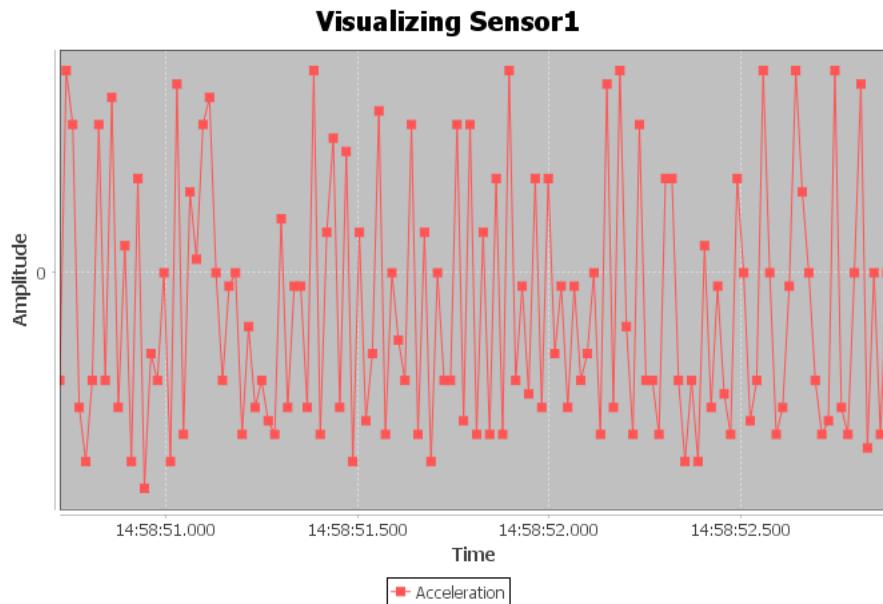
Gambar C.19: Hasil Sensing Sensor5 dengan topologi star dan window *Hanning* di Jalan



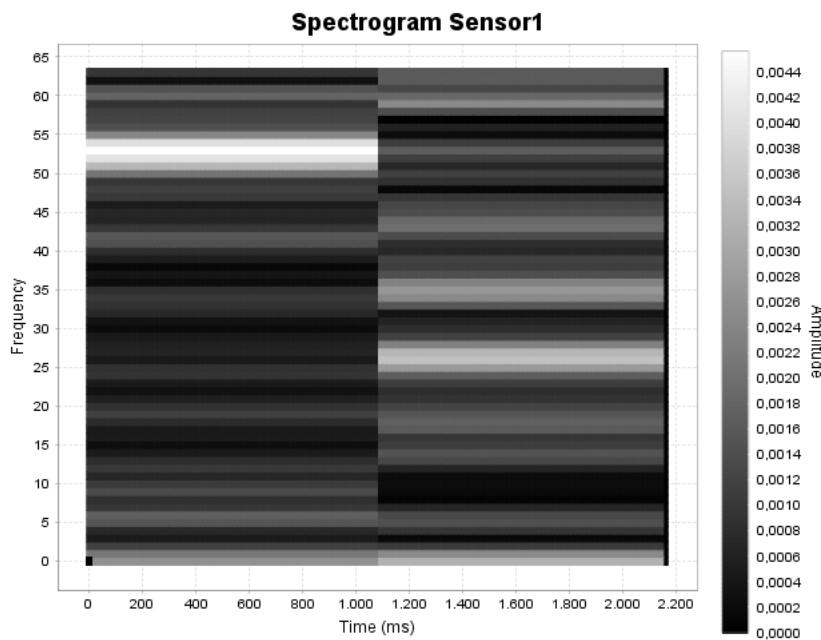
Gambar C.20: Hasil Spectrogram Sensor5 dengan topologi star dan window *Hanning* di Jalan

C.1.3 Hamming Window

- Sensor1

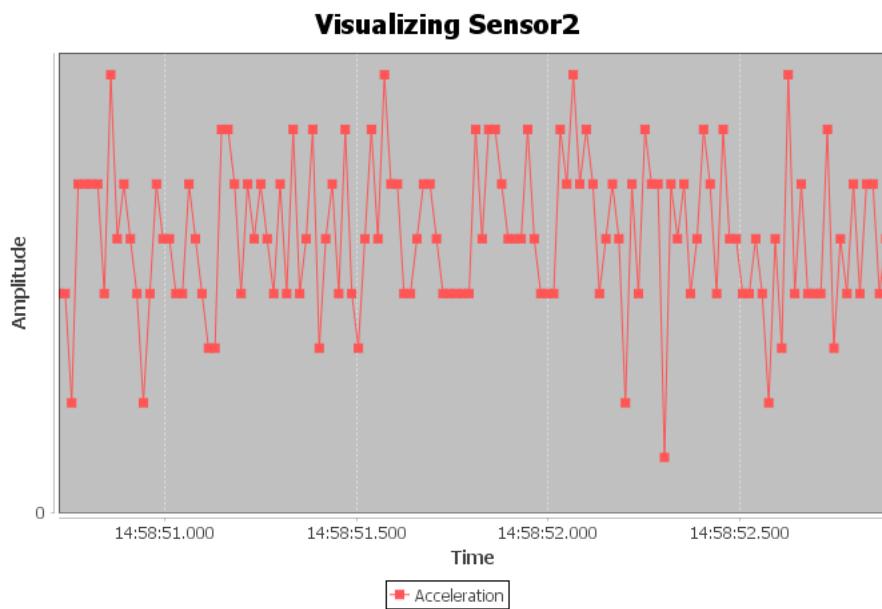


Gambar C.21: Hasil Sensing Sensor1 dengan topologi star dan window *Hamming* di Jalan

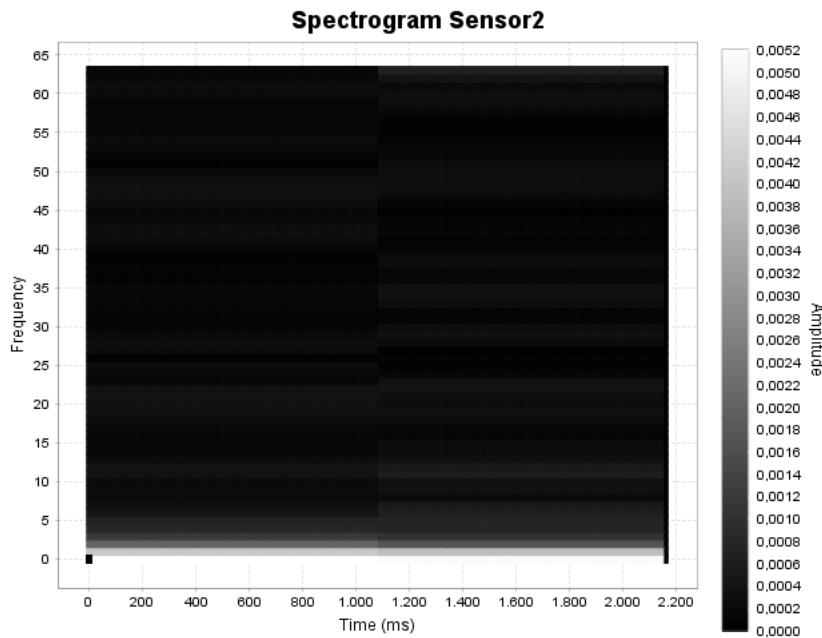


Gambar C.22: Hasil Spectrogram Sensor1 dengan topologi star dan window *Hamming* di Jalan

- Sensor2

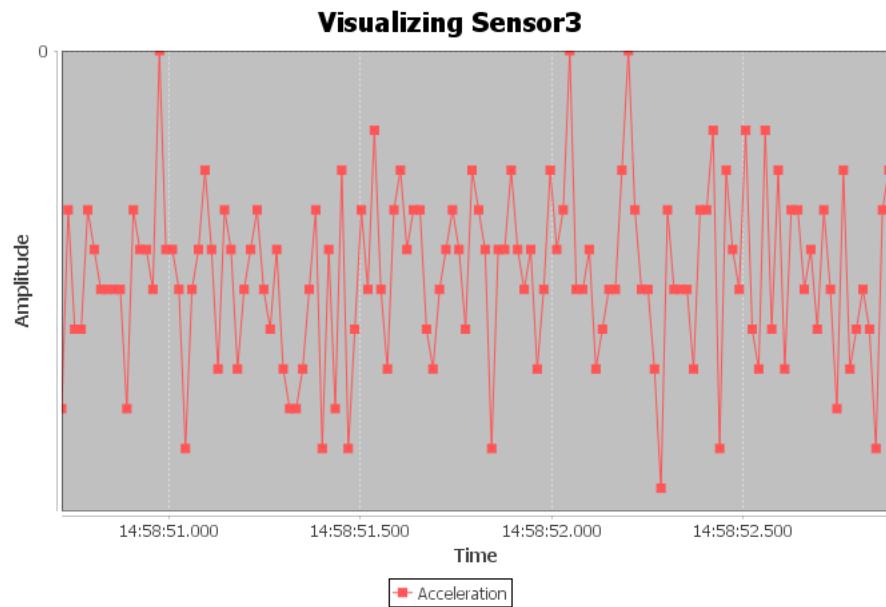


Gambar C.23: Hasil Sensing Sensor2 dengan topologi star dan window *Hamming* di Jalan

