TUGAS AKHIR – EE 184801

Implementasi dan Analisis Kinerja Algoritma Kompresi Data pada Jaringan Sensor Nirkabel berbasis Arduino

Fachri Akbar Rafsanzani
NRP 07111540000123

Dosen Pembimbing
Dr. Ir. Wirawan, DEA

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

FINAL PROJECT – EE 184801

IMPLEMENTATION AND PERFORMANCE ANALYSIS OF DATA COMPRESSION ALGORITHM ON ARDUINO-BASED WIRELESS SENSOR NETWORKS

Fachri Akbar Rafsanzani
NRP 07111540000123

Supervisors
Dr. Ir. Wirawan, DEA

DEPARTEMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Engineering
Institut Teknologi Sepuluh Nopember
Surabaya 2019

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan tugas akhir saya dengan judul "Implementasi Dan Analisis Kinerja Algoritma Kompresi Data Pada Jaringan Sensor Nirkabel Berbasis Arduino" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri. Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2019

Fachri Akbar Rafsanzani
NRP 07 11 15 4000 0123

IMPLEMENTASI DAN ANALISIS KINERJA ALGORITMA KOMPRESI DATA PADA JARINGAN SENSOR NIRKABEL BERBASIS ARDUINO

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyartan
Untuk Memperoleh Gelar Serjana Teknik Elektro
Pada
Bidang Studi Telekomunikasi Multimedia
Departemen Teknik Elektro
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui,
Dosen Pembimbing

Dr. Ir. Wirawan, DEA.
NIP. 196311091989031011

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
JULI, 2019

IMPLEMENTASI DAN ANALISIS KINERJA ALGORITMA KOMPRESI DATA PADA JARINGAN SENSOR NIRKABEL BERBASIS ARDUINO

Nama Mahasiswa : Fachri Akbar Rafsanzani
Dosen Pembimbing : Dr. Ir. Wirawan, DEA

ABSTRAK

Jaringan sensor nirkabel merupakan sebuah jaringan yang terdiri dari beberapa sensor yang saling terhubung secara nirkabel. Jaringan sensor nirkabel umumnya digunakan untuk beberapa hal seperti sistem *monitoring* atau untuk mendeteksi suatu kejadian. Permasalahan yang paling sering ditemui ialah mengenai konsumsi energi dari satu *node*. Untuk mengirim satu bit dari satu *node* ke *node* yang lain atau ke *sink*, diperlukan energi yang cukup besar bila dibandingkan untuk komputasi atau akuisisi data dari sensor. Beberapa pendekatan dilakukan untuk mengurangi masalah yang dialami oleh *node*. Pendekatan yang akan dilakukan di tugas akhir ini yaitu menggunakan teknik kompresi data untuk mengurangi konsumsi energi saat dilakukan transmisi dan diimplementasikan ke dalam mikrokontroller yaitu Arduino. Sebelum dilakukan implementasi ke Arduino, perlu dilakukan simulasi menggunakan Matlab. Arduino sendiri akan digunakan untuk mendapatkan data pengukuran. Melalui simulasi, didapatkan bahwa metode *sequential lossless entropy coding (S-LEC)* merupakan metode kompresi data yang paling efisien. Hasil pengukuran juga menunjukkan bahwa metode S-LEC merupakan metode yang sangat efisien dalam mengurangi energi yang dikonsumsi dengan efisiensi sebesar 18,63%.

Kata Kunci : *Jaringan sensor nirkabel, Kompresi data, Arduino, Matlab*

Halaman ini sengaja dikosongkan

Implementation and Perfomance Analysis of Data Compression Algorithm on Arduino-based Wireless Sensor Network

Name : Fachri Akbar Rafsanzani
Advisor : Dr. Ir. Wirawan, DEA

ABSTRACT

Wireless sensor network is a network consisting of several sensors that are connected wirelessly to each other. Wireless sensor networks are generally used for several things such as monitoring systems or to detect an event. The most common problem is about energy consumption from one node. To send one bit from one node to another node or to the sink, it requires considerable energy compared to computing or data acquisition from the sensor. Some approaches are carried out to reduce the problems experienced by nodes. The approach to be carried out in this final project is to use data compression techniques to reduce energy consumption when carried out transmission and implemented into a microcontroller, namely Arduino. Before implementing the Arduino, a simulation using Matlab is required. Arduino itself will be used to obtain measurement data. Through simulation, it was found that the sequential lossless entropy coding (S-LEC) method was the most efficient data compression method. The measurement results also show that the S-LEC method is a very efficient method in saving energy consumed with.an efficiency of 18,63%.

Keywords : *Wireless Sensor Networks, Data compression, Arduino, Matlab*

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Segala puji bagi Allah SWT yang telah memberi kekuatan dan petunjuk untuk menyelesaikan tugas akhir dengan judul **"Implementasi dan Analisis Kinerja Algoritma Kompresi Data pada Jaringan Sensor Nirkabel berbasis Arduino"**. Tugas Akhir ini disusun sebagai salah satu persyaratan untuk menyelesaikan jenjang pendidikan S1 Teknik Elektro pada Bidang Studi Telekomunikasi Multimedia, Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember. Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan bimbingan:

1. Kedua orang tua yang selalu memberikan semangat dan motivasi
2. Bapak Dr. Ir. Wirawan, DEA yang selalu memberikan bimbingan dan arahan
3. Bapak/Ibu dosen bidang studi Telekomunikasi Multimedia yang memberikan nasihat dalam pengerjaan tugas akhir ini
4. Teman-teman yang membantu dan menjadi teman diskusi dalam pengerjaan tugas akhir ini.

Demikian semoga buku tugas akhir ini dapat memberikan manfaat.

Surabaya, Mei 2019

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

1.1 Latar Belakang

Sensor adalah sebuah komponen yang berfungsi untuk mengambil sebuah data dari parameter yang diukur. Contoh beberapa parameter yang diukur adalah suhu, kelembaban, atau medan magnet. Jika sensor tidak hanya satu tetapi ada lebih dari satu dan saling berkomunikasi disebut jaringan sensor nirkabel. Jaringan sensor nirkabel merupakan sebuah jaringan yang terdiri dari beberapa sensor yang terhubung secara nirkabel. Jaringan sensor nirkabel umumnya digunakan untuk mendeteksi suatu kejadian atau untuk sistem *monitoring*. Jaringan sensor nirkabel umumnya menggunakan baterai sebagai sumber daya utama. Permasalahan yang sering dialami oleh jaringan sensor nirkabel adalah efisiensi energi dari *node.* Energi menjadi hal yang penting dari *node.* Energi dari node harus digunakan sehemat mungkin karena pada umumnya tujuan dari *node* sendiri untuk sistem monitoring dalam jangka waku yang lama. Untuk mengirim dari satu bit dari satu *node* ke *node* yang lain atau ke *sink*, diperlukan energi yang cukup besar bila dibandingkan untuk komputasi dari mikrokontroller atau akuisisi data dari sensor. Pada penelitian yang telah ada, kompresi data menarik perhatian sebagai metode untuk meningkatkan efisiensi energi dari node. Kompresi data adalah proses untuk mereduksi data dengan mempertahankan keutuhan informasi. Dengan kata lain, beberapa bit yang dikirim akan direduksi tetapi tidak mengubah informasi yang akan disampaikan. Banyak algoritma kompresi data yang sudah diimplementasikan pada jaringan sensor nirkabel untuk meningkatkan efisiensi energi dari node seperti Huffman, LZW atau *lookup table*.

1.2 Perumusan Masalah

Permasalahan yang dibahas pada penelitian tugas akhir ini adalah sebagai berikut:

1. Bagaimana merancang alat yang berbasis arduino untuk jaringan sensor nirkabel yang menghemat energi dari node?
2. Bagaimana mengimplementasikan algoritma kompresi data pada jaringan sensor nitkabel berbasis arduino?
3. Bagaimana pengaruh kompresi data terhadap data yang ditransmisikan?

4. Bagaimana pengaruh kompresi data terhadap efisiensi energi dari node?

1.3 Batasan Masalah

Batasan permasalahan dari penelitian tugas akhir ini adalah sebagai berikut:

1. Pengukuran dilakukan di sekitar Laboratorium Instrumentasi Pengukuran dan Identifikasi Sistem Tenaga
2. Data yang diukur yaitu Suhu dan Kelembaban
3. Simulasi menggunakan perangkat lunak Matlab
4. Implementasi alat menggunakan mikrokontroler Arduino UNO dan protokol Zigbee
5. Algoritma kompresi yang diimplementasikan yaitu Lossless Entropy Coding (LEC) dan Sequential Lossless Entropy Coding

1.4 Tujuan

Tujuan yang ingin dicapai dari penelitian tugas akhir ini adalah sebagai berikut:

1. Merancang alat untuk jaringan sensor nirkabel dengan teknik kompresi data untuk menghemat energi dari node
2. Mengimplementasikan algoritma kompresi data pada jaringan sensor nirkabel berbasis arduino
3. Mengetahui pengaruh kompresi data terhadap data yang ditransmisikan
4. Mengetahui pengaruh kompresi data terhadap efisiensi energi dari node

1.5 Metodologi

Metodologi yang diterapkan dalam penerapan tugas akhir ini terdiri dari tahapan-tahapan sebagai berikut:

1. Studi Literatur
   Pada tahap ini, dilakukan pencarian dan penggalian informasi dari berbagai sumber referensi diantaranya buku, internet, dan tugas akhir, sebelumnya. Di tahap ini, juga dilakukan observasi mengenai perangkat yang akan dilakukan di penelitian tugas akhir ini.

2. Simulasi
   Pada tahap ini, dilakukan percobaan, simulasi, dari metode kompresi yang akan diimplementasikan.
3. Desain Alat
   Pada tahap ini, dilakukan perancangan dan implementasi algoritma kompresi ke dalam perangkan jaringan sensor nirkabel.
4. Pengukuran
   Pada tahap ini, dilakukan pengukuran dari rancangan alat yang algoritma kompresinya telah diimplementasikan.
5. Analisis Data
   Pada tahap ini, dilakukan analisis data hasil pengukuran dengan tujuan untuk memudahkan menarik kesimpulan.

1.6 Sistematika Penulisan

Pembahasan tugas akhir ini terbagi dalam lima bab dengan rincian sebagai berikut:

BAB I PENDAHULUAN

Bab ini mencakup latar belakang, perumusan masalah, batasan masalah, tujuan penelitian, metodologi penelitian, sistematika penulisan, dan relevansi dari hasil penelitian.

BAB II TINJAUAN PUSTAKA

Bab ini berisi tentang tinjauan pustaka dan teori-teori yang berhubungan dengan jaringan sensor nirkabel, sensor, mikrokontroler, zigbee, distribusi probabilitas, dan metode kompresi.

BAB III METODOLOGI PENELITIAN

Bab ini akan dijelaskan tentang simulasi dan langkah-langkah dalam merancang perangkat jaringan sensor nirkabel dan pengimplentasian metode kompresi ke dalam perangkat.

BAB IV PENGUKURAN DAN ANALISIS DATA

Pada bab ini akan ditampilkan hasil pengukuran dari perangkat yang telah dirancang, kemudian dilakukan analisis dari data yang telah diperoleh.

BAB V PENUTUP

Bab ini berisi kesimpulan dan saran berdasarkan berbagai proses yang telah dilakukan dalam penelitian ini.

1.7 Relevansi

Adapun relevansi dari hasil penelitian tugas akhir ini adalah sebagai berikut:

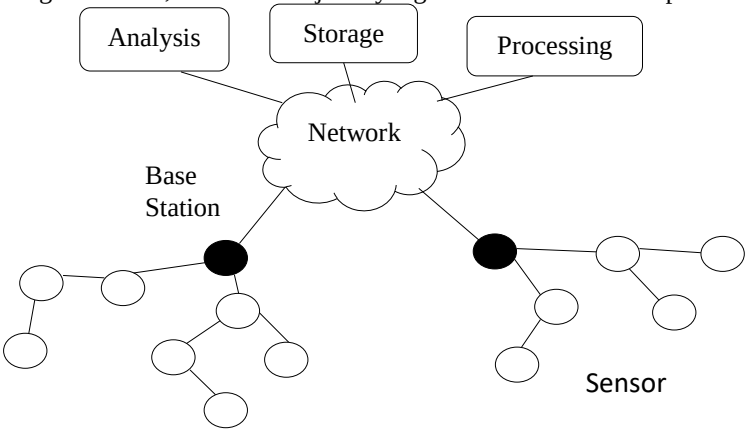
1. Sebagai referensi pengembangan algoritma kompresi data pada jaringan sensor nirkabel berbasis arduino untuk diimplementasikan ke dalam sistem monitoring.
2. Sebagai referensi bagi mahasiswa atau industri untuk melakukan pengembangan algoritma kompresi data pada jaringan sensor nirkabel berbasis Arduino.

BAB II

TINJAUAN PUSTAKA

2.1 Jaringan Sensor Nirkabel

Jaringan sensor nirkabel didefinisikan sebagai sebuah sistem jaringan yang terdiri dari beberapa sensor yang saling saling terhubung dan saling berkomunikasi antar sensor dan bertujuan untuk mendapatkan suatu hasil pengukuran [1]. Jaringan sensor nirkabel mempunyai kemiripan dengan jaringan ad-hoc nirkabel. Kemiripan tersebut yaitu kedua jaringan bergantung pada konektivitas nirkabel dan tidak bergantung pada topologi jaringan. Keunggulan dari jaringan sensor nirkabel antara lain bahan pembuatan murah, pemrograman dan pengaturan jaringan yang fleksibel, hemat energi. Untuk beberapa parameter fisik, teknologi sensor yang ada dapat diintegrasikan ke jaringan sensor nirkabel. Beberapa parameter fisik yang biasanya digunakan yaitu sensor temperatur, kelembaban udara, sensor visual, cahaya inframerah, sinyal akustik, getaran, tekanan, sensor zat kimia, dan sensor magnetik. Beberapa sensor di atas dapat dikombinasikan sehingga membentuk suatu aplikasi dengan tujuan tertentu. Atas dasar sensor yang memiliki penginderaan fisik, dikombinasikan dengan kemampuan komputasi dan komunikasi, berbagai macam aplikasi dapat dibangun, dengan jenis sensor yang sangat berbeda, bahkan dari jenis yang berbeda dalam satu aplikasi.



Gambar 2.1 Ilustrasi jaringan sensor nirkabel

Contoh satu aplikasi yaitu dalam hal kontrol lingkungan. Jaringan sensor nirkabel dapat digunakan untuk mengontrol lingkungan seperti memonitoring sungai dari bahaya limbah atau polusi zat kimia lainnya. Kelebihan dari jaringan sensor nirkabel yang ditunjukkan di salah satu aplikasi ini adalah tanpa perawatan, beroperasi secara nirkabel, untuk keperluan jangka panjang, dan umumnya mempunyai ukuran yang kecil.