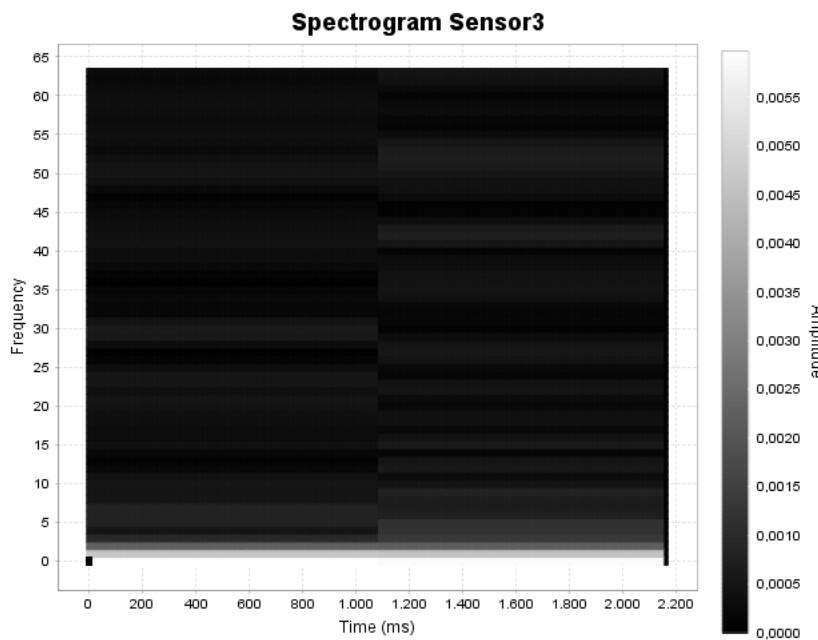


Gambar C.24: Hasil Spectrogram Sensor2 dengan topologi star dan window *Hamming* di Jalan

- Sensor3

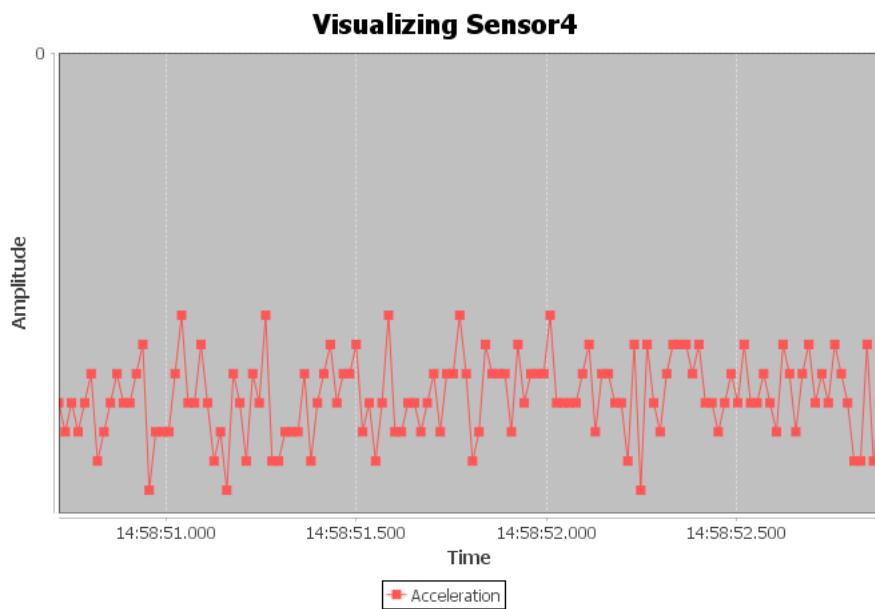


Gambar C.25: Hasil Sensing Sensor3 dengan topologi star dan window *Hamming* di Jalan

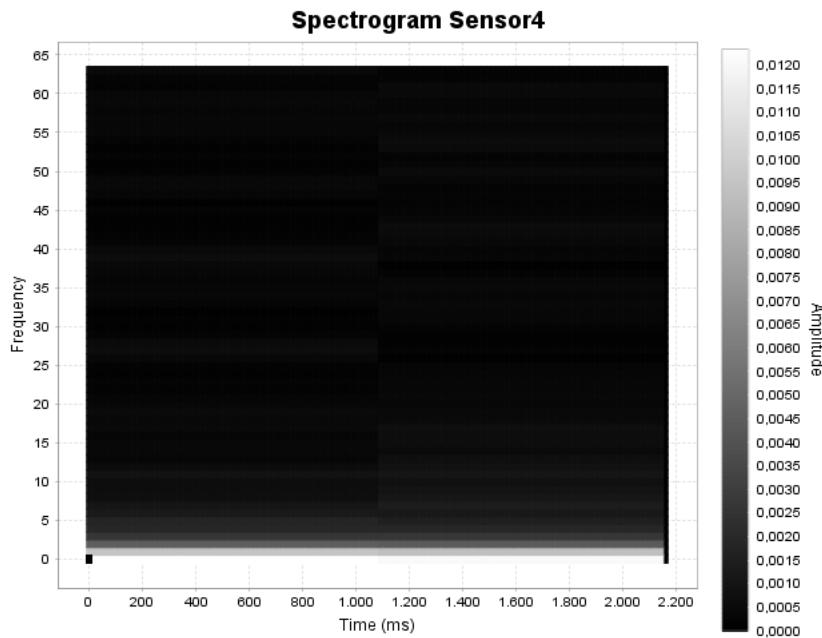


Gambar C.26: Hasil Spectrogram Sensor3 dengan topologi star dan window *Hamming* di Jalan

- Sensor4

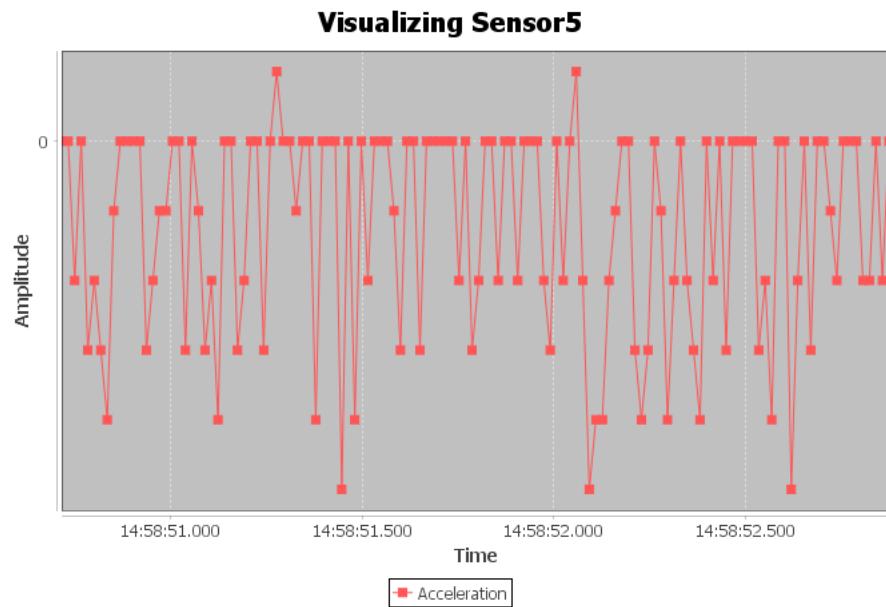


Gambar C.27: Hasil Sensing Sensor4 dengan topologi star dan window *Hamming* di Jalan

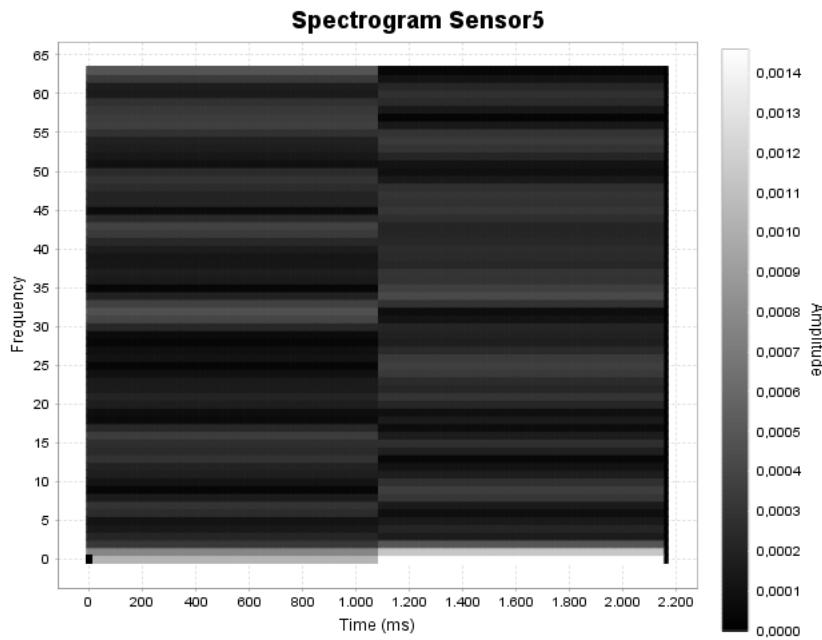


Gambar C.28: Hasil Spectrogram Sensor4 dengan topologi star dan window *Hamming* di Jalan

- Sensor5



Gambar C.29: Hasil Sensing Sensor5 dengan topologi star dan window *Hamming* di Jalan

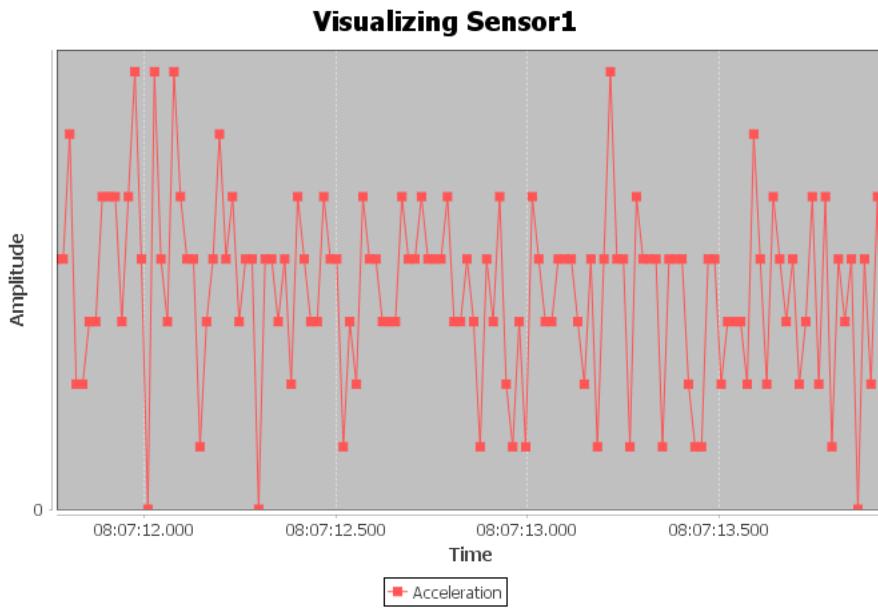


Gambar C.30: Hasil Spectrogram Sensor5 dengan topologi star dan window *Hamming* di Jalan

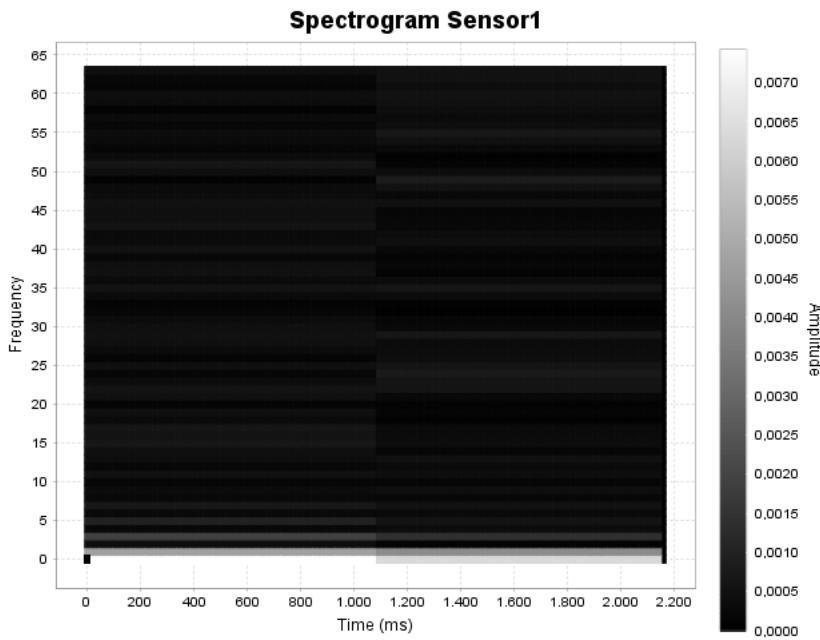
C.2 Topologi Tree

C.2.1 Rectangle Window

- Sensor1

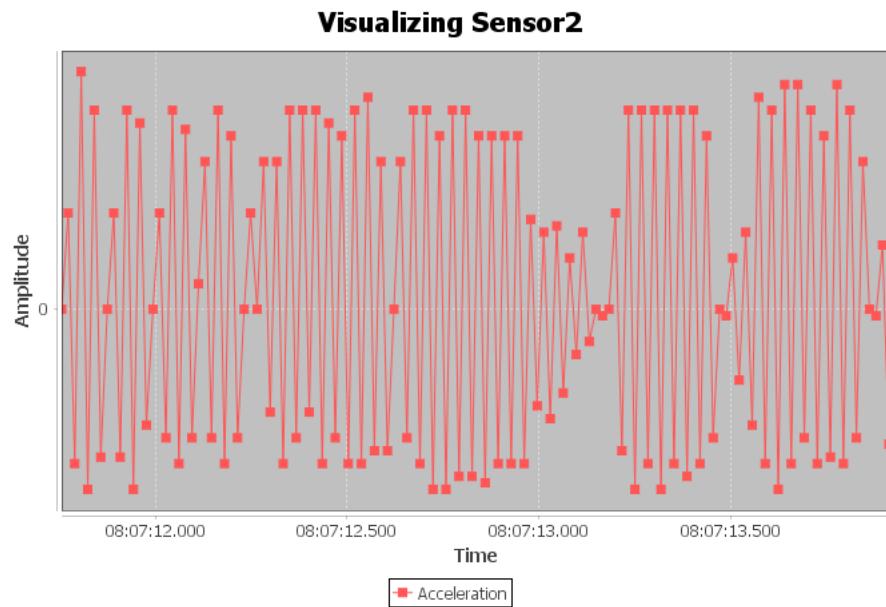


Gambar C.31: Hasil Sensing Sensor1 dengan topologi tree dan window *Rectangle* di Jalan

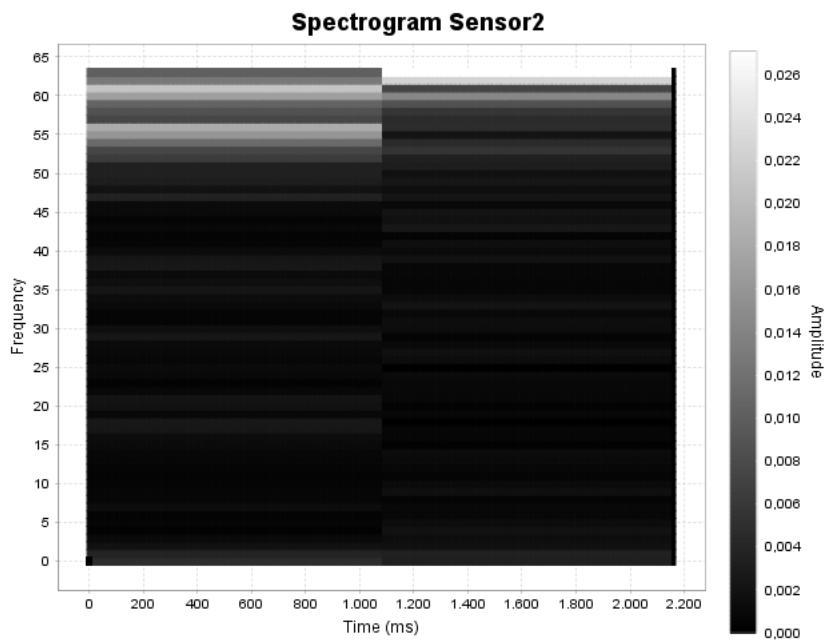


Gambar C.32: Hasil Spectrogram Sensor1 dengan topologi tree dan window *Rectangle* di Jalan

- Sensor2

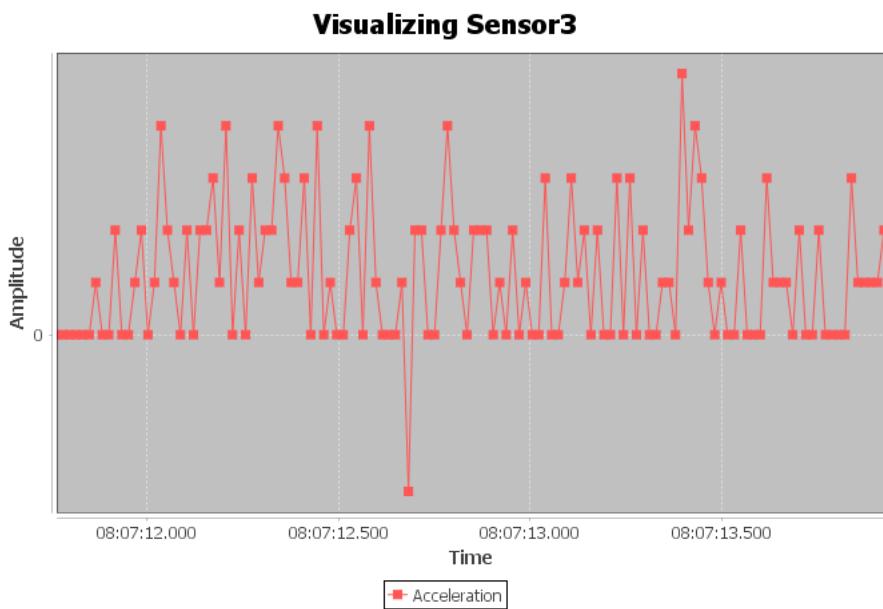


Gambar C.33: Hasil Sensing Sensor2 dengan topologi tree dan window *Rectangle* di Jalan