Jaringan sensor nirkabel terdiri dari 2 bagian yaitu *node* dan *sink*. Node terdiri dari sensor, kontroler, memori, suplai daya, dan alat komunikasi, biasanya menggunakan modul radio. Peran dari node sendiri dalam jaringan sensor nirkabel yaitu mengirimkan hasil pengukuran dari sensor ke sink. Sedangkan sink sendiri mempunyai bagian-bagian yang sama dengan node namun mempunyai peran yang berbeda. Peran sink adalah untuk menerima hasil pengukuran dari node untuk selanjutnya data hasil pengukuran dimanfaatkan baik manusia atau program dari sink itu sendiri.

Fungsi-fungsi dari komponen utama penyusun sebuah node antara lain:

- Sensor berfungsi untuk mendeteksi adanya perubahan lingkungan secara fisik atau kimia dan mengubahnya menjadi suatu besaran baik analog maupun digital sehingga dapat dibaca oleh rangkaian elektronik
- Kontroler berfungsi untuk mengontrol suatu sistem rangkaian elektronik
- Memori berfungsi sebagai tempat penyimpanan perintah-perintah yang nantinya akan dijalankan oleh rangkaian elektronik
- Suplai Daya sebagai penyuplai energi supaya komponen dan rangkaian dapat beroperasi
- Alat komunikasi radio berfungsi untuk membuat koneksi antar node atau lebih

2.2 Sensor

Sensor merupakan komponen utama dalam sebuah node atau dengan kata lain sensor didefinisikan sebagai sebuah komponen elektronika yang mampu mengubah suatu parameter fisik menjadi sebuah sinyal listrik. parameter fisik yang dimaksud adalah suatu besaran yang dapat diukur. Contoh parameter fisik yang sering diukur atau dideteksi adalah suhu kelembaban, kecepatan, visual, suara, dan

lain-lain. Tanpa sebuah sensor, node tidak akan berfungsi dengan sebagaimana mestinya. Secara garis besar sensor diklasifikasikan menjadi dua kategori, yaitu sensor pasif dan sensor aktif. Perbedaan dari dua kategori ini ialah sensor pasif tidak membutuhkan atau mengeluarkan energi untuk mendapatkan hasil pengukuran sedangkan sensor aktif membutuhkan energi untuk mendapatkan hasil pengukuran. Contoh dari sensor aktif adalah sensor sonar atau radar yang mengeluarkan gelombang [1].

Sensitivitas sensor mengindikasikan seberapa banyak output dari sensor berubah ketika kuantitas yang diukur atau input dari sensor berubah. Misalkan, jika merkuri yang ada di termometer bergerak 1 cm ketika temperatur berubah sebesar 1 °C, sensitivitas dari termometer tersebut yaitu sebesar 1 cm/°C. Beberapa sensor juga dapat memengaruhi apa yang diukur, contohnya adalah ketika termometer dimasukkan ke dalam wadah yang berisi air panas. Air panas akan menaikkan suhu dari termometer sedangkan termometer akan menurunkan suhu dari air panas. Sensor biasanya didesain untuk mempunyai efek yang kecil terhadap apa yang diukur. Teknologi yang akan datang memungkinkan banyak sensor yang akan diproduksi dalam skala mikroskopis, atau bisa juga disebut microsensor [2].

Cara kerja sensor yaitu dengan cara membangkitkan tegangan berdasarkan parameter yang diukur. Besar tegangan yang dibangkitkan proporsional dengan besaran yang diukur. Sensor mendeteksi adanya perubahan dari besaran yang diukur lalu mengubahnya menjadi sinyal listrik, baik tegangan maupun arus. Sebagai contoh, sebuah sensor suhu membangkitkan tegangan dari 0 volt sampai 10 volt. Besar tegangan akan bervariasi dengan suhu yang

Gambar 2.2 Sensor DHT11

diukur. Misalkan, tegangan 0 volt menunjukkan bahwa suhu ruangan berada pada 0 °C atau tegangan 10 volt menunjukkan bahwa suhu ruangan berada pada 100 °C.

Contoh-contoh sensor yang sering diaplikasikan dalam rangkaian elektronik adalah sensor suhu, kelembaban, atau sensor proximity. DHT11 merupakan sensor digital yang dapat mengukur besaran suhu dan kelembaban. DHT11 sangat mudah digunakan, kestabilan yang tinggi, dan juga dapat dikalibrasi sehingga sangat mudah diaplikasikan dan banyak diaplikasikan seperti sistem monitoring.

2.3 Mikrokontroler

Mikrokontroler merupakan sebuah kontroler berukuran kecil yang terintegrasi dari beberapa komponen elektronika yang berfungsi sebagai pengontrol rangkaian elektronik dengan komponen inti yaitu mikroprosesor. Komponen penyusun mikrokontroler umumnya terdiri dari CPU berupa mikroprosesor, Memori baik RAM maupun ROM, port I/O, dan juga ADC.

Mikrokontroler memiliki beberapa keunggulan. Keunggulan utama yang dimiliki dari mikrokontroler yaitu Adanya RAM dan ROM yang dapat diprogram berulang-ulang dan port I/O (Input dan Output) sehingga board mikrokontroler yang digunakan menjadi ringkas dan praktis. Keunggulan lain yang dimiliki dari mikrokontroler antara lain mempunyai periferal seperti timer atau watchdog, pembangkit clock, dan Analog-to-Digital Converter (ADC).

Gambar 2.3 Arduino UNO

Arduino Uno adalah sebuah mikrokontroler open source berbasis mikroprosesor ATmega328P. Arduino Uno memiliki 14 pin digital input output, 6 diantaranya dapat digunakan sebagai output PWM (Pulse width modulation), 6 input analog, 16 MHz kristal kuarsa, koneksi USB, power jack, header ICSP dan tombol reset. Arduino Uno mempunyai yang dibutuhkan untuk mendukung mikrokontroler, praktis dan simpel dengan menghubungkan arduino uno ke komputer dengan kabel USB atau menggunakan suplai daya dari adapter AC-to-DC atau bisa juga menggunakan baterai [3].

Mikrokontroler ini digunakan untuk mengolah data yang dihasilkan dari sensor. Pengolahan data yang dimaksud antara lain dapat berupa penentuan alamat tujuan, kompresi dan dekompresi, peningkatan keamanan data, dan lain-lain. Selain melakukan pengolahan data yang dihasilkan oleh sensor, mikrokontroler juga memberikan perintah kepada komponen yang bertujuan untuk berkomunikasi dengan mikrokontroler lain.

Arduino menyediakan sebuah aplikasi bernama Arduino IDE yang berfungsi untuk melakukan penulisan dan compiling kode yang akan diunggah ke arduino dan fungsi lainnya, mengunggah program yang sudah di-compile sebelumnya melalui koneksi USB. Arduino IDE mendukung dalam berbagai sistem operasi seperti Windows,

Gambar 2.4 Software Arduino IDE

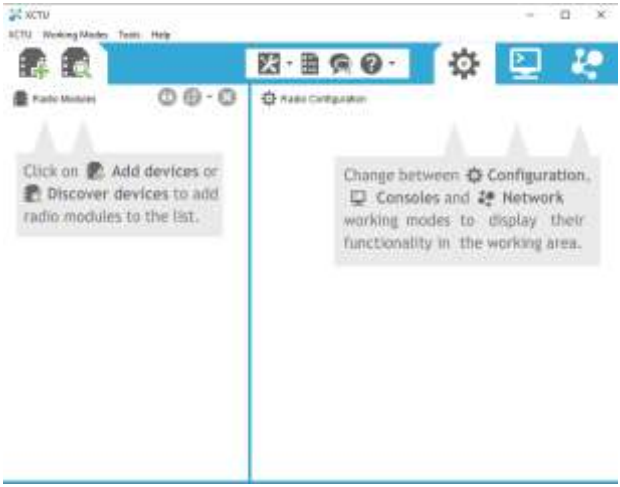
Linux, atau MacOS. Bahasa pemrograman yang digunakan dalam aplikasi Arduino IDE adalah bahasa pemrograman C sehingga penulisan kode dapat dilakukan dengan mudah. Namun bahasa pemrograman C juga memiliki kekurangan. Pada dasarnya, penulisan kode dalam Arduino IDE dibagi menjadi 2 bagian yaitu bagian kode yang dijalankan satu kali dan bagian kode yang dijalankan berkali-kali.

2.4 Zigbee

Zigbee adalah sebuah modul radio dengan protokol komunikasi radio digital yang mempunyai daya rendah berdasarkan standard IEEE 802.15.4. Jangkauan yang dijangkau oleh zigbee dapat mencapai 100 meter dengan syarat antara pemancar dan penerima harus line of sight [4]. Harga zigbee pun juga terjangkau sehingga zigbee memungkinkan digunakan di banyak aplikasi di dunia nyata seperti aplikasi monitoring atau untuk mendeteksi kejadian. Zigbee dapat menggunakan baterai dengan daya yang kecil dikarenakan zigbee mempunyai daya yang rendah. Zigbee juga mendukung jaringan mesh sehingga zigbee mempunyai reliabilitas yang tinggi dan juga memperluas area yang dijangkau.

Xbee adalah nama merek dari keluarga modul radio, yang dikeluarkan oleh Digi International dan berbasis standar IEEE 802.15.4 didesain untuk komunikasi point-to-point dan topologi star dengan baudrate 250 kbit/s saat di udara. Xbee membutuhkan daya yang sangat kecil untuk beroperasi dengan baik, yaitu sebesar 1 mW. Untuk pengoperasian Xbee sendiri sangatlah mudah. Xbee hanya membutuhkan suplai daya tegangan sebesar 3.3 V, data in dan data

Gambar 2.5 Modul radio Xbee



Gambar 2.6 Software XCTU

out (UART), dan juga dapat ditambahkan seperti tombol reset dan pengaturan sleep [4].

Pengaturan dasar Xbee dapat diatur dengan software yang dikeluarkan oleh Digi International bernama XCTU. Pengaturan yang dapat dilakukan yaitu seperti pembaharuan firmware, peran node sebagai pengirim atau penerima, dan kondisi atau status dari Xbee. Beberapa status diantaranya seperti Xbee bekerja di kanal berapa, Xbee mempunyai PAN ID berapa atau serial number dari Xbee.

2.5 Distribusi Probabilitas

Dalam teori probabilitas dan stokastik, distribusi probabilitas adalah fungsi matematika yang menyediakan probabilitas terjadinya berbagai kemungkinan hasil dalam percobaan [5]. Dalam istilah yang lebih teknis, distribusi probabilitas adalah deskripsi dari fenomena acak dalam hal probabilitas suatu kejadian. Misalnya, jika variabel acak X digunakan untuk menunjukkan hasil lemparan koin, maka distribusi probabilitas X akan mengambil nilai 0,5 untuk X = kepala, dan 0,5 untuk X = ekor.

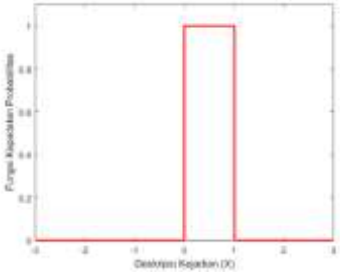
Distribusi probabilitas secara umum dibagi menjadi dua bagian. Distribusi probabilitas diskrit (berlaku untuk skenario di mana mempunyai kemungkinan himpunan hasil berupa nilai diskrit, seperti

lemparan koin atau gulungan dadu) dapat dikodekan dengan daftar diskrit dari probabilitas hasil, yang dikenal sebagai fungsi massa probabilitas. Di sisi lain, distribusi probabilitas kontinu (berlaku untuk skenario di mana himpunan hasil yang mungkin dapat mengambil nilai dalam rentang kontinu (misalnya bilangan real), seperti suhu pada hari tertentu) biasanya dijelaskan oleh fungsi kepadatan probabilitas.

2.5.1 Distribusi Uniform

Distribusi uniform kontinu merupakan distribusi probabilitas yang sederhana di antara distribusi probabilitas yang lain. Dalam distribusi uniform, Setiap variabel acak mempunyai probabilitas terjadi yang sama. Distribusi ini sering dinotasikan dengan U(a,b) dengan a merupakan nilai minimum dan b merupakan nilai maksimum. Fungsi densitas probabilitas distribusi uniform adalah

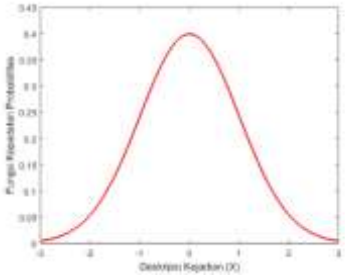
$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & x < a \; atau \; x > b \end{cases} \tag{2.1}$$



Gambar 2.7 Grafik fungsi kepadatan probabilitas dari distribusi uniform

2.5.2 Distribusi Normal

Distribusi normal adalah distribusi probabilitas kontinu yang sangat umum. Distribusi normal penting dalam statistik dan sering digunakan dalam ilmu alam dan sosial untuk mewakili variabel acak bernilai nyata yang distribusinya tidak diketahui. Variabel acak dengan distribusi Gaussian dapat dikatakan terdistribusi normal.



Gambar 2.8 Grafik fungsi kepadatan probabilitas dari distribusi normal

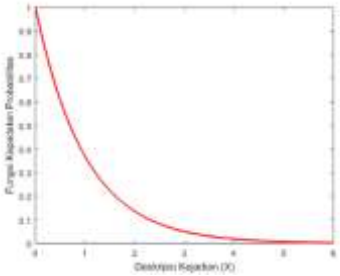
Fungsi densitas probabilitas distribusi normal adalah

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, -\infty < x < \infty \tag{2.2}$$

2.5.3 Distribusi Eksponensial

Distribusi eksponensial, atau disebut juga eksponensial negatif, adalah distribusi probabilitas yang mendeskripsikan waktu antara peristiwa dalam proses Poisson, yaitu proses dimana peristiwa terjadi secara terus menerus dan independen dengan laju rata-rata konstan. Distribusi eksponensial memoryless atau tidak memiliki memori, dan juga termasuk kasus khusus dalam distribusi gamma. Fungsi densitas probabilitas distribusi eksponensial adalah

$$f(x) = \lambda e^{-\lambda x}, x \geq 0 \tag{2.3}$$



Gambar 2.9 Grafik fungsi kepadatan probabilitas dari distribusi eksponensial

2.6 Kompresi

Dalam konteks pemrosesan sinyal, kompresi data merupakan teknik untuk mengkodekan informasi dengan menggunakan bit yang lebih sedikit daripada bit aslinya untuk merepresentasikan informasi tersebut. Kompresi dapat berupa kompresi lossy atau kompresi lossless. Kompresi lossless mengurangi bit dengan mengidentifikasi dan menghilangkan redudansi statistik. Tidak ada informasi yang hilang dalam kompresi lossless. Sedangkan kompresi lossy mengurangi bit dengan menghapus informasi yang tidak dibutuhkan atau mengurangi informasi yang tidak penting [6].

Proses mengurangi ukuran data sering disebut sebagai kompresi data. Dalam konteks pengiriman data, disebut pengkodean sumber. Pengkodean dilakukan pada sumber data sebelum disimpan atau dikirim. Pengkodean sumber tidak boleh dikacaukan dengan pengkodean saluran, untuk deteksi kesalahan dan koreksi atau pengkodean garis, sarana untuk memetakan data ke sinyal.