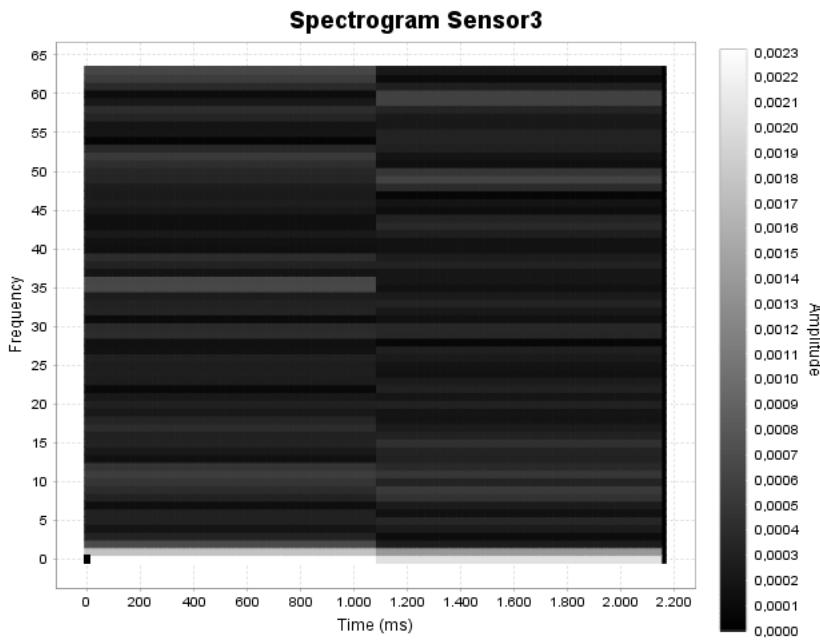


Gambar C.34: Hasil Spectrogram Sensor2 dengan topologi tree dan window *Rectangle* di Jalan

- Sensor3

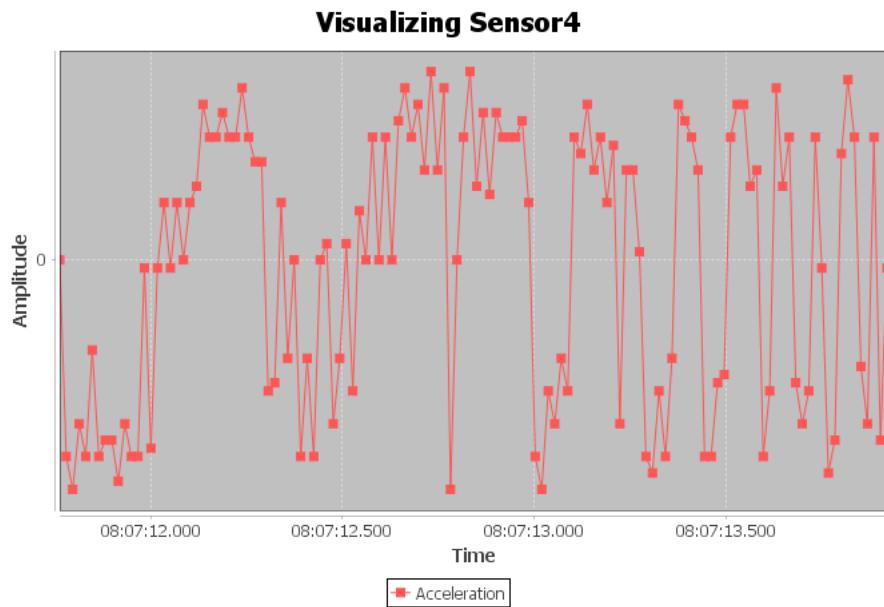


Gambar C.35: Hasil Sensing Sensor3 dengan topologi tree dan window *Rectangle* di Jalan

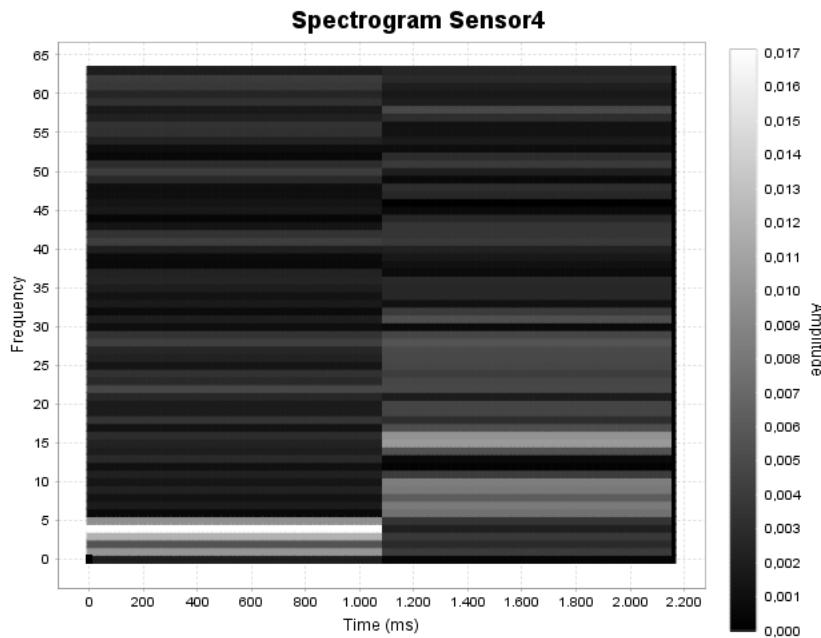


Gambar C.36: Hasil Spectrogram Sensor3 dengan topologi tree dan window *Rectangle* di Jalan

- Sensor4

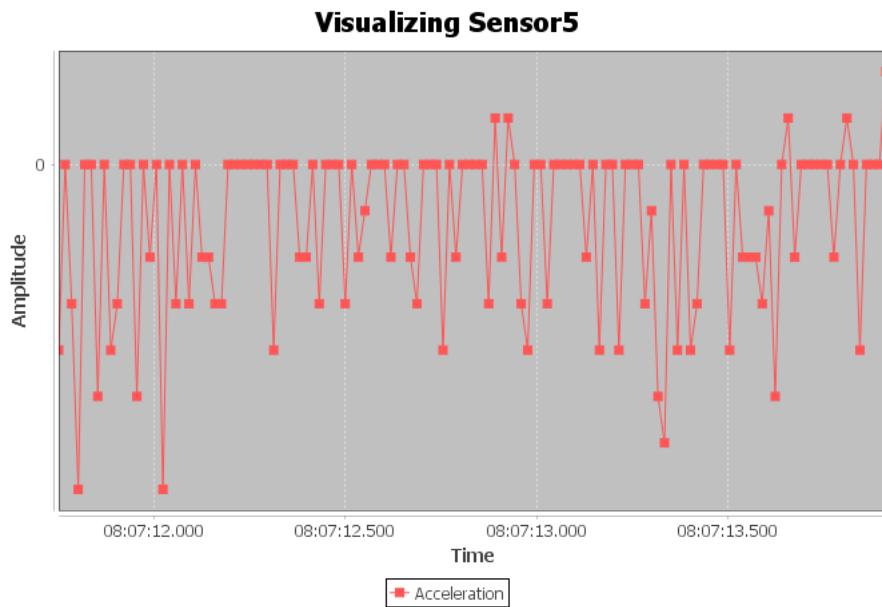


Gambar C.37: Hasil Sensing Sensor4 dengan topologi tree dan window *Rectangle* di Jalan

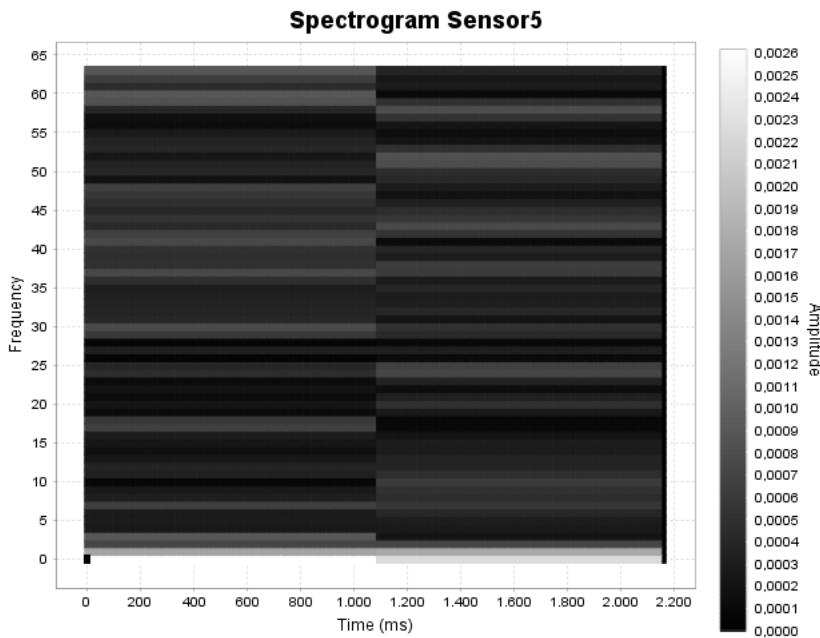


Gambar C.38: Hasil Spectrogram Sensor4 dengan topologi tree dan window *Rectangle* di Jalan

- Sensor5



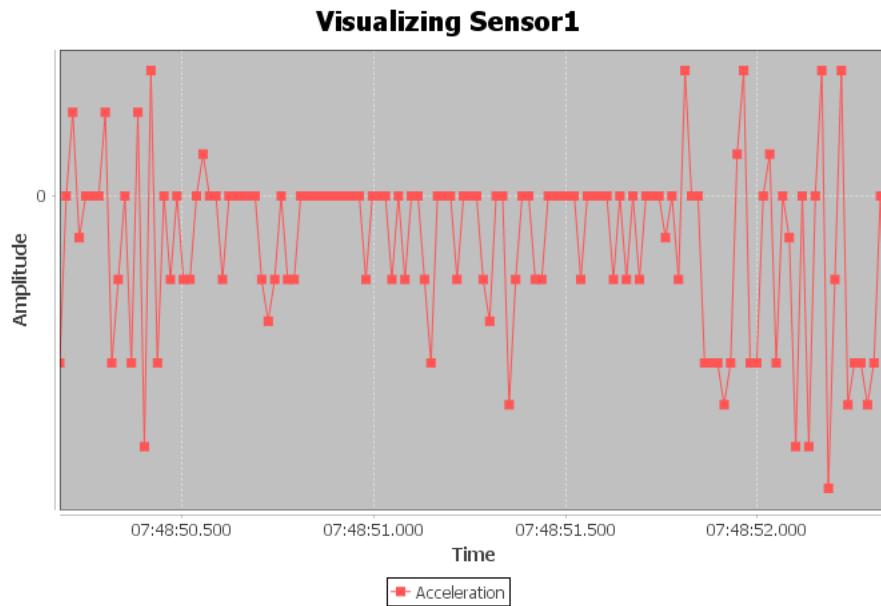
Gambar C.39: Hasil Sensing Sensor5 dengan topologi tree dan window *Rectangle* di Jalan



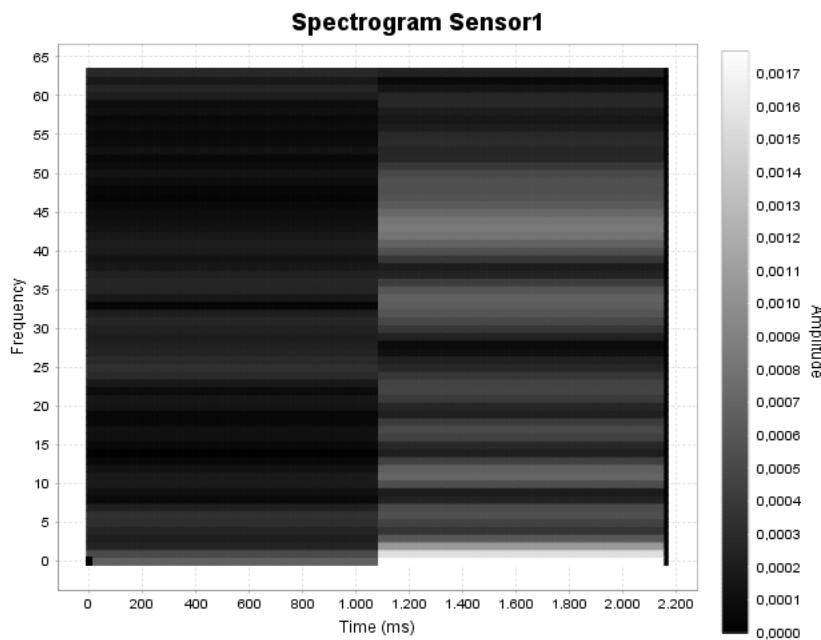
Gambar C.40: Hasil Spectrogram Sensor5 dengan topologi tree dan window *Rectangle* di Jalan

C.2.2 Hanning Window

- Sensor1

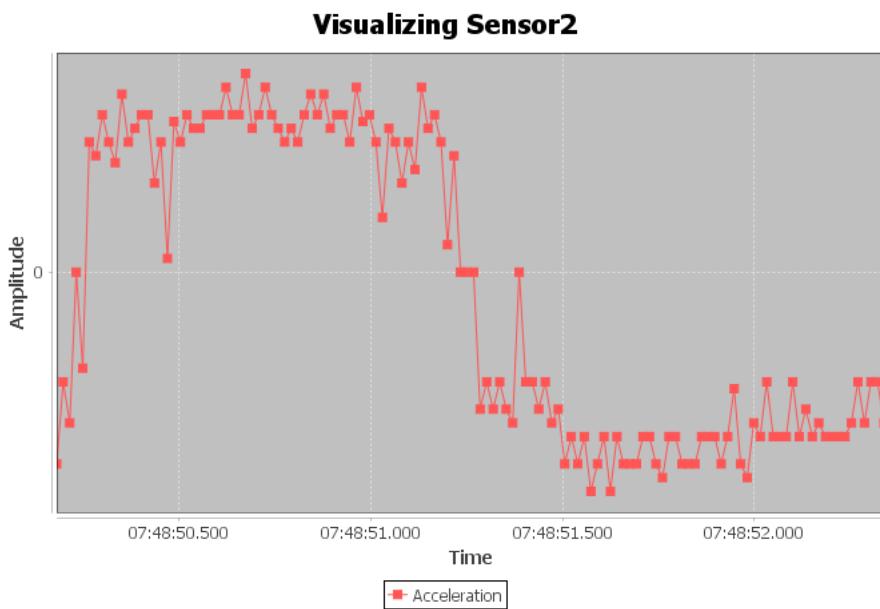


Gambar C.41: Hasil Sensing Sensor1 dengan topologi tree dan window *Hanning* di Jalan

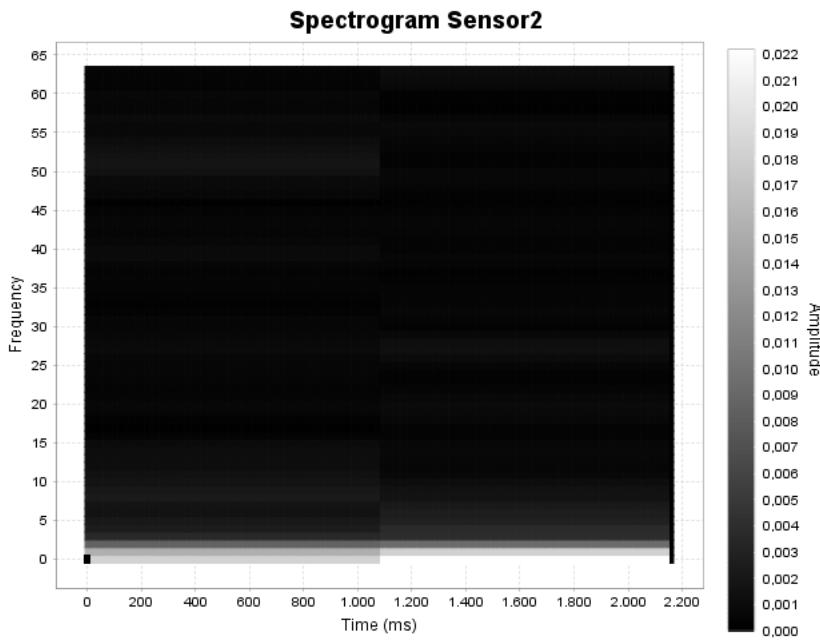


Gambar C.42: Hasil Spectrogram Sensor1 dengan topologi tree dan window *Hanning* di Jalan

- Sensor2

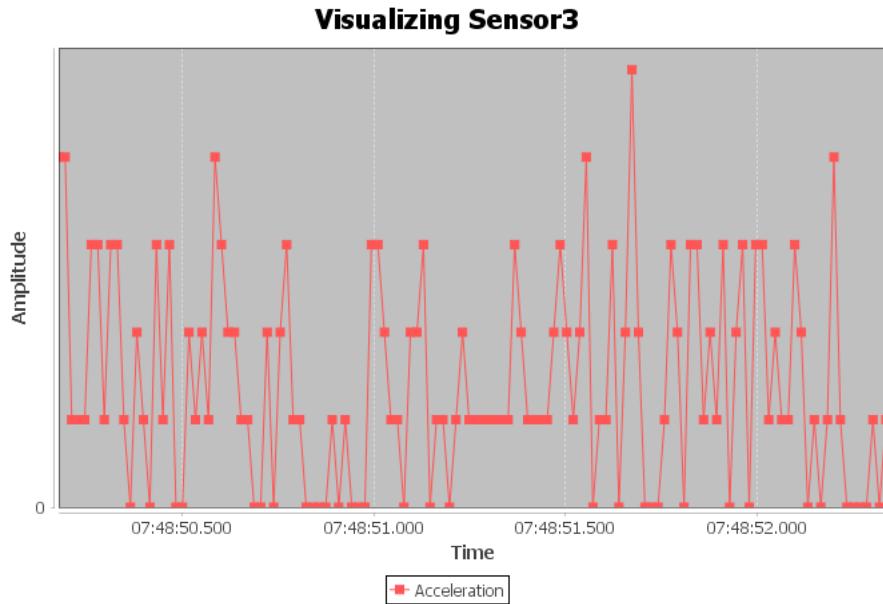


Gambar C.43: Hasil Sensing Sensor2 dengan topologi tree dan window *Hanning* di Jalan