Kompresi data bermanfaat karena kompresi mengurangi sumber daya yang dibutuhkan untuk menyimpan dan mengirim data. Sumber daya komputasi dibutuhkan dalam proses kompresi dan biasanya, juga dibutuhkan dalam proses kebalikan dari kompresi, yaitu dekompresi. Secara ringkas, kompresi data menukar sumber daya penyimpanan menjadi sumber daya waktu dan komputasi. Misalnya, Kompresi untuk video membutuhkan perangkat keras yang mahal dan mumpuni supaya video yang akan dikompresi cukup cepat agar video dapat dilihat disaat juga melakukan dekompresi secara bersamaan. Desain dari skema kompresi data melibatkan pengorbanan dari beberapa faktor, yaitu rasio kompresi, besar distorsi (ketika menggunakan kompresi lossy), dan sumber daya komputasi dibutuhkan untuk melakukan kompresi dan dekompresi data.

Salah satu faktor penting yang digunakan untuk mengukur seberapa besar data yang terkompresi adalah rasio kompresi. Rasio kompresi mengukur kuantitas dari data yang terkompresi dan dibandingkan dengan kuantitas dari data aslinya. Fungsi dari rasio kompresi:

$$Rasio\ Kompresi = \frac{Panjang\ data\ original}{Panjang\ data\ kompresi} \tag{2.4}$$

Dari persamaan diatas, dapat dilihat bahwa semakin besar rasio kompresi, maka semakin baik teknik kompresi yang digunakan. Cara lain yang digunakan untuk mengukur seberapa baik teknik kompresi yang digunakan adalah dengan menggunakan figure of merit. Figure of merit merupakan fungsi kebalikan dari rasio kompresi, yaitu dengan persamaan:

$$Figure\ of\ Merit = \frac{Panjang\ data\ kompresi}{Panjang\ data\ original} \quad (2.5)$$

Dalam menurunkan algoritma kompresi, banyak cara dalam pendekatannya [7]. Pendekatan sistematis dapat dibagi menjadi 8 tahap sebagai berikut:

1. Deskripsi Permasalahan
   Dalam permasalahan kompresi, dari sudut pandang algoritma, adalah bagaimana cara menemukan sebuah algoritma yang efektif dan efisien untuk menghilangkan berbagai redudansi dari berbagai tipe data.
2. Model Matematika
   Permodelan merupakan sebuah pengaturan dalam suatu lingkungan untuk memberikan variable supaya dapat diobservasi atau perilaku tertentu dari siste untuk dieksplorasi.
3. Desain Algoritma
   Mendesain algoritma merupakan tugas yang menantang dan menarik. Tekniknya sangat bergantung atas pemilihan model matematika. Kita mungkin menambahkan detail lebih ke dalam model, mempertimbangkan feedback untuk realisasi model, menggunakan Teknik algoritma kompresi yang standar. Kita juga dapat melakukan pendekatan top-down, dan mengidentifikasi algoritma yang ada dan efisien untuk mendapatkan solusi parsial supaya dapat memecahkan masalah.
4. Verifikasi Algoritma
   Dalam tahap verifikasi algoritma, kita memeriksa kebenaran dari algoritma, kualitas kompresi, dan efisiensi dari coder. Hal ini relatif mudah untuk mengecek kualitas kompresi. Contohnya, kita dapat menggunakan rasio kompresi untuk melihat seberapa efektif kompresi yang diperoleh. Sedangkan efisiensi coder didefinisikan sebagai perbedaan antara panjang rata-rata dari codeword dan entropi.

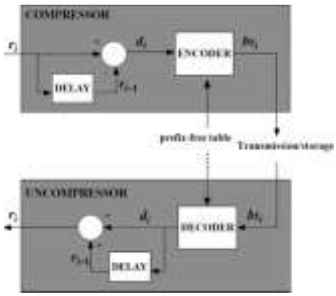
5. Estimasi dari Kompleksitas Komputasi
   Dalam tahap estimasi, sangat mungkin untuk memerkirakan dan memrediksi perilaku dari software yang dikembangkan menggunakan sumber daya yang minimum. Karena itu, kita harus membandingkan minimal dua calon algoritma dalam bentuk efisiensi, umumnya efisiensi waktu. Algoritma yang lebih efisien juga harus dicek untuk memastikan bahwa algoritma memenuhi batasan teori tertentu sebelum dipilih dan diimplementasikan.
6. Implementasi
   Dalam pengimplementasian, tidak ada batasan pada bahasa pemrograman tingkat tinggi yang digunakan dan bagaimana prosedur atau fungsi implementasinya.

2.6.1 Lossless Entropy Coding

Lossless entropy coding (LEC) mengeksploitasi versi modifikasi dari kode Eksponensial-Golomb (Exp-Golomb) orde 0, yang merupakan jenis kode universal. Ide dasarnya adalah untuk membagi alfabet angka menjadi kelompok-kelompok yang ukurannya meningkat secara eksponensial. Codeword dalam pengkodean Golomb adalah gabungan dari kode unary dan kode binary: lebih jelasnya, kode unary untuk menentukan grup, sedangkan kode biner untuk mewakili indeks dalam grup [8].

Dalam LEC, pendekatannya membagi alfabet angka ke dalam kelompok yang ukurannya meningkat secara eksponensial, tetapi kelompok diberi kode entropi, bukan kode unary. Modifikasi ini memperkenalkan kemungkinan menentukan kode awalan yang bebas untuk grup. Selain itu, karena versi asli Exp-Golomb hanya mengelola bilangan bulat non-negatif, modifikasi kecil pada skema asli dapat memetakan nilai aktual ke domain non-negatif.

Dalam unit penginderaan dari node sensor, setiap pengukuran m_i yang diperoleh oleh sensor dikonversi oleh ADC ke representasi biner r_i pada R bit, dimana R adalah resolusi dari ADC, yaitu, jumlah (2^R) dari nilai diskrit yang ADC dapat hasilkan dalam rentang nilai analog. Gambar 2.10 menunjukkan skema blok dari pendekatan LEC. Untuk setiap akuisisi baru m_i, LEC menghitung selisih $d_i = r_i - r_{i-1}$, yang merupakan input ke enkoder entropi (untuk menghitung d_0 kita asumsikan bahwa r_{-1} sama dengan nilai pusat di



Gambar 2.10 Blok diagram dari kompresor dan dekompresor [8]

antara 2^R nilai diskrit yang mungkin). Encoder entropi melakukan kompresi lossless dengan mengkodekan selisih secara lebih pendek berdasarkan karakteristik statistik mereka. Setiap nilai yang bukan bernilai nol diwakili sebagai urutan bit b_{si} yang terdiri dari dua bagian $s_i|a_i$, di mana s_i merupakan hasil pengkodean jumlah n_i dari bit yang diperlukan untuk mewakili d_i (yaitu, grup yang menjadi milik d_i) dan a_i sendiri adalah representasi dari d_i (yaitu, posisi indeks dalam grup). Ketika d_i sama dengan 0, grup yang sesuai memiliki ukuran sama dengan 1 dan oleh karena itu tidak perlu untuk mengkodekan posisi indeks dalam grup yang berarti bahwa a_i tidak diwakili.

Untuk nilai sebarang yang tidak bernilai nol, n_i dihitung sebagai $\lceil \log_2 (|d_i|) \rceil$: biasanya n_i sama dengan R. Dengan demikian, untuk mengkodekan n_i, tabel untuk grup entri R+1 harus ditentukan. Tabel ini tergantung pada distribusi selisih d_i. Untuk mengelola angka negatif d_i, LEC memetakan perbedaan input ke indeks non-negatif, menggunakan:

$$index = \begin{cases} d_i\,, & d_i \geq 0 \\ 2^{ni} - 1 - |d_i|\,, & d_i < 0 \end{cases} \qquad (2.5)$$

Prosedur yang digunakan untuk menghasilkan ai menjamin bahwa semua nilai yang mungkin memiliki kode yang berbeda. Dengan menggunakan Tabel 1, kita memiliki, misalnya, bahwa di = 0, di = +1, di = −1, di = +255 dan di = −255 dikodekan, masing-masing sebagai 00, 010|1, 010|0, 111110|11111111 dan 111110|00000000. Setelah bsi dihasilkan, ditambahkan ke bitstream yang membentuk hasil kompresi.

Tabel 2.1 Kode grup dalam LEC

n_i	s_i	d_i
0	00	0
1	010	±1
2	011	±2, ±3
3	100	±4, . . . , ±7
4	101	±8, . . . , ±15
5	110	±16, . . . , ±31
6	1110	±32, . . . , ±63
7	11110	±64, . . . , ±127
8	111110	±128, . . . , ±255
9	1111110	±256, . . . , ±511
10	11111110	±512, . . . , ±1023
11	111111110	±1024, . . . , ±2047
12	1111111110	±2048, . . . , ±4095
13	11111111110	±4096, . . . , ±8191
14	111111111110	±8192, . . . , ±16383
15	1111111111110	±16384, . . . , ±32767

Gambar 2.11 menunjukkan pseudo-code fungsi dari *compress*() berdasarkan algoritma LEC. Rangkaian dari deretan bit *a* dan *b* ditunjukkan sebagai (*a*, *b*). Setelah menghitung selisih d_i antara representasi biner selisih yang sekarang r_i dan selisih yang sebelumnya r_{i-1}, fungsi *compress*() memanggil fungsi *encode*() dengan parameter input dari selisih d_i. Fungsi *compress*() mengembalikan deretan bit bs_i sesuai dengan selisih d_i.Deret bs_i setelah itu digabungkan dengan aliran deretan bit yang sudah dibangkitkan sebelumnya.

Gambar 2.12 menunjukkan fungsi dari encode(). Disini pertama berapa angka n_i dari bit yang dibutuhkan untuk mengkodekan

```
compress(ri, ri_1, stream)
// menghitung selisih di
SET di TO ri – ri_1
// mengkodekan selisih di
CALL encode() with di RETURNING bsi
// menambahkan bsi ke aliran deretan bit
SET stream TO <<stream,bsi>>
RETURN stream
```

Gambar 2.11 Pseudo-code dari fungsi *compress*()

```
encode(di)
// menghitung kategori dari di
IF di = 0
        SET ni TO 0
ELSE
        SET ni TO ⌈log2 (|di|)⌉
ENDIF
// mengekstrak si dari tabel
SET si TO Tabel[ni]
// menyusun bsi
IF ni = 0 THEN
        // ai tidak dibutuhkan
        SET bsi TO si
ELSE
        // menyusun ai
        IF di > 0 THEN
                SET ai TO (di)|ni
        ELSE
                SET ai TO (di-1)|ni
        ENDIF
        // menyusun bsi
        SET bsi TO <<si,ai>>
ENDIF
RETURN stream
```

Gambar 2.12 Pseudo-code dari fungsi *encode*()

nilai dari d_i dihitung. Kemudian, bagian dari deretan bit bs_i, s_i, dibangkitkan menggunakan table yang berisi kamus yang diadopsi kompresor entropi. Terakhir, bagian dari deretan bit bs_i, a_i, dibangkitkan.

Logaritma biner pada umumnya tidak disediakan set instruksi dari mikroprosesor dalam node dan oleh karena itu, $\lceil \log_2 (|d_i|) \rceil$ tidak dapat langsung dieksekusi. Untuk mengatasi masalah ini, kita dapat menggunakan fungsi *computeBinaryLog*() yang ditunjukkan oleh gambar 2.13. Fungsi ini mengembalikan angka n_i dari bit yang dibutuhkan untuk mengkodekan nilai dari d_i hanya dengan menggunakan pembagian integer. Dari melihat penjelasan diatas, algoritma kompresi LEC sangat simpel dan hanya membutuhkan memori untuk menyimpan s_i menurut table 2.1.

```
computeBinaryLog(di)
// ⌈log2 (|di|)⌉
SET ni TO 0
WHILE di > 0
        SET di TO di/2
        SET ni TO ni + 1
ENDWHILE
RETURN ni
```

Gambar 2.13 Pseudo-code dari fungsi *computeBinaryLog*()

2.6.2 Sequential Lossless Entropy Coding

Dalam algoritma lossless entropy coding (LEC), residu, atau selisih, dalam deretan nilai residu $\{r_i\}$ (i = 1,2,3,...,*M*, dimana *M* merupakan ukuran dari blok data), yang sedang dikodekan dalam enkoder entropi, dipertimbangkan tidak mempunyai korelasi diantaranya, dan demikian juga dikodekan secara independen. Wawasan yang digunakan dalam sequential lossless entropy coding (S-LEC) adalah karena predictor selisih yang simpel merupakan hal umum untuk sebarang aliran data sensor, hal ini tidak dapat menangkap dan menghilangkan secara keseluruhan dari korelasi temporal diantara data sensor yang deretan selisihnya berubah-ubah. Berdasarkan dengan asumsi implisit dari residu yang independen, LEC akan menunjukkan performa yang layak jika karakteristik korelasi dari aliran data sensor ditangkap oleh predictor selisih kurang lebih, tetapi LEC akan menunjukkan performa yang buruk jika sebaliknya. Untuk mengatasi isu ini, S-LEC memperkenalkan kode sekuensial dan mengembangkan LEC.

Sequential Lossless entropy coding (S-LEC) merupakan metode kompresi hasil lanjutan dari Lossless entropy coding (LEC) untuk mengatasi mengenai masalah ketangguhan dengan cara mengambil dari data sebelumnya. S-LEC memperkenalkan kode sekuensial tambahan ke dalam codeword-nya, dan karenanya memperluas codeword r_i dengan LEC dari $h_i|a_i$ menjadi $s_i|h_i|a_i$. Ide dasarnya adalah bahwa jika residu, yaitu selisih data yang sekarang dan data berikutnya, berikutnya r_{i+1} memiliki grup yang sama atau grup tetangganya sebagai residu r_i, kode grup h_{i+1} untuk r_{i+1} dapat disimpulkan dari h_i sebelumnya dan dengan demikian, codeword r_{i+1} adalah $s_{i+1}|a_{i+1}$ daripada $s_{i+1}|h_{i+1}|a_{i+1}$. Codeword ini mengurangi ukuran berdasarkan informasi konteks yang berdekatan di antara residu. Untuk mengkodekan informasi konteks sekuensial, dua bit s_i

Tabel 2.2 Kode Sekuensial dalam S-LEC

s_i	Konteks Informasi	Penjelasan
00	$h_i = h_{i-1}$	Grup yang sama
01	$h_i = \begin{cases} h(n_{i-1}-1), n_{i-1} \geq 1 \\ h(2) \quad\quad , n_{i-1} = 0 \end{cases}$	1 grup dibawah
10	$h_i = \begin{cases} h(n_{i-1}-1), n_{i-1} < K \\ h(2) \quad\quad , n_{i-1} = K \end{cases}$	1 grup diatas
11	h_i tidak dapat dihilangkan, dan $s_i \| h_i \| a_i$ dibutuhkan	Lain-lain

disusun dalam algoritma S-LEC dan ditentukan pada Tabel 2.2. Dalam S-LEC, s_1 selalu dihilangkan, untuk residu pertama r_1 dari blok data hasil akuisisi data dari sensor, dan karenanya h_i harus ada [9].

Dalam kasus s_i = 11, ketika kode grup h_i tidak dapat dihilangkan sepenuhnya, kode grup h_i dapat dikurangi ukurannya dalam beberapa situasi konteks tanpa ambiguitas. Untuk menyelidiki kemungkinan ini, biarkan semua kelompok LEC selanjutnya dikelompokkan menjadi tiga kelompok sebagai berikut: $C_1 = \{n_i | i = 0, 1, 2, 3\}$, $C_2 = \{n_i \mid i = 4, 5\}$, dan $C_3 = \{n_i \mid i = 6, \ldots, K\}$, dimana K adalah jumlah bit pembacaan sensor, yaitu ketelitian konverter A/D (ADC).Kemudian diberikan aturan pengurangan h_i berdasarkan $r_{i-1} \in C_j$ (j = 1, 2, 3) pada Tabel 2.3 ketika s_i = 11 dan $n_i > n_{i-1}$.

Sebagai contoh, mari perhatikan deretan residu r_1 = 9, r_2 = 128, r_3 = −130, r_4 = 16, dan r_5 = −32, yang akan dikodekan oleh S-LEC menjadi 101|1001, 11|1110|10000000, 00|01111101, 11|110|10000, 10|011111. Sebaliknya, deretan residu yang sama akan dikodekan oleh LEC menjadi 101|1001, 111110|10000000, 111110|01111101, 110|10000, 1110|011111.

Dalam pengimplementasian algoritma S-LEC ke dalam jaringan dengan alat yang bernama telosB, algoritma S-LEC lebih unggul dari model Kruskal-Wallis dan model Bartlett [10]. Algoritma S-LEC unggul dari jumlah pengukuran. Jumlah pengukuran yang didapat di sink dengan algoritma S-LEC lebih banyak dari pada model lainnya. Namun jika dibandingkan dengan model lainnya dalam hal seperti energi yang dikonsumsi dan data loss, model Kruskal-Wallis lebih unggul.

Tabel 2.3 Reduksi Kode Grup

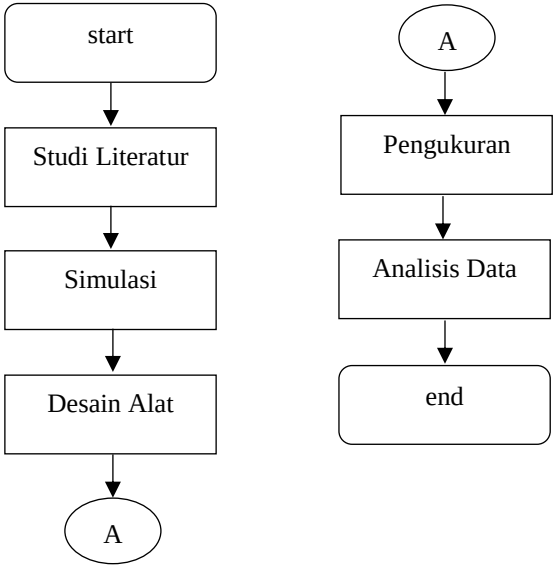
r_{i-1}	Reduksi terjadi ketika $s_i = 11$ dan $n_i > n_{i-1}$	Penjelasan
C_1	Reduksi "1" h_i	Menghilangkan '1' pertama pada LEC h_i, misal "1110" menjadi "110"
C_2	Reduksi "11" h_i	Menghilangkan '11' pertama pada LEC h_i, misal "11110" menjadi "110"
C_3	Reduksi "111" h_i	Menghilangkan '111' pertama pada LEC h_i, misal "111110" menjadi "110"

BAB III

METODE PERANCANGAN DAN IMPLEMENTASI

3.1 Metodologi Penelitian

Penelitian yang dilakukan pada penelitian tugas akhir ini dilakukan melalui beberapa tahapan. Tahap awal yang dilakukan pada penelitian tugas akhir ini adalah studi literatur. Tahapan studi literatur merupakan sebuah kegiatan yang dilakukan untuk mencari referensi dari berbagai sumber informasi. Sumber informasi yang dimaksud antara lain, buku, paper, internet, dan tugas akhir sebelumnya. Di tahap ini, juga dilakukan penggalian informasi mengenai perangkat keras yang akan digunakan di penelitian tugas akhir ini. Hasi dari tahap studi literatur adalah algoritma kompresi yang akan diimplementasikan, perangkat keras yang akan digunakan, dan bagaimana cara pengujiannya. Selain perangkat keras, perlu disiapkan juga perangkat lunak untuk konfigurasi alat yang akan digunakan. Perangkat lunak yang akan digunakan yaitu Arduino IDE dan XCTU.



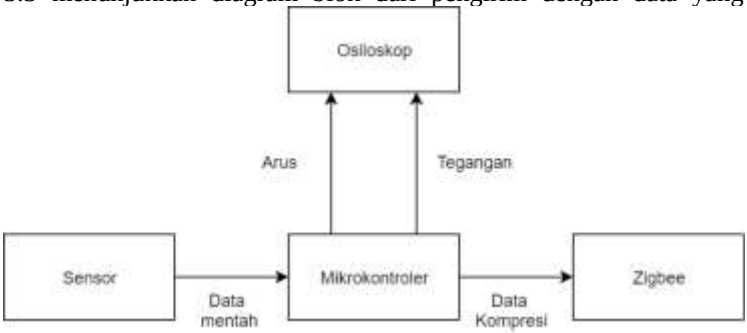
Gambar 3.1 Alur metodologi penelitian

Sebelum diimplementasikan ke dalam Arduino, algoritma kompresi akan disimulasikan terlebih dahulu. Simulasi algoritma kompresi dilakukan di MATLAB. Setelah dilakukannya simulasi, algoritma kompresi akan diimplementasikan ke dalam Bahasa pemrograman Arduino. Pengujian alat dilakukan dalam 2 bagian, yaitu data angka yang diambil dari sensor suhu dan kelembaban, dan data angka yang dibangkitkan acak dengan distribusi probabilitas tertentu. Topologi yang dipakai adalah topologi *point-to-point.* Setelah dilakukan penujian, analisis data dan pengukuran akan dilakukan menggunakan clamp meter dan osiloskop. Parameter yang diukur adalah arus dan tegangan dengan tujuan untuk mengetahui seberapa besar energi yang dikonsumsi oleh 1 node.

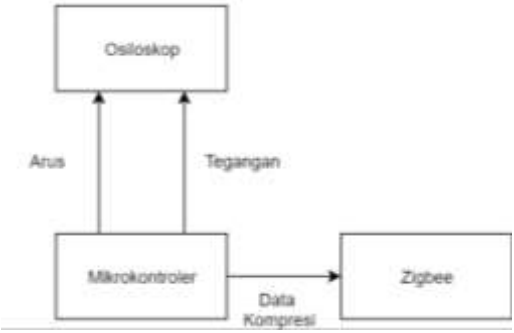
3.2 Perancangan Hardware

Sebelum dilakukannya implementasi algoritma kompresi data, hal yang perlu dilakukan adalah merancang sistem dari hardware. Rancangan jaringan dalam tugas akhir ini menggunakan jaringan point-to-point. Gambar 3.2 menunjukkan diagram blok dari pengirim dengan sensor. Sensor mengambil data baik suhu maupun kelembaban lalu data tersebut dimasukkan ke dalam mikrokontroler. Di dalam mikrokontroler, data tersebut diolah dan dikompresi. Setelah data berhasil dikompresi, hasil kompresi diteruskan ke zigbee supaya dapat dikirimkan.

Dalam tugas akhir ini, data yang digunakan dibagi menjadi 2, yaitu data riil yang didapatkan dari sensor dan data angka yang dibangkitkan sendiri dengan distribusi probabilitas tertentu. Gambar 3.3 menunjukkan diagram blok dari pengirim dengan data yang



Gambar 3.2 Diagram blok node pengirim dengan sensor



Gambar 3.3 Diagram blok node pengirim dengan data yang dibangkitkan sendiri

dibangkitkan sendiri.

Data yang dikirimkan oleh node pengirim nantinya akan diterima oleh node penerima. Data yang dikirimkan diterima oleh zigbee penerima. Selanjutnya zigbee meneruskan hasil kompresi data yang diterima ke mikrokontroler yang tersambung dengan laptop supaya data yang diterima dapat ditampilkan oleh laptop. Gambar 3.4 menunjukkan diagram blok dari node penerima.

3.3 Instalasi Perangkat Lunak

Dari beberapa komponen-komponen yang digunakan, ada komponen-komponen yang perlu dikonfigurasi menggunakan perangkat lunak sebelum disusun menjadi sebuah alat. Konfigurasi yang diperlukan adalah pengaturan pengirim dan penerima, penulisan kode pemrograman untuk implementasi algoritma. Berikut adalah perangka lunak yang diguanakan, cara instalasinya, dan cara mengonfigurasi komponen yang digunakan.

3.3.1 Perangkat Lunak Arduino IDE

Untuk melakukan konfigurasi pada mikrokontroler Arduino UNO, diperlukan perangkat lunak bernama Arduino IDE. Perangkat

Gambar 3.4 Diagram blok node penerima

Gambar 3.5 Tampilan awal Arduino IDE

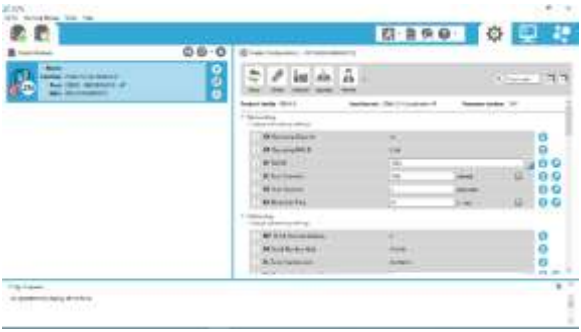
lunak Arduino IDE dapat diunduh pada website resmi Arduino sendiri. Versi Arduino IDE yang digunakan adalah versi 1.8.9. Setelah perangkat lunak Arduino IDE terinstal, Arduino IDE dapat dijalankan dan akan keluar tampilan seperti pada gambar 3.5. Untuk melakukan penulisan dan mengunggah kode pemrograman, pengaturan awal perlu dilakukan. Beberapa hal yang perlu dilakukan agar kode dapat diunggah yaitu sebagai berikut:

1. Menentukan port serial USB dimana Arduino dihubungkan
2. Menentukan tipe board Arduino
3. Melakukan compile pada kode pemrograman agar tidak terjadi error
4. Mengunggah kode pemrograman ke Arduino

Konfigurasi diatas dapat dilakukan dengan memilih perintah pada taskbar di Arduino IDE. Untuk bahasa pemrograman, Arduino IDE menggunakan bahasa C atau C++. Namun pada Arduino IDE, bahasa pemrogramannya mempunyai beberapa syntax yang berbeda.

3.3.2 Perangkat Lunak XCTU

Perangkat lunak XCTU digunakan untuk mengonfigurasi perangkat XBee agar dapat berkomunikasi dengan satu sama lain. Versi perangkat lunak XCTU yang digunakan adalah versi 6.4.2 dan



Gambar 3.6 Tampilan awal XCTU

perangkat lunak ini dapat diunduh secara gratis di web resmi dari Digi. XCTU menyajikan banyak sekali fitur jika ingin mengonfigurasikan perangkat dengan protocol zigbee seperti XBee S2. Pengaturan utama dari XCTU adalah XCTU dapat mengatur XBee apakah sebagai *Coordinator* atau sebagai *End Device.* Pengaturan yang lain pada perangkat lunak XCTU adalah pengaturan alamat pengiriman, mode *Sleep*, besar eergi yang dikeluarkan, dan lain-lain. Tampilan awal XCTU dapat dilihat pada gambar 3.6.

Pada tugas akhir ini, perangkat yang akan digunakan pada tugas akhir ini untuk mengirimkan data yang berasal dari Arduino UNO adalah XBee S2. Cara pengiriman yang akan dilakukan adalah *point-to-point*. Jadi, perangkat yang dibutuhkan dan dikonfigurasi hanyalah 2 XBee S2, yaitu sebagai *coordinator* dan *end device*.

Untuk mengonfigurasi perangkat XBee, ini pertama-tama instal XCTU dengan benar. Setelah terinstal dengan benar, buka XCTU lalu pada menu taskbar pilih "Add Radio Module" untuk menambahkan perangkat XBee S2 ke dalam XCTU dan pilih port serial yang sesuai dengan XBee S2. Pada pengaturan firmware, pilih XB24-B untuk *product* family dan pilih *function set* sesuai dengan peran dari XBee, yaitu sebagai *coordinator* atau sebagai *end device.* Pembaharuan *firmware* dapat dilakukan jika ada pemberitahuan dari XCTU. Setelah itu, Konfigurasi parameter pada XBee S2 dapat dilakukan. Konfigurasi parameter XBee S2 Coordinator dan End Device dapat dilihat pada tabel 3.1 dan tabel 3.2. Setelah kedua XBee selesai dikonfigurasi, XBee sudah siap digunakan dan dipasangkan dengan Arduino UNO.

Tabel 3.1 Konfigurasi XBee S2 Coordinator

AT Command	Function	Value
ATID	Personal Area Network ID	1234
ATSH	Serial Number High	13A200
ATSL	Serial Number Low	40B58D7C
ATDH	Destination Address High	0
ATDL	Destination Address Low	FFFF
ATBH	Broadcast Radius	0
ATBD	Baud Rate	9600

3.4 Simulasi Distribusi Probabilitas pada MATLAB

Sebelum dilakukan implementasi pada Arduino UNO, sebelumnya angka yang akan dibangkitkan dengan disribusi probabilitas tertentu akan disimulasikan terlebih dahulu dengan menggunakan MATLAB supaya lebih mudah saat penulisan kode pemrograman dalam bahasa C pada perangkat lunak Arduino IDE. Simulasi pada MATLAB akan mensimulasikan 3 macam angka yang dibangkitkan dengan distribusi probabilitas, yaiu distribusi probabilitas uniform, normal dan eksponensial.