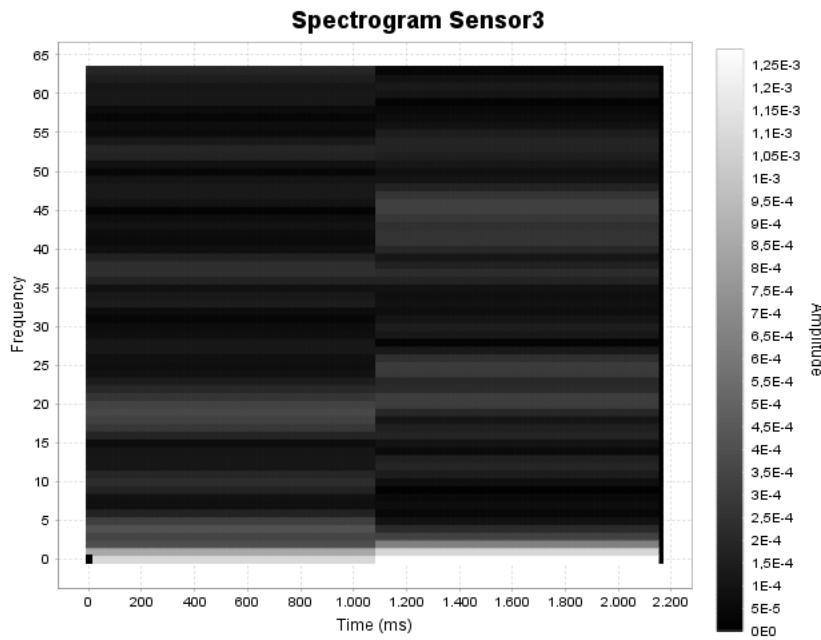


Gambar C.44: Hasil Spectrogram Sensor2 dengan topologi tree dan window *Hanning* di Jalan

- Sensor3

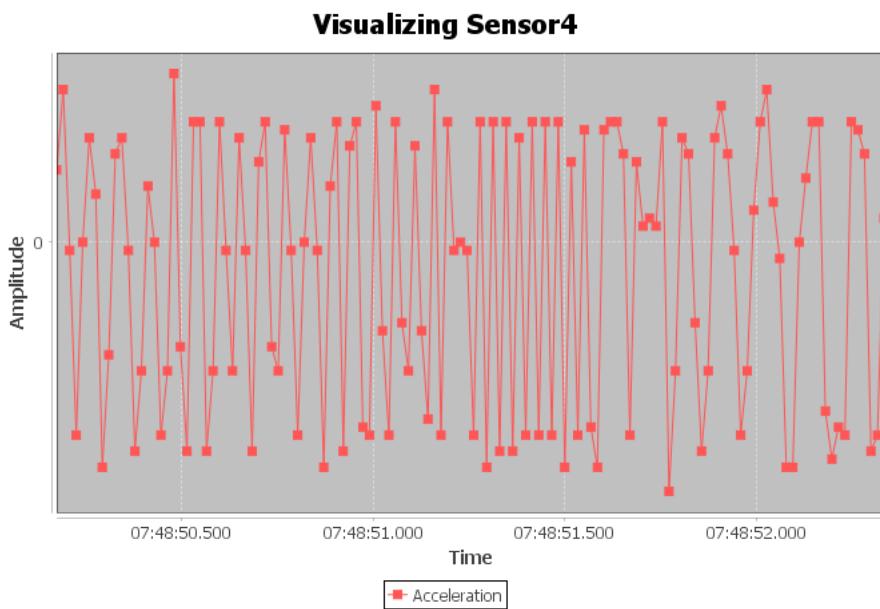


Gambar C.45: Hasil Sensing Sensor3 dengan topologi tree dan window *Hanning* di Jalan

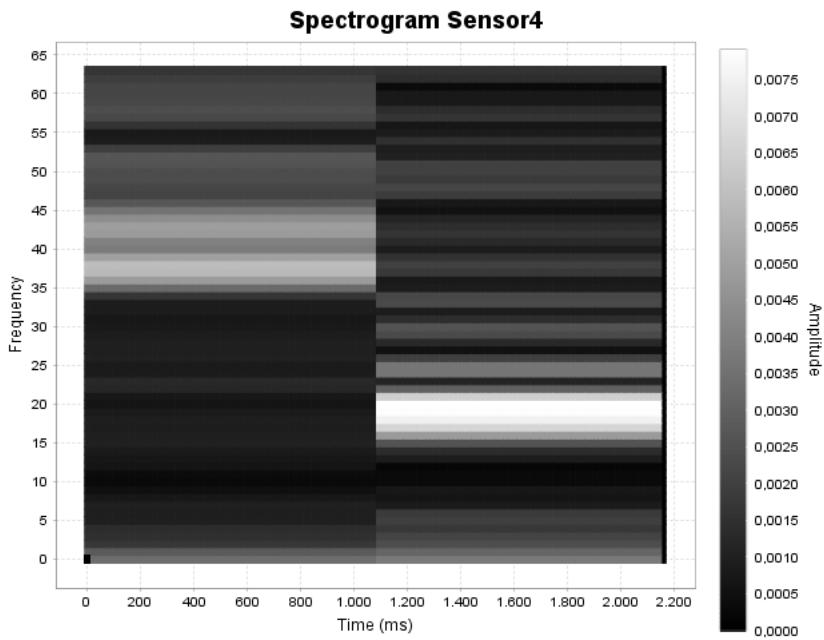


Gambar C.46: Hasil Spectrogram Sensor3 dengan topologi tree dan window *Hanning* di Jalan

- Sensor4

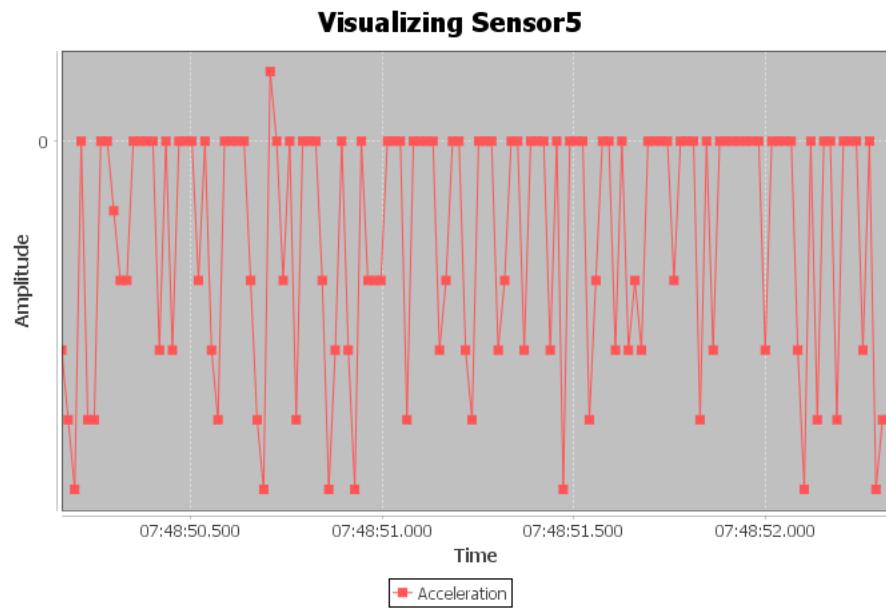


Gambar C.47: Hasil Sensing Sensor4 dengan topologi tree dan window *Hanning* di Jalan

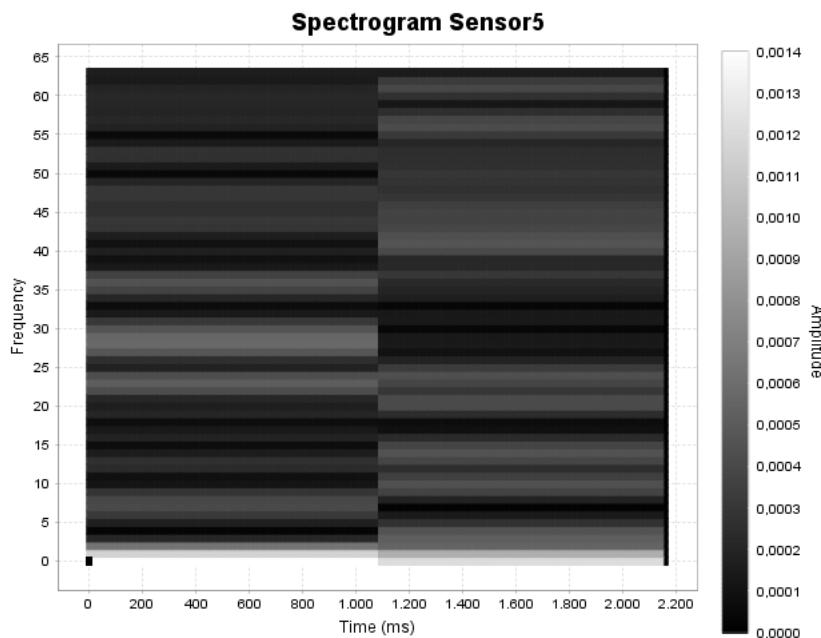


Gambar C.48: Hasil Spectrogram Sensor4 dengan topologi tree dan window *Hanning* di Jalan