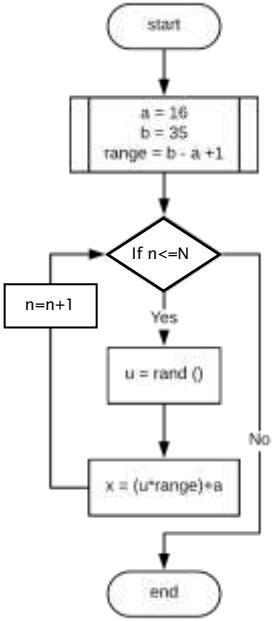
Tabel 3.2 Konfigurasi XBee S2 End Device

AT Command	Function	Value
ATID	Personal Area Network ID	1234
ATSH	Serial Number High	13A200
ATSL	Serial Number Low	40BBE42C
ATDH	Destination Address High	0
ATDL	Destination Address Low	FFFF
ATBH	Broadcast Radius	0
ATBD	Baud Rate	9600

3.4.1 Distribusi Probabilitas Uniform

Untuk membangkitkan suatu angka acak dengan probabilitas uniform, maka dibutuhkan beberapa parameter dari fungsi kepadatan probabilitas uniform. Parameter yang dimaksud menurut fungsi kepadatan probabilitas (2.1) adalah a, yang merupakan batas bawah dari angka yang akan dibangkitkan dan b, yang merupakan batas atas dari angka yang akan dibangkitkan. Gambar 3.7 dan gambar 3.8 menunjukkan flowchart dan kode pemrograman pada MATLAB untuk simulasi distribusi probabilitas uniform.

Pada gambar 3.8, simulasi dilakukan dengan parameter batas bawah atau a dengan nilai 16 dan batas atas atau b dengan nilai 40. Setelah dilakukan inisialisasi nilai batas atas dan batas bawah, hal yang dilakukan selanjutnya adalah membangkitkan angka sesuai dengan distirbusi probabilitas uniform. Langkah pertama adalah

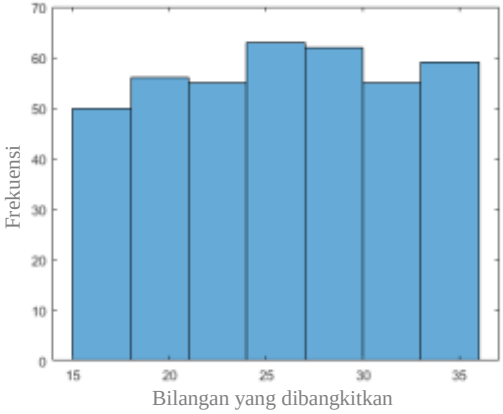


Gambar 3.7 Flowchart pembangkitan angka dengan distribusi probabilitas uniform

```
a=16;
b=35;
u = rand(1,600);
range = b - a + 1;
x = (u*range) + a;
histogram(x)
```

Gambar 3.8 Kode MATLAB untuk simulasi distibusi probabilitas uniform

membangkitkan angka dengan distribusi probabilitas uniform dengan fungsi *rand* yang ada pada MATLAB. Fungsi ini akan membangkitkan angka dari 0 sampai 1. Fungsi *rand* dengan argumen (1,600) mempunyai pengertian bahwa MATLAB akan membangkitkan angka dari 0 sampai 1 sebanyak matrix 1x600. Jadi, 600 angka dibangkitkan dengan distribusi probabilitas uniform. Selanjutnya, hasil angka yang dibangkitkan dikalikan dengan *range*, yaitu selisih antara batas atas dan batas bawah ditambah 1, lalu ditambahkan dengan batas bawah sehingga didapatkan angka yang dibangkitkan secara acak dengan distribusi probabilitas uniform. Gambar 3.9 menunjukkan histogram dari angka yang dibangkitkan. Dari gambar 3.9, dengan melihat histogtam, terbukti bahwa angka yang dibangkitkan sesuai dengan distribusi probabilitas uniform.



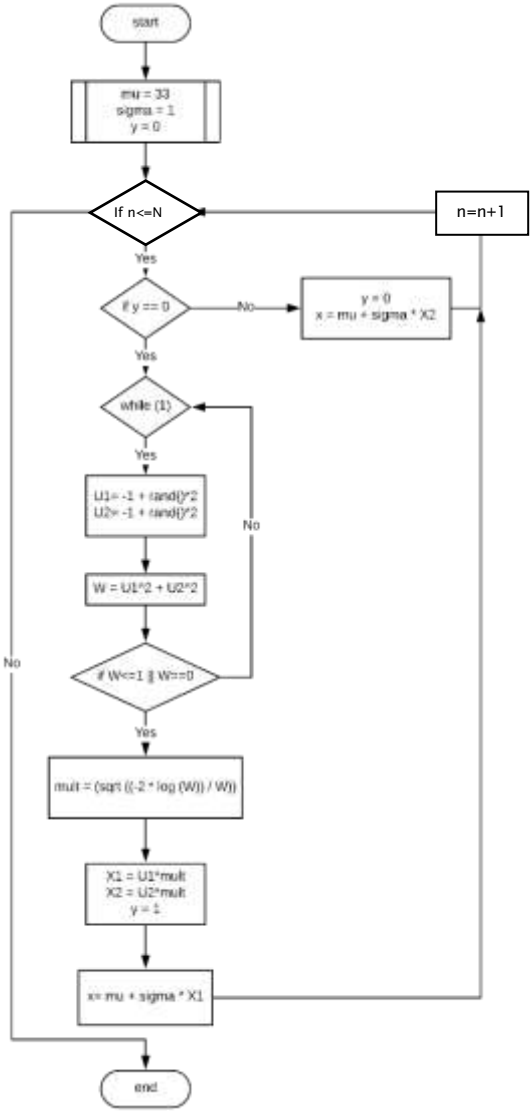
Gambar 3.9 Histogram angka acak yang dibangkitkan dengan distribusi probabilitas uniform

3.4.2 Distribusi Probabilitas Normal

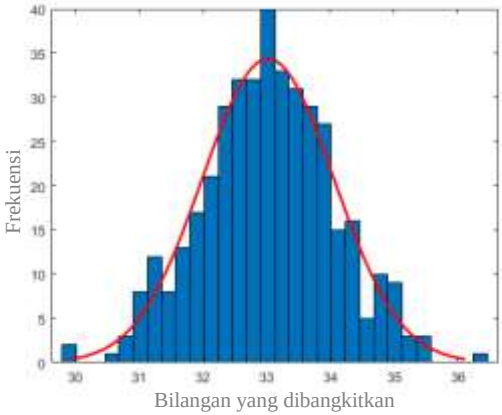
Pada distribusi probabilitas normal, parameter yang dibutuhkan untuk membangkitkan angka secara acak menurut fungsi kepadatan probabilitas distribusi normal (2.2) adalah adalah rata-rata (mean) dan standar deviasi. Rata-rata biasanya disimbolkan dengan simbol μ sedangkan standar deviasi disimbolkan dengan simbol σ. Gambar 3.10 dan gambar 3.11 menunjukkan kode pemrograman pada MATLAB dan flowchart untuk simulasi distribusi probabilitas normal. Pada gambar 3.10, simulasi dilakukan dengan parameter rata-rata atau mu dengan nilai 33 dan sandar deviasi atau sigma dengan nilai 1. Metode yang digunakan unuk membangkitkan angka dengan distribusi probabilitas uniform adalah metode polar Marsaglia.

```
mu=33;
sigma=1;
i=0;
y=0;
for i=1:600
if (y == 0)
    while 1
      U1 = -1 + (rand) * 2;
      U2 = -1 + (rand) * 2;
      W = U1^2 + U2^2;
      if (W <= 1 || W == 0)
          break;
      end
    end
  mult = (sqrt ((-2 * log (W)) / W));
  X1 = U1 * mult;
  X2 = U2 * mult;
  y = ~y;
  x(1,i) = mu + sigma * X1;
end
if (y == 1)
      y = ~y;
      x(1,i) = mu + sigma * X2;
end
end
histfit(x,30,'normal')
```

Gambar 3.10 Kode MATLAB untuk simulasi distibusi probabilitas normal



Gambar 3.11 Flowchart pembangkitan angka dengan distirbusi probabilitas normal



Gambar 3.12 Histogram angka acak yang dibangkitkan dengan distribusi probabilitas normal

Metode polar marsaglia merupakan metode untuk membangkitkan sepasang angka acak yang independen terhadap satu sama lain. Langkah petrama yang dilakukan yaitu membangkitkan sepasang angka acak dengan disribusi probabilitas uniform dari -1 sampai 1. Fungsi W merupakan fungsi luasan berbentuk persegi dari $-1 < U1 < 1$, $-1 < U2 < 1$ dan digunakan unuk memilih sebuah titik acak didaam luasan persegi tersebut. Nilai X1 dan X2 ialah sepasang angka acak dengan nilai mean 0 dan standar deviasi 1. Supaya mempunyai mean dan standar deviasi yang diinginkan, nilai X1 dan X2 dikalikan dengan standar deviasi yang diinginkan lalu ditambah dengan mean yang diinginkan. Gambar 3.12 menunjukkan histogram dari angka yang dibangkikan. Dari gambar 3.12, dengan melihat histogram, terbukti bahwa angka yang dibangkitkan sesuai dengan distribusi probabilitas normal.

3.4.3 Distribusi Probabilitas Eksponensial

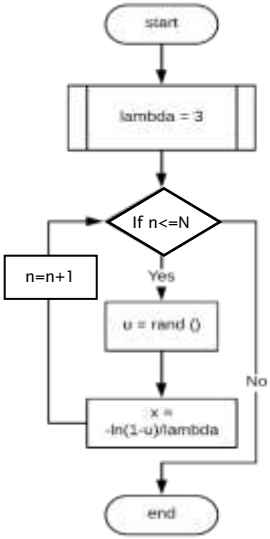
Parameter yang dibutuhkan untuk membangkitkan acak dengan distribusi probabilitas eksponensial adalah parameter rasio. Parameter rasio umumnya disimbolkan dengan simbol λ. Parameter rasio menentukan bagaimana bentuk dari angka yang didistribusikan. Gambar 3.13 dan gambar 3.14 menunjukkan kode pemrograman pada

```
lambda=3;
u = rand(1,600);
x = -log(1- u) / lambda;
histfit(x,30,'exponential')
```

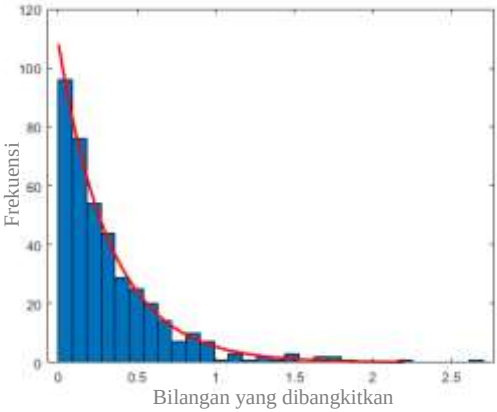
Gambar 3.13 Kode MATLAB untuk simulasi distibusi probabilitas eksponensial

MATLAB dan flowchart untuk simulasi distribusi probabilitas eksponensial.

Pada gambar 3.13, simulasi dilakukan dengan parameter rasio sama dengan 3. Setelah dilakukannya inisialisasi, Langkah selanjutnya adalah membangkitkan angka acak dengan distribusi uniform dari 0 sampai 1 dengan fungsi *rand*. Dari fungsi kepadatan probabilitas eksponensial (2.3), didapakan fungsi invers sehingga angka acak dapat dibangkitkan.



Gambar 3.14 Flowchart pembangkitan angka dengan distirbusi probabilitas eksponensial



Gambar 3.15 Histogram angka acak yang dibangkitkan dengan distribusi probabilitas eksponensial

Gambar 3.15 menunjukkan histogram dari angka yang dibangkikan dengan disribusi probabilitas ekspoensial. Dari gambar 3.15, dengan melihat histogram, terbukti bahwa angka yang dibangkitkan sesuai dengan distribusi probabilitas normal.

3.5 Simulasi Algoritma Kompresi pada MATLAB

Setelah angka acak dibangkitkan, maka angka acak yang dibangkitkan akan dikompresi. Sebelum dilakukan implementasi pada Arduino UNO, angka acak yang sudah dibangkitkan akan melakukan simulasi mengenai kompresi terlebih dahulu dengan menggunakan MATLAB supaya lebih mudah saat penulisan kode pemrograman dalam bahasa C pada perangkat lunak Arduino IDE. Simulasi kompresi pada MATLAB akan mensimulasikan dua macam algoritma kompresi, yaiu Lossless entropy compression (LEC) dan Sequential lossless entropy compression (S-LEC).

3.5.1 Simulasi Algoritma Kompresi LEC

Sebelum dilakukannya kompresi, inisialisasi perlu dilakukan. inisialisasi tersebut adalah nilai atau angka yang akan dikompresi, jumah angka yang akan dikompresi dan variabel lain yang diperlukan. Gambar 3.16 menunjukkan fungsi inisialisasi sebelum dilakukannya kompresi. Pada gambar 3.16, angka yang dibangkitkan berdasarkan distribusi uniform diantara angka 16 sampai 35 dengan jumlah 600.

```
ri = 16 + ((35-16)*rand(1000,1));
ri=round(ri'*100);
sri=size(ri);
hi=[];
comp_size_EC=0;
```

Gambar 3.16 Inisialisasi sebelum dilakukan kompresi

Setelah inisialisasi dilakukan, maka tahap yang dilakukan selanjutnya adalah menentukan grup dari data angka yang dikompresi dan membenuk kode dari grup tersebut, sesuai dengan tabel (2.2). Gambar 3.17 menunjukkan fungsi penentuan grup dan pembentukan kode grup. Pengulangan pada gambar 3.17 berfungsi untuk melakukan kompresi pada angka yang dibangkitkan sampai jumah angka yang akan dikompresi atau sesuai dengan ukuran mariks.

Tahap terakhir dari algoritma kompresi LEC adalah penentuan dan pembentukan kode indeks berdasarkan dari angka yang dikompresi, sesuai dengan (2.5). Gambar 3.18 menunjukkan fungsi penentuan dan pembentukan kode indeks. Jika angka yang dikompresi bernilai 0, maka deret bit-nya mempunyai kode grup "00" dan tidak mempunyai kode indeks. Selain angka yang bernilai 0, maka penentuan indeksnya sesuai dengan (2.5) dan dikonversi ke dalam bentuk deret bit.

```
for R=1:sri(1,2)
ni= ceil(log2(abs(ri(1,R))));
if ni>4
    hi=[ones(1,(ni-3)) 0];
     else
         switch ni
        case 1
            hi=[0 1 0];
        case 2
            hi=[0 1 1];
        case 3
            hi=[1 0 0];
        case 4
            hi=[1 0 1];
    end
end
```

Gambar 3.17 Fungsi penentuan dan pembentukan kode grup

```
if ri(1,R)==0
    bitstream{1,R}=[0 0];
    else if ri(1,R)>0
        ai(1,R)=ri(1,R);
        ai=de2bi(ai(1,R),'left-msb');
        bitstream{1,R}=[hi ai];
        else if ri(1,R)<0
            ai(1,R)=abs(ri(1,R));
            ai=de2bi(ai(1,R),'left-msb');
            bitstream{1,R}=[hi ~ai];
            end
        end
end
end
```

Gambar 3.18 Fungsi penentuan dan pembentukan kode indeks

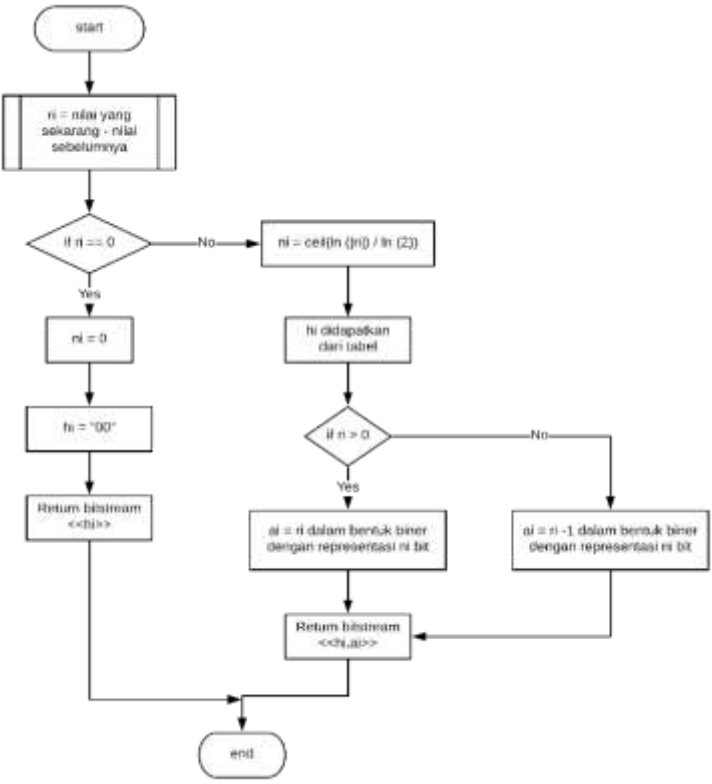
Untuk mengetahui besar ukuran dari seluruh angka yang dikompresi, maka pada gambar 3.16 ditunjukkan fungsi untuk menghitung besar ukuran hasil kompresi dari seluruh angka yang dibangkitkan. Algoritma kompresi data lossless entropy coding secara keseluruhan dapat dilihat dalam flowchart pada gambar 3.20.

3.5.2 Simulasi Algoritma Kompresi Sequential LEC

Simulasi algoritma kompresi sekuensial LEC mempunyai kode dan fungsi pemrograman MATLAB yang hampir sama dengan simulasi algoritma kompresi LEC. Dalam kompresi sekuensial LEC, deret angka pertama yang akan dikompresi harus menggunakan metode kompresi LEC. Berikutnya, setelah angka pertama berhasil dikompresi, angka selanjunya dikompresi menggunakan algoritma sekuensial LEC.

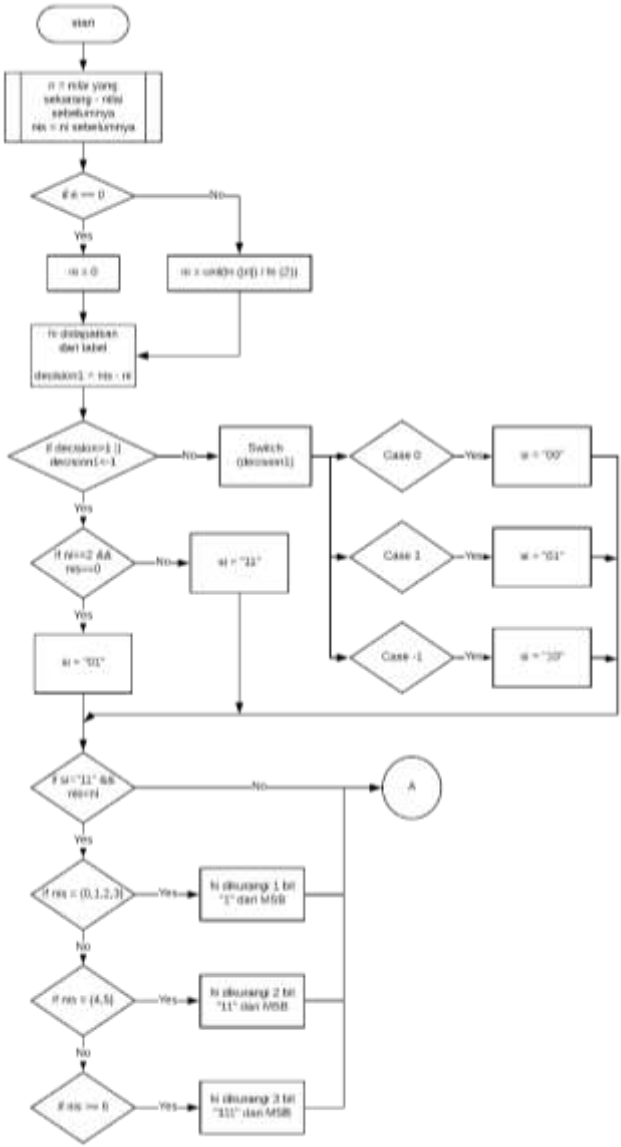
```
for R=1:sri(1,2)
comp_size_EC=comp_size_EC+size(bitstream{1,R
});
end
comp_size_EC(1,2)
```

Gambar 3.19 Fungsi penghitung besar ukuran dari seluruh hasil kompresi



Gambar 3.20 Flowchart algoritma kompresi Lossless Entropy Coding

Perbedaan yang mendasar antara kompresi LEC dan sekuensial LEC adalah adanya kode sekuensial yang dapat menghilangkan kode grup agar *codeword* menjadi lebih ringkas dan ketika kode grup tidak dapat dihilangkan, satu sampai tiga bit dari kode grup dapat dihilangkan. Jadi terdapat dua fungsi tambahan pada simulasi kompresi sekuensial LEC. Gambar 3.23 menunjukkan fungsi untuk membentuk kode sekuensial.



Gambar 3.21 Flowchart algoritma kompresi Sequential Lossless Entropy Coding (Bagian 1)

Gambar 3.22 Flowchart algoritma kompresi Sequential Lossless Entropy Coding (Bagian 2)

```
if ni(1,R-1)==ni(1,R)
    si=[0 0];
    hi=[];
elseif ni(1,R-1)-ni(1,R)==1 || (ni(1,R)==2
&& ni(1,R-1)==0)
        si=[0 1];
        hi=[];
elseif ni(1,R-1)-ni(1,R)==-1
            si=[1 0];
            hi=[];
        else
            si=[1 1];
end
```

Gambar 3.23 Fungsi untuk membentuk kode sekuensial

```
if si==[1 1]
    if (ni(1,R)>ni(1,R-1))
        if (ni(1,R-1)>=0) && (ni(1,R-1)<=3)
            if ni(1,R)==2
                hi(2)=[];
            else
                hi(1)=[];
            end
        elseif (ni(1,R-1)>=4) && (ni(1,R-
1)<=5)
            hi=hi([3:end]);
        elseif (ni(1,R-1)>=6)
            hi=hi([4:end]);
        end
    end
end
```

Gambar 3.24 Fungsi untuk mereduksi kode grup h_i

Fungsi pembentukan kode sekuensial yang digunakan pada kompresi sekuensial LEC berdasar pada tabel (2.2). Jika kode sekuensial yang dibentuk adalah "11", maka kode grup h_i tidak bisa dihilangkan. Namun jika kode sekuensial yang dibentuk adaah selain "11", maka kode grup h_i dapat dihilangkan. Ketika kode grup h_i tidak dapat dihilangkan kode grup h_i dapat direduksi. Gambar 3.24 menunjukkan fungsi yang digunakan untuk mereduksi kode grup h_i. Algoritma kompresi data sequential lossless entropy coding secara keseluruhan dapat dilihat dalam flowchart pada gambar 3.21 dan 3.22.

3.6 Implementasi pada Arduino

Implementasi algoritma kompresi pada arduino mempunyai kode pemrograman yang hampir mirip dengan MATLAB. Pada penulisan kode pemrograman Arduino, terdapat tambahan fungsi, misal fungsi yang digunakan untuk mengirim hasil kompresi. Oleh karena itu, pada penulisan kode pemrograman arduino ini diharuskan merubah dari bahasa pemrograman MATLAB menjadi bahasa pemrograman arduino.

3.6.1 Implementasi Distribusi Probabilitas

Implementasi pada bahasa pemrograman Arduino untuk membangkitkan angka acak berdasar pada simulasi MATLAB yang sudah dibahas pada subbab sebelumnya. Untuk membangkitkan

angka acak pada bahasa pemrograman arduino, algoritma untuk membangkikannya mempunyai algoritma yang sama namun hanya penulisan bahasanya yang sedikit berbeda. Gambar 3.25, 3.26 2.27 secara berturut-turut menunjukkan 3 fungsi yang digunakan untuk membangkitkan angka acak dengan 3 disribusi probabilitas, yaitu distribusi uniform, distribusi normal, dan distribusi eksponensial.

3.6.2 Implementasi Algoritma Kompresi LEC

Dalam kompresi LEC ini, bahasa pemrograman pada MATLAB kurang lebih sama dengan bahasa pemrograman arduino. Gambar 3.28 menunjukkan fungsi yang digunakan untuk melakukan kompresi metode LEC dalam bahasa pemrograman arduino. Dalam fungsi tersebut, terdapat fungsi bernama dec2bin. Fungsi ini digunakan unuk mengubah bilangan dengan basis desimal menjadi bilangan dengan basis biner. Fungsi dec2bin ditunjukkan pada gambar

```
double rand_uni (int rangeLow, int
rangeHigh) {
    double myRand = rand ()/ (1.0 +
RAND_MAX);
    int range = rangeHigh - rangeLow + 1;
    double myRand_scaled = (myRand * range)
+ rangeLow;
    return myRand_scaled;
}
```

Gambar 3.25 Fungsi untuk membangkitkan angka acak dengan distribusi uniform

```
double ran_norm(double mu, double sigma){
  double U1, U2, W, mult;
  static double X1, X2;
  static int call = 0;

  if (call == 1)
    {
      call =! call;
      return (mu + sigma*(double) X2);
    }

  do
    {
      U1=-1+((double)rand()/RAND_MAX) *2;
```

```
      U2=-1+((double)rand()/RAND_MAX) *2;
      W = pow (U1, 2) + pow (U2, 2);
    }
  while (W >= 1 || W == 0);

  mult = sqrt ((-2 * log (W)) / W);
  X1 = U1 * mult;
  X2 = U2 * mult;
  call =! call;
  return (mu + sigma * (double) X1);
}
```

Gambar 3.26 Fungsi untuk membangkitkan angka acak dengan distribusi normal

3.29. Fungsi ini juga harus dimasukkan kedalam kode pemrograman, berbeda dengan MATLAB yang sudah mempunyai library-nya. Kode pemrograman arduino dengan metode kompresi LEC secara menyeluruh dapat diihat dalam bagian lampiran.

```
double ran_expo (double lambda) {
    double u;
    u = rand () / (RAND_MAX + 1.0);
    return (-log (1- u) / lambda);
}
```

Gambar 3.27 Fungsi untuk membangkitkan angka acak dengan distribusi eksponensial

```
String LEC (int ri_suhu) {
  String bitstream, ai;
  int base=2;
  if (ri_suhu==0) {
    ni=0;
    bitstream="00";
  }
  else
    ni=0;
ni=ceil(log(abs(ri_suhu))/log(base)+0.001);
    ai=dec2bin(ri_suhu);
    bitstream=hi[ni]+ai;
  return bitstream;
}
```

Gambar 3.28 Fungsi kompresi LEC

```
String dec2bin(int n) {
  int rem;
  String coba="";
  if (n>0) {
    while (n>=1) {
      rem=n%2;
      n/=2;
      if (rem==1) {
        coba="1"+coba;
      }
      else {
        coba="0"+coba;
      }
    }
  }
  else {
    n=abs(n);
    while (n>=1) {
      rem=n%2;
      n/=2;
      if (rem==1) {
        coba="0"+coba;
      }
      else {
        coba="1"+coba;
      }
    }
  }
return coba;
}
```

Gambar 3.29 Fungsi dec2bin

3.6.3 Implementasi Algoritma Kompresi Sequential LEC

Fungsi sekuensial LEC yang ditunjukkan pada gambar 3.30, pada bahasa pemrograman arduino, sedikit berbeda dengan bahasa pemrograman MATLAB. Variabel ri_suhu dalam kode pemrograman merupakan input dari fungsi tersebut. Variabel ri_suhu merupakan angka acak yang dibangkitkan atau data angka yang diambil dari sensor suhu. Inisialisasi dalam fungsi tersebut diperlukan seperi nis yang artinya kode grup sebelumnya, dan juga variabel decision1 yang digunakan untuk menentukan kode sekuensial. Output dari fungsi

kompresi ini merupakan string yang berisikan deretan bit. Output dari fungsi ini juga bergantung dari kode sekuensial yang digunakan untuk menentukan apakah codeword dari kode grup dimasukkan dalam deretan bit apa tidak. Kode pemrograman arduino dengan metode kompresi sekuensial LEC secara menyeluruh dapat diihat dalam bagian lampiran.

```
String S_LEC (int ri_suhu) {

  int nis, decision1;
  String si[4]={"00","01","10","11"};
  String bitstream,si_tx,hi_tx,ai;

  nis=ni;
  ni=0;
 ni=ceil(log(abs(ri_suhu))/log(base)+0.001);
  decision1=nis-ni;
  if (decision1>1 || decision1<-1) {
    if (ni==2 && nis==0) {
      si_tx=si[1];
    }
    else {
    si_tx=si[3];
    }
  }
  else {
    switch (decision1) {
    case 0 :
    si_tx=si[0];
    break;
    case 1 :
    si_tx=si[1];
    break;
    case -1 :
    si_tx=si[2];
    break;
    }
  }
  hi_tx=hi[ni];
  if (si_tx=="11" && ni>nis) {
    if (nis>=0 && nis<=3) {
        hi_tx.remove(0,1);
```

```
    }
    else {
      if (nis==4 || nis==5) {
        hi_tx.remove(0,2);
      }
      else {
        if (nis>=6) {
          hi_tx.remove(0,3);
        }
      }
    }
  }
    if (ri_suhu==0) {
      ai="";
    }
    if (si_tx=="11") {
      ai=dec2bin(ri_suhu);
      bitstream=si_tx+hi_tx+ai;
    }
    else {
      ai=dec2bin(ri_suhu);
      bitstream=si_tx+ai;
    }
  return bitstream;
}
```

Gambar 3.30 Fungsi kompresi Sekuensial LEC

3.6.4 Metode Pengiriman Hasil Kompresi

Setelah data angka berhasil dikompresi, langkah selanjutnya adalah pengiriman deretan bit hasi kompresi. Pengiriman deretan bit menggunakan metode komunikasi serial melalui zigbee. Gambar 3.31 dan gambar 3.32 menunjukkan flowchart dan kode pemrograman untuk mengirimkan hasil kompresi.