- Sensor5



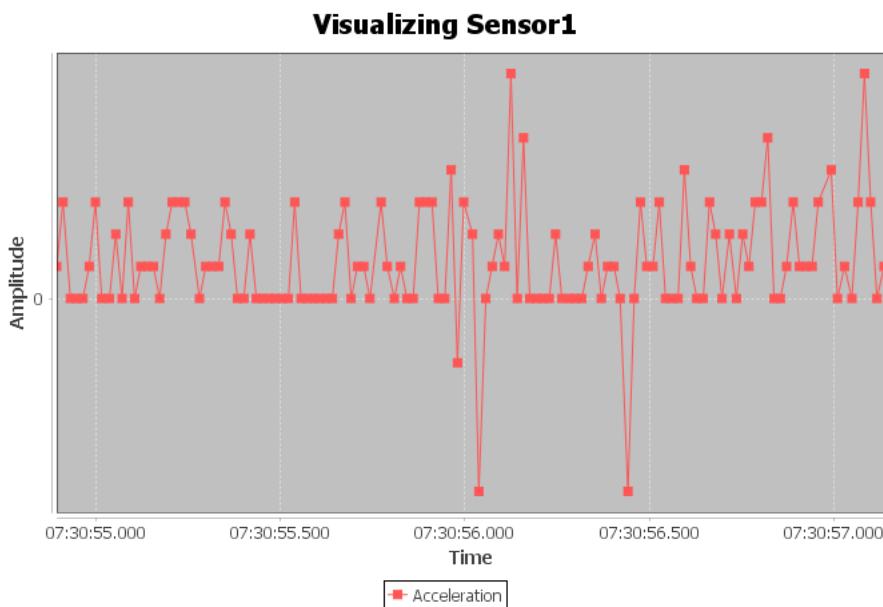
Gambar C.49: Hasil Sensing Sensor5 dengan topologi tree dan window *Hanning* di Jalan



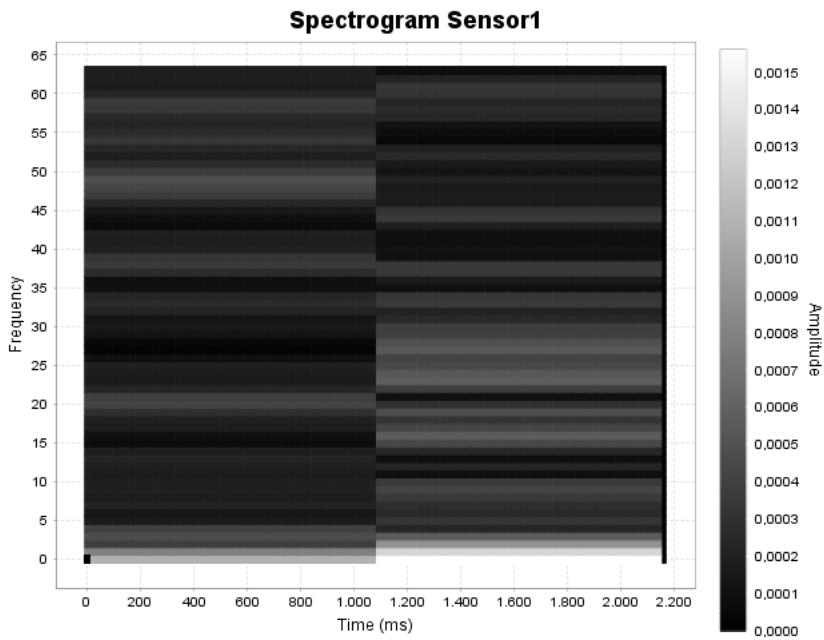
Gambar C.50: Hasil Spectrogram Sensor5 dengan topologi tree dan window *Hanning* di Jalan

C.2.3 Hamming Window

- Sensor1

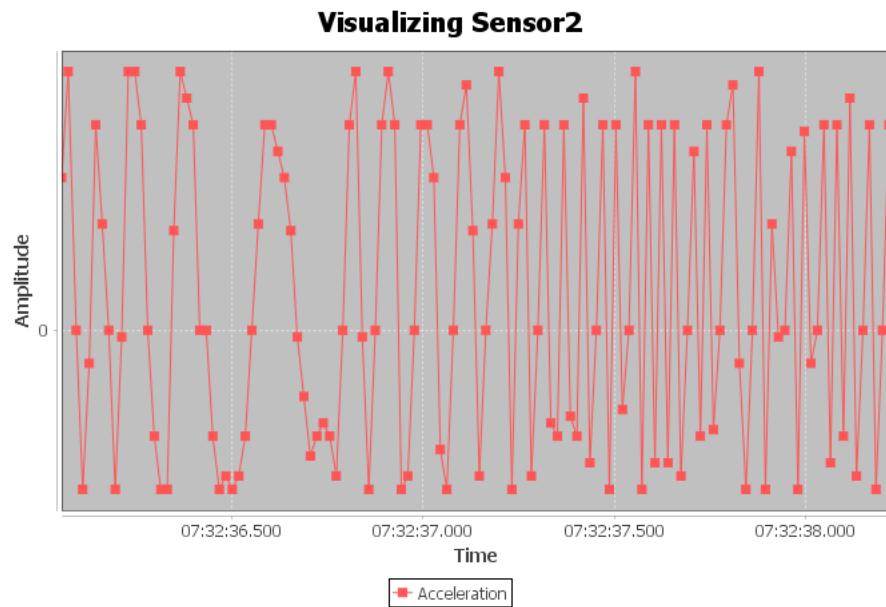


Gambar C.51: Hasil Sensing Sensor1 dengan topologi tree dan window *Hamming* di Jalan

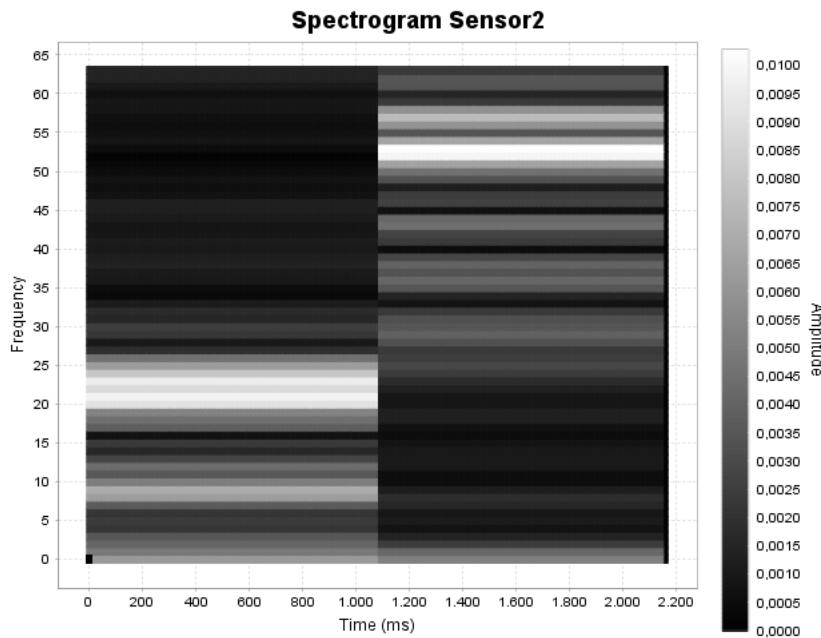


Gambar C.52: Hasil Spectrogram Sensor1 dengan topologi tree dan window *Hamming* di Jalan

- Sensor2

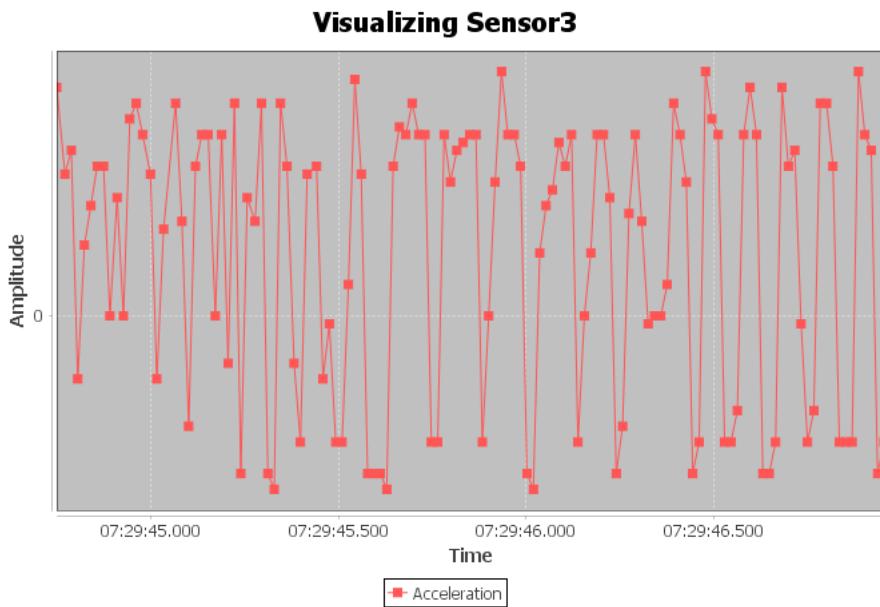


Gambar C.53: Hasil Sensing Sensor2 dengan topologi tree dan window *Hamming* di Jalan