Pada gambar 3.32, kode pemrograman menjelaskan bahwa satu sampel data pertama kali dikirimkan lalu setelahnya baru dikirimkan hasil kompresi. Setelah data pertama berhasil dikirim, data selanjutnya diambil setiap detik lalu terlebih dahulu dikumpulkan sebelum dikirimkan sampai terkumpul 40 data angka. Lalu angka satu per satu dikompresi lalu hasil kompresi tersebut satu per satu dikumpulkan dan akhirnya menjadi deretan bit. Dari deretan bit tersebut, setiap 8 bit atau 1 byte, yang awalnya berupa string, diubah

Gambar 3.31 Flowchart algoritma pengiriman hasil kompresi

menjadi bilangan integer untuk selanjutnya dikirimkan melalui komunikasi serial. Sebelum bilangan integer dikirimkan, untuk memulai komunikasi antara dua arduino, *start symbol* dan *stop symbol* dibutuhkan. Dalam gambar 3.28 dan 3.29 *start symbol* dan *stop symbol* yang digunakan adaah < dan > yang per simbolnya berukuran 1 byte.

```
void loop() {
  int suhu_skrg, suhu_sblm, suhu_awal = 0,
chk;
  int  i, j, lop;
  double inttx;
  String kuy;
  delay(10000);
```

```
  chk = DHT.read11(iPinDHT11);
  suhu_skrg = DHT.humidity;
  Serial.write('<');
  Serial.print((int)suhu_skrg);
  Serial.write('>');
  while (1) {
    tx = "";
    for (j = 0; j < 40; j++) {
      chk = DHT.read11(iPinDHT11);
      delay(1000);
      suhu_sblm = suhu_skrg;
      suhu_skrg = DHT.humidity;
ri_suhu = (int)suhu_skrg - (int)suhu_sblm;
      tx = tx + LEC((int)ri_suhu);
    }
    tx = "1" + tx;
lop = tx.length();
lop = lop % 8 if (lop != 0) {
      for (j = 0; j < 8 - lop; j++) {
        tx = "0" + tx;
    }
}
    Serial.write('<');  //Starting symbol
    for (j = 0; j < tx.length() / 8; j++) {
      kuy = (String)tx[j * 8] + (String)tx[j
* 8 + 1] + (String)tx[j * 8 + 2] +
(String)tx[j * 8 + 3] + (String)tx[j * 8 +
4] + (String)tx[j * 8 + 5] + (String)tx[j *
8 + 6] + (String)tx[j * 8 + 7];
      inttx = bi2de(kuy);
      if ((int)inttx == 60) {
        Serial.write('0');
        Serial.write('0');
      }
      else if ((int)inttx == 62) {
        Serial.write('1');
        Serial.write('1');
      }
      else
        Serial.write((int)inttx);
    }
    Serial.write('>');//Ending symbol
```

```
    }
}
```

Gambar 3.32 Kode pemrograman pengiriman

Halaman ini sengaja dikosongkan

BAB IV

PEMBAHASAN

Pada bab ini, akan dibahas mengenai hasil data yang diperoleh dari hasil simulasi maupun dari hasil pengukuran dari sistem yang telah didesain. Alat hasil implementasi kompresi data pada jaringan sensor nirkabel diuji dengan 3 algoritma yaitu tanpa kompresi, lossless entropy coding (LEC), dan sequential lossless entropy coding (S-LEC). Data yang dikompresi juga diambil dari beberapa cara. 5 cara pengambilan data yang akan dikompresi antara lain, dari sensor suhu, sensor kelembaban, juga data yang dibangkitkan dengan distribusi probabilitas uniform, normal, dan eksponensial. Pengukuran dilakukan dengan menggunakan alat probe tegangan, *clamp* meter, dan osilosokop untuk mengukur tegangan dan arus sehingga daya dan energi didapatkan. Setiap pengukuran dilakukan dengan durasi kurang lebih 10 menit. Pada pengukuran, daya yang dipancarkan oleh modul radio tiap byte mempunyai daya yang sama. Diasumsikan juga bahwa jarak antara pengirim dan penerima tidak berpengaruh pada pengukuran sehingga kesalahan bit dalam pengiriman dapat diabaikan.

4.1 Hasil Simulasi

Simulasi yang dilakukan pada MATLAB mensimulasikan dengan menggunakan 3 algoritma yaitu tanpa kompresi, lossless entropy coding (LEC), dan sequential lossless entropy coding (S-LEC). Data suhu yang disimulasikan dan yang akan dibangkitkan menggunakan 3 distribusi probabilitas yaitu distribusi uniform dengan batas bawah 16 dan batas atas 35, distribusi normal dengan rata-rata 33 dan standar deviasi bernilai 1, dan distribusi ekponensial dengan rasio 0,1. Tabel 4.1 menunjukkan hasil simulasi dalam jumlah bit.

Tabel 4.1 Jumlah bit dari hasil simulasi

Jumlah bit	Tanpa Kompresi	LEC	S-LEC
Uniform	4800	3535	3404
Normal	4800	2267	1903
Eksponensial	4800	3725	3714

Hasil simulasi menunjukkan bahwa sequential lossless entropy coding (S-LEC) merupakan algoritma yang lebih baik dari lossless entropy coding (LEC). Kompresi S-LEC mempunyai jumlah bit yang lebih sedikit dari LEC apapun jenis distribusi probabilitasnya.

4.2 Ujicoba Algoritma Kompresi Data

Sebelum dilakukan pengukuran arus dan tegangan pada arduino, algoritma kompresi data dilakukan ujicoba terlebih dahulu. Pengujian dilakukan dengan dilakukan input terhadap fungsi lossless entropy coding (LEC) dan sequential lossless entropy coding (S-LEC). Input dari 2 fungsi algortima kompresi data tersebut adalah bilangan integer dengan outputnya adalah deretan bit dalam bentuk string. Bilangan yang diinputkan yaitu 0, 1, -2, 2, -1, 5, 0, -8, 1, -1. Setelah bilangan tersebut diinputkan, output dari 2 fungsi algoritma tersebut dapat dilihat dalam serial monitor didalam software Arduino IDE. Gambar 4.1 dan gambar 4.2 berturut-turut menunjukkan output ujicoba dari kompresi lossless entropy coding dan sequential lossless entropy coding.

4.3 Hasil Pengukuran

Pada sub bab ini, akan disajikan hasil data pengukuran energi di node pengirim dan data yang diterima oleh coordinator atau penerima baik yang diambil dari sensor atau data yang dibangkitkan dengan probabilitas tertentu. Lalu hasil pengukuran tiap metode kompresi dibandingkan.

Gambar 4.1 Output ujicoba dari kompresi lossless entropy coding

Gambar 4.2 Output ujicoba dari kompresi sequential lossless entropy coding

4.3.1 Perbandingan Kompresi Data Riil Menggunakan Sensor Suhu

Pada pengirimin dengan data yang kirimkan adalah suhu, Kompresi LEC memiliki daya rata-rata yang lebih kecil daripada kompresi S-LEC dan tanpa kompresi. Dari penerima, dapat dilihat bahwa data yang dikirimkan dengan metode tanpa kompresi adalah 520 byte. Sedangkan data yang dikirimkan dengan metode kompresi LEC dan S-LEC mempunyai nilai yang sama, yaitu sebesar 156 byte. Dari data yang diterima oleh penerima, jika dilakukan dekompresi secara manual, maka hasil dekompresi yang merupakan bilangan integer yang terdistribusi uniform. Jumlah energi yang dikonsumsi dengan metode tanpa kompresi adalah 99,72 J. Sedangkan jumlah energi yang dikonsumsi dengan metode kompresi LEC dan S-LEC,

Tabel 4.2 Statistik penggunaan daya per data suhu

Daya (W)	Tanpa Kompresi	LEC	S-LEC
Rata - rata	0.166192472	0.11564496	0.139769573
Jangkauan	0.2736	0.2508	0.2552
Varians	0.000980305	0.000926699	0.000985496
Standar Deviasi	0.031309829	0.030441731	0.031392617

berturut-turut sebesar 69,39 J dan 83,86 J. Tabel 4.2 menunjukkan statistik penggunaan daya per data suhu.

4.3.2 Perbandingan Kompresi Data Riil Menggunakan Sensor Kelembaban

Pada pengirimin dengan data yang kirimkan adalah kelembaban, Kompresi S-LEC memiliki daya rata-rata yang lebih kecil daripada kompresi LEC dan tanpa kompresi. Dari penerima, dapat dilihat bahwa data yang dikirimkan dengan metode tanpa kompresi adalah 520 byte. Sedangkan data yang dikirimkan dengan metode kompresi LEC dan S-LEC, berrturut-turut mempunyai nilai sebesar 157 byte dan 156 byte. Dari data yang diterima oleh penerima, jika dilakukan dekompresi secara manual, maka hasil dekompresi yang merupakan bilangan integer yang terdistribusi uniform. Jumlah energi yang dikonsumsi dengan metode tanpa kompresi adalah 114,22 J. Sedangkan jumlah energi yang dikonsumsi dengan metode kompresi LEC dan S-LEC, berturut-turut sebesar 73,83 J dan 65,82 J. Tabel 4.3 menunjukkan statistik penggunaan daya per data kelembaban.

4.3.3 Perbandingan Kompresi Data yang Dibangkitkan Sendiri dengan Distribusi Uniform

Pada pengirimin dengan data yang kirimkan adalah angka yang dibangkitkan dengan distribusi uniform, Kompresi S-LEC memiliki daya rata-rata yang lebih kecil daripada kompresi LEC dan tanpa kompresi. Dari penerima, dapat dilihat bahwa data yang dikirimkan dengan metode tanpa kompresi adalah 520 byte.

Tabel 4.3 Statistik penggunaan daya per data kelembaban

Daya (W)	Tanpa Kompresi	LEC	S-LEC
Rata - rata	0.190359633	0.123043632	0.109698621
Jangkauan	0.3024	0.2508	0.2508
Varians	0.00093101	0.000848739	0.000908261
Standar Deviasi	0.030512454	0.029133119	0.030137376

Tabel 4.4 Statistik penggunaan daya per distribusi uniform

Daya (W)	Tanpa Kompresi	LEC	S-LEC
Rata - rata	0.117168926	0.118253641	0.106533459
Jangkauan	0.2736	0.2508	0.228
Varians	0.00110446	0.000899881	0.000880471
Standar Deviasi	0.033233415	0.029998014	0.029672732

Sedangkan data yang dikirimkan dengan metode kompresi LEC dan S-LEC, berrturut-turut mempunyai nilai sebesar 520 byte dan 415 byte. Jumlah energi yang dikonsumsi dengan metode tanpa kompresi adalah 70,3 J. Sedangkan jumlah energi yang dikonsumsi dengan metode kompresi LEC dan S-LEC, berturut-turut sebesar 70,95 J dan 63,92 J. Tabel 4.4 menunjukkan statistik penggunaan daya per data dengan distribusi uniform.

4.3.4 Perbandingan Kompresi Data yang Dibangkitkan Sendiri dengan Distribusi Normal

Pada pengirimin dengan data yang kirimkan adalah angka yang dibangkitkan dengan distribusi nnormal, Kompresi S-LEC memiliki daya rata-rata yang lebih kecil daripada kompresi LEC dan tanpa kompresi. Dari penerima, dapat dilihat bahwa data yang dikirimkan dengan metode tanpa kompresi adalah 520 byte. Sedangkan data yang dikirimkan dengan metode kompresi LEC dan S-LEC, berrturut-turut

Tabel 4.5 Statistik penggunaan daya per distribusi normal

Daya (W)	Tanpa Kompresi	LEC	S-LEC
Rata - rata	0.135195582	0.11703539	0.10865327
Jangkauan	0.2784	0.232	0.2736
Varians	0.000961417	0.000932164	0.000872248
Standar Deviasi	0.031006725	0.030531355	0.029533848

mempunyai nilai sebesar 278 byte dan 225 byte. Jumlah energi yang dikonsumsi dengan metode tanpa kompresi adalah 81,11 J. Sedangkan jumlah energi yang dikonsumsi dengan metode kompresi LEC dan S-LEC, berturut-turut sebesar 70,22 J dan 65,19 J. Tabel 4.5 menunjukkan statistik penggunaan daya per data dengan distribusi normal.

4.3.5 Perbandingan Kompresi Data yang Dibangkitkan Sendiri dengan Distribusi Eksponensial

Pada pengirimin dengan data yang kirimkan adalah angka yang dibangkitkan dengan distribusi eksponensial, Kompresi S-LEC memiliki daya rata-rata yang lebih kecil daripada kompresi LEC dan tanpa kompresi. Dari penerima, dapat dilihat bahwa data yang dikirimkan dengan metode tanpa kompresi adalah 520 byte. Sedangkan data yang dikirimkan dengan metode kompresi LEC dan S-LEC, berrturut-turut mempunyai nilai sebesar 447 byte dan 437 byte. Jumlah energi yang dikonsumsi dengan metode tanpa kompresi adalah 83,12 J. Sedangkan jumlah energi yang dikonsumsi dengan metode kompresi LEC dan S-LEC, berturut-turut sebesar 79,5 J dan 77,97 J. Tabel 4.6 menunjukkan statistik penggunaan daya per data dengan distribusi eksponensial.

4.4 Efisiensi Daya

Sub bab ini bertujuan untuk membandingkan semua hasil pengukuran dari jenis scenario data yang berbeda-beda dengan efisiensi. Daya yang digunakan untuk menghitung efisiensi adalah daya rata-rata. Berikut merupakan rumus yang digunakan untuk

Tabel 4.6 Statistik penggunaan daya per distribusi eksponensial

Daya (W)	Tanpa Kompresi	LEC	S-LEC
Rata - rata	0.138534607	0.13250184	0.129946625
Jangkauan	0.2324	0.2784	0.2736
Varians	0.000956819	0.000962302	0.000959242
Standar Deviasi	0.030932488	0.031020985	0.030971634

menghitung efisiensi daya.

$$Efisiensi = 1 - \frac{Daya\ rata - rata\ dengan\ kompresi}{Daya\ rata - rata\ tanpa\ kompresi}\ x\ 100\%$$

Dari tabel 4.7, terlihat bahwa efisiensi daya dalam distribusi uniform dengan kompresi LEC dalam kasus ini menjadi tidak efisien karena energi yang dikonsumsi dengan metode kompresi data LEC menjadi lebih banyak daripada tanpa kompresi. Metode kompresi data S-LEC umumnya mempunyai efisiensi daya yang lebih baik daripada menggunakan metode kompresi LEC. Tabel 4.7 menunjukkan efisiensi daya dari kompresi LEC dan S-LEC.

Dari tabel 4.7, selanjutnya dilakukan uji statistik yang diberikan pada tabel 4.8. Uji statistik ini bertujuan untuk membandingkan 2 algoritma kompresi data LEC dan S-LEC. Dari uji statistik yang dilakukan, didapatkan bahwa metode algoritma kompresi data sequential lossless entropy coding dengan rata-rata 18,63% daripada algoritma kompresi data lossless entropy coding dengan rata-rata 16,52%.