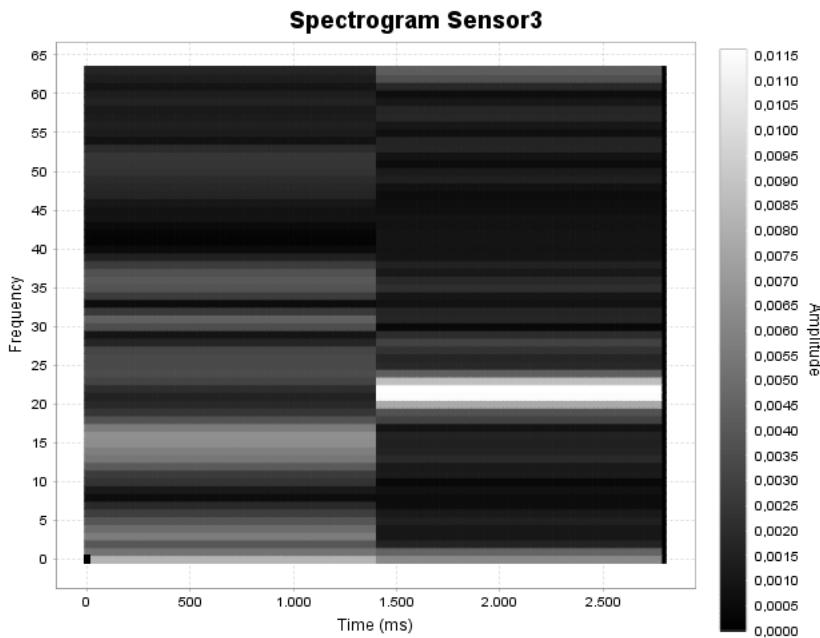


Gambar C.54: Hasil Spectrogram Sensor2 dengan topologi tree dan window *Hamming* di Jalan

- Sensor3

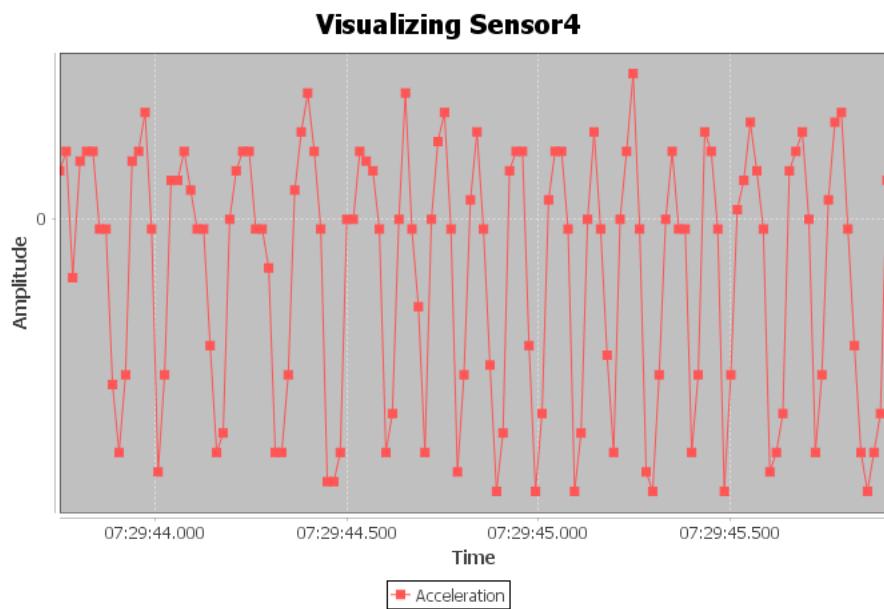


Gambar C.55: Hasil Sensing Sensor3 dengan topologi tree dan window *Hamming* di Jalan

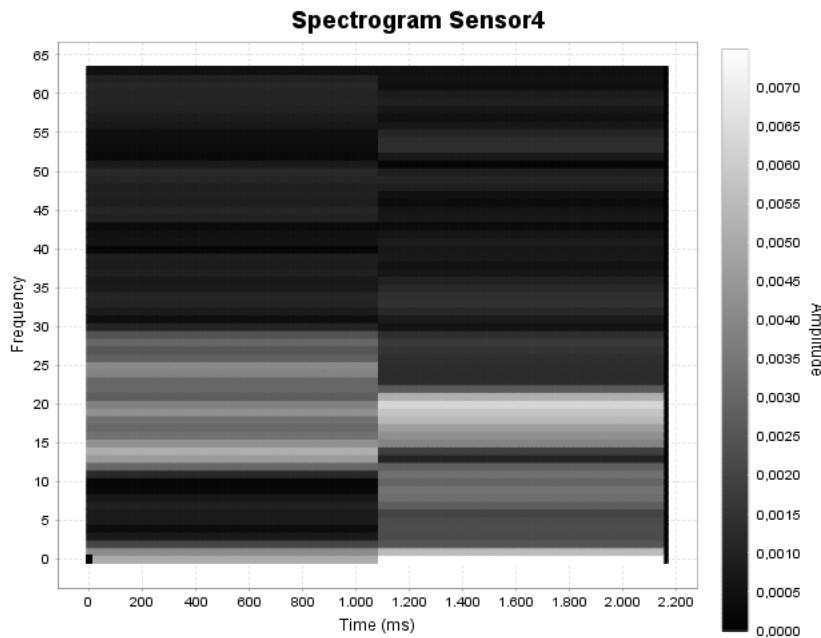


Gambar C.56: Hasil Spectrogram Sensor3 dengan topologi tree dan window *Hamming* di Jalan

- Sensor4

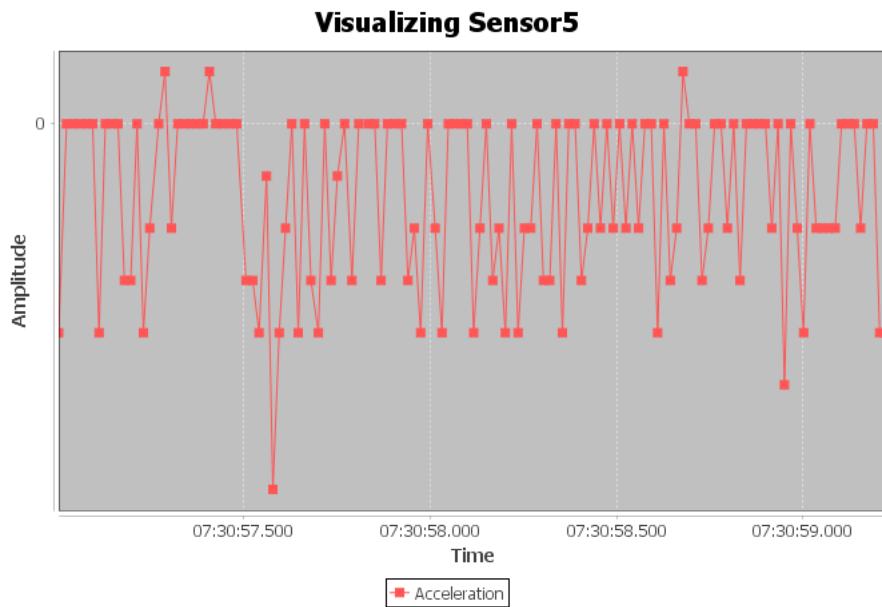


Gambar C.57: Hasil Sensing Sensor4 dengan topologi tree dan window *Hamming* di Jalan

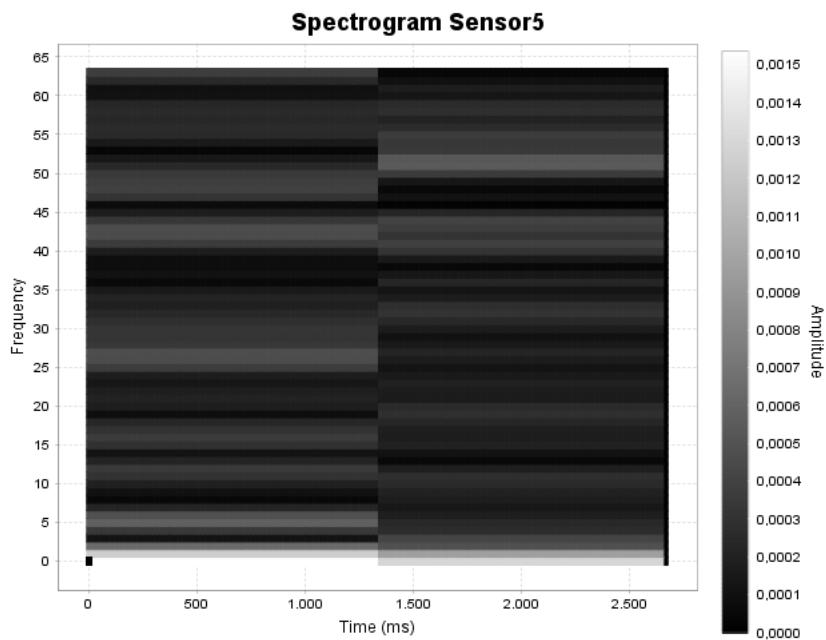


Gambar C.58: Hasil Spectrogram Sensor4 dengan topologi tree dan window *Hamming* di Jalan

- Sensor5



Gambar C.59: Hasil Sensing Sensor5 dengan topologi tree dan window *Hamming* di Jalan



Gambar C.60: Hasil Spectrogram Sensor5 dengan topologi tree dan window *Hamming* di Jalan