Tabel 4.7 Efisiensi daya dari kompresi LEC dan S-LEC

Efisiensi Daya (%)	LEC	S-LEC
Suhu	30.41504327	15.89897465
Kelembaban	35.36253995	42.37296033
Distribusi Uniform	-0.925770495	9.077036775
Distribusi Normal	13.43253343	19.632529
Distribusi Eksponensial	4.354700499	6.199159898

Tabel 4.8 Uji statistik efisiensi daya

Lossless Entropy Coding		Sequential Lossless Entropy Coding	
Mean	16.52781	**Mean**	18.63613
Standard Error	7.106318	**Standard Error**	6.395219
Standard Deviation	15.89021	**Standard Deviation**	14.30014
Sample Variance	252.4988	**Sample Variance**	204.4941
Kurtosis	-2.55172	**Kurtosis**	2.455866
Skewness	0.229446	**Skewness**	1.504278
Range	36.28831	**Range**	36.1738
Minimum	-0.92577	**Minimum**	6.19916
Maximum	35.36254	**Maximum**	42.37296
Sum	82.63905	**Sum**	93.18066

BAB V
PENUTUP

5.1 Kesimpulan

Dari hasil penelitian pada tugas akhir ini, dapat disimpulkan hal-hal sebagai berikut:

1. Rancangan jaringan dalam tugas akhir ini menggunakan jaringan point-to-point. Sensor mengambil data baik suhu maupun kelembaban lalu data tersebut dimasukkan ke dalam mikrokontroler Arduino. Di dalam mikrokontroler arduino, data tersebut diolah dan dikompresi. Setelah data berhasil dikompresi, hasil kompresi diteruskan ke zigbee supaya dapat dikirimkan.
2. Dari beberapa komponen-komponen yang digunakan, ada komponen-komponen yang perlu dikonfigurasi menggunakan perangkat lunak. Konfigurasi yang diperlukan adalah pengaturan pengirim dan penerima, penulisan kode pemrograman untuk implementasi algoritma.
3. Pada data riil dengan sensor suhu, algoritma kompresi data *lossless entropy coding* (LEC) dan *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 30% dari byte data aslinya.
4. Pada data riil dengan sensor kelembaban, algoritma kompresi data *lossless entropy coding* (LEC) dan *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 30% dari byte data aslinya. Sedangkan algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 30,2% dari byte data aslinya.
5. Pada data yang dibangkitkan sendiri dengan distribusi uniform, algoritma kompresi data *lossless entropy coding* (LEC) dan *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 100% dari byte data aslinya. Sedangkan algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 79,8% dari byte data aslinya.
6. Pada data yang dibangkitkan sendiri dengan distribusi normal, algoritma kompresi data *lossless entropy coding* (LEC) dan *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 53,46% dari byte data aslinya. Sedangkan algoritma kompresi data *sequential lossless entropy*

coding (S-LEC) mampu mengurangi byte data yang dikirim menjadi 43,27% dari byte data aslinya.

7. Pada data yang dibangkitkan sendiri dengan distribusi eksponensial, algoritma kompresi data *lossless entropy coding* (LEC) dan *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 85,96% dari byte data aslinya. Sedangkan algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mampu mengurangi byte data yang dikirim menjadi 84,04% dari byte data aslinya.
8. Pada data riil dengan sensor suhu, algoritma kompresi data *lossless entropy coding* (LEC) mempunyai efisiensi daya yang lebih baik, yaitu sebesar 30,41%, daripada algoritma kompresi data *sequential lossless entropy coding* (S-LEC), yaitu sebesar 15,89%.
9. Pada data riil dengan sensor kelembaban, algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mempunyai efisiensi daya yang lebih baik, yaitu sebesar 42,37%, daripada algoritma kompresi data *lossless entropy coding* (LEC), yaitu sebesar 35,36%.
10. Pada data yang dibangkitkan sendiri dengan distribusi uniform, algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mempunyai efisiensi daya yang lebih baik, yaitu sebesar 42,37%, daripada algoritma kompresi data *lossless entropy coding* (LEC), yaitu sebesar 35,36%.
11. Pada data yang dibangkitkan sendiri dengan distribusi normal, algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mempunyai efisiensi daya yang lebih baik, yaitu sebesar 19,63%, daripada algoritma kompresi data *lossless entropy coding* (LEC), yaitu sebesar 13,43%.
12. Pada data yang dibangkitkan sendiri dengan distribusi eksponensial, algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mempunyai efisiensi daya yang lebih baik, yaitu sebesar 6,19%, daripada algoritma kompresi data *lossless entropy coding* (LEC), yaitu sebesar 4,35%.
13. Algoritma kompresi data *lossless entropy coding* (LEC) mempunyai efisiensi paling baik pada data riil dengan sensor kelembaban. Sedangkan algoritma kompresi data *lossless entropy coding* (LEC) mempunyai efisiensi paling buruk pada data yang dibangkitkan sendiri dengan distribusi uniform.
14. Algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mempunyai efisiensi paling baik pada data riil dengan

sensor kelembaban. Sedangkan algoritma kompresi data *sequential lossless entropy coding* (S-LEC) mempunyai efisiensi paling buruk pada data yang dibangkitkan sendiri yang distribusi eksponensial.

15. Dari uji statistik, algoritma kompresi data *lossless entropy coding* (LEC) dapat menghemat energi yang dikonsumsi sebesar 16,52%.
16. Dari uji statistik, algoritma kompresi data *sequential lossless entropy coding* (S-LEC) dapat menghemat energi yang dikonsumsi sebesar 18,63%.
17. Dari uji statistik, algoritma kompresi data s*equential lossless entropy coding (S-LEC)* lebih baik dalam menghemat energi yang dikonsumsi daripada algoritma kompresi data *lossless entropy coding (LEC).*

5.2 Saran

1. Dengan banyaknya keterbatasan perangkat, algoritma kompresi dapat diimplementasikan ke dalam perangkat lainnya, seperti Raspberry Pi.
2. Diperlukan waktu yang presisi pada pengukuran tegangan dan arus pada osiloskop sehingga data pengukuran yang didapatkan dapat dibandingkan.
3. Algoritma dekompresi belum diimplementasikan ke dalam Arduino dan masih dilakukan dekompresi manual sehingga kedepannya dapat dilakukan pengimplementasian algoritma dekompresi pada Arduino.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] K. Holger and W. Andreas, Protocols and Architectures for Wireless Sensor Networks, Chicester: John Wiley & Sons, Ltd, 2005.

[2] J. Yan, Marchinery Prognostics and Prognosis Oriented Maintenance Management, John & Wiley Sons, Ltd, 2015.

[3] Arduino, "Arduino Uno Rev3," [Online]. Available: https://store.arduino.cc/usa/arduino-uno-rev3. [Accessed 26 Mei 2019].

[4] Digi International Incorporated, "IoT Solutions Provider, Hardware, and Wireless Design Services | Digi International," [Online]. Available: https://www.digi.com/resources/documentation/digidocs/PDFs/90000976.pdf. [Accessed 26 Mei 2019].

[5] J. A. Gubner, Probability and Random Processes for Electrical and Computer Engineers, Cambridge: Cambridge University Press, 2006.

[6] M. N. Farid, Penerapan Kompresi Data untuk Efisiensi Daya Jaringan Sensor Nirkabel, Surabaya: Jurusan Teknik Elektro - Institut Teknologi Sepuluh Nopember, 2016.

[7] I. M. Pu, Fundamental Data Compression, Oxford: Elsevier, 2006.

[8] M. Francesco and V. Massimo, "A Simple Algorithm for Data Compression in Wireless Sensor Networks," *IEEE Communication Letter,* vol. 12, no. 6, pp. 411 - 413, 2008.

[9] L. Yao and L. Yimei, "An Efficient and Robust Data Compression Algorithm in Wireless Sensor Networks," *IEEE Communications Letters,* vol. 18, no. 3, pp. 439 - 442, 2014.

[10] A. Makhoul and H. Harb, "Data Reduction in Sensor Networks: Performance Evaluation in a Real Environment," *IEEE Embedded Systems Letters*, vol. 9, 2017.

[11] Y. Beihua, "An Energy-efficient Compression Algorithm for Spatial Data in Wireless Sensor Networks," in *18th International Conference on Advanced Communication Technology*, 2016.

[12] A. Jaber, A. Makhoul, H. Harb, M. A. Taam, C. A. Jaoude and O. Zahwe, "Reducing Data Transmission in Sensor Networks through Kruskal-Wallis model," in *13th International Conference on Wireless and Mobile Computing, Netowrking and Communications (WiMob)*, 2017.

LAMPIRAN A

LEMBAR PENGESAHAN PROPOSAL TUGAS AKHIR

Departemen Teknik Elektro
Fakultas Teknologi Elektro – ITS

EE 184801 TUGAS AKHIR – 6 SKS

12 FEB 2019

Nama Mahasiswa	:	Fachri Akbar Rafsanzani
Nomer Pokok	:	07111540000123
Bidang Studi	:	Telekomunikasi Multimedia
Tugas Diberikan	:	Semester Genap Tahun Ajaran 2018/2019
Dosen Pembimbing	:	1. Dr. Ir. Wirawan, DEA
Judul Tugas Akhir	:	**Implementasi dan Analisis Kinerja Algoritma Kompresi Data pada Jaringan Sensor Nirkabel berbasis Arduino (Implementation and Performance Analysis of Data Compression Algorithm on Arduino-based Wireless Sensor Networks)**

Uraian Tugas Akhir :

Jaringan sensor nirkabel merupakan sebuah jaringan yang terdiri dari beberapa sensor yang saling terhubung secara nirkabel. Jaringan sensor nirkabel umumnya digunakan untuk beberapa hal seperti sistem *monitoring* atau untuk mendeteksi suatu kejadian. Permasalahan yang paling sering ditemui adalah mengenai konsumsi energi dari satu *node*. Untuk mengirim satu bit dari satu *node* ke *node* yang lain atau ke *sink*, diperlukan energi yang cukup besar bila dibandingkan untuk komputasi atau akuisisi data dari sensor. Beberapa pendekatan dilakukan untuk mengurangi masalah yang dialami oleh *node*. Pendekatan yang akan dilakukan di tugas akhir ini yaitu menggunakan teknik kompresi data untuk mengurangi konsumsi energi saat dilakukan transmisi dan diimplementasikan ke dalam mikrokontroller yaitu Arduino. Sebelum dilakukan implementasi ke Arduino, perlu dilakukan simulasi menggunakan Matlab. Arduino sendiri akan digunakan untuk mendapatkan data pengukuran. Dari hasil pengukuran tersebut, diharapkan dapat membuktikan bahwa kompresi data dapat menghemat energi yang diimplementasikan pada Arduino untuk jaringan sensor nirkabel.

Kata Kunci : *Wireless Sensor Networks, Data compression, Arduino, Matlab*

Dosen Pembimbing

Dr. Ir. Wirawan, DEA
NIP : 196311091989031011

Mengetahui,
Kepala Program Studi S1

Dr. Ir. Wirawan, S.T., M.Eng., Ph. D
NIP : 197311192000031001

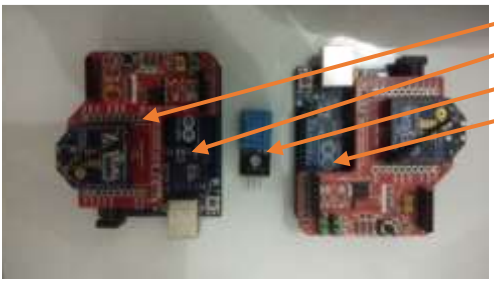
Menyetujui,
Kepala Laboratorium Komunikasi Multimedia

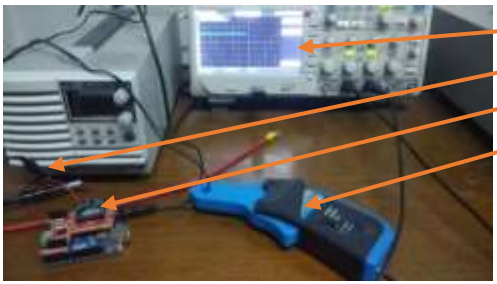
Dr. Ir. Endroyono, DEA
NIP : 196504041991021001

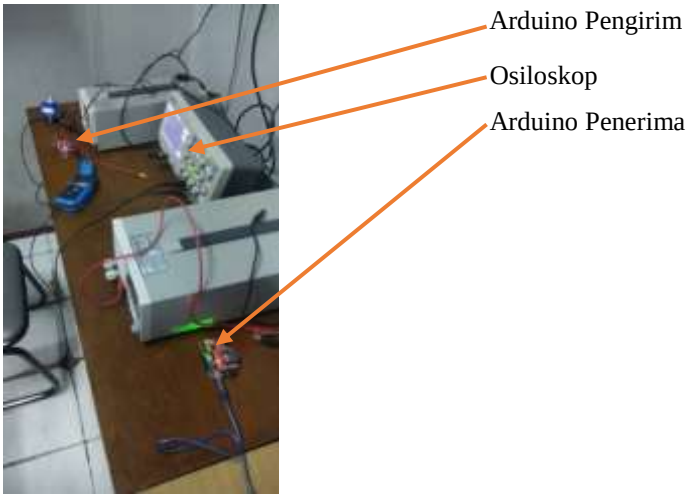
Halaman ini sengaja dikosongkan

LAMPIRAN B

FOTO ALAT DAN PENGUKURAN







Halaman ini sengaja dikosongkan

LAMPIRAN C

KODE PEMROGRAMAN MATLAB

Distribusi Uniform

```
clear all
a=16;
b=35;
u = rand(1,600);
range = b - a + 1;
x = (u*range) + a;
suhu=x;
histogram(x)
```

Distribusi Normal

```
clear all
mu=33;
sigma=1;
i=0;
y=0;
for i=1:600
if (y == 1)
      y = ~y;
      x(1,i) = mu + sigma * X2;
end
if (y == 0)
    while 1
      U1 = -1 + (rand) * 2;
      U2 = -1 + (rand) * 2;
      W = U1^2 + U2^2;
      if (W <= 1 || W == 0)
          break;
      end
    end
  mult = (sqrt ((-2 * log (W)) / W));
  X1 = U1 * mult;
  X2 = U2 * mult;

  y = ~y;
  x(1,i) = mu + sigma * X1;
```

```
end
end
suhu=x;
histfit(x,30,'normal')
```

Distribusi Eksponensial

```
clear all
lambda=.1
u = rand(1,600);
x = -log(1- u) / lambda;
suhu=x;
histfit(x,30,'exponential')
```

Lossless Entropy Coding

```
ri = suhu(2:600)-suhu(1:599);
figure
  histogram(suhu)
figure
  histogram(ri)

sri=size(ri);
hi=[];

firs=de2bi(128+ri(1,1),8,'left-msb');
orig_size=size(firs);
comp_size=0;
% orig_size;
for R=2:sri(1,2)
raw{1,R}=de2bi(128+ri(1,R),8,'left-msb');
orig_size=orig_size+size(raw{1,R});
end
orig_size(1,2)

% Comp_size
for R=1:sri(1,2)
ni=ceil(log2(abs(ri(1,R)))+0.0001);
if ni>4
    hi=[ones(1,(ni-3)) 0];
     else
```

```
         switch ni
        case 1
            hi=[0 1 0];
        case 2
            hi=[0 1 1];
        case 3
            hi=[1 0 0];
        case 4
            hi=[1 0 1];
    end
end

%Pembentukan bitsream
if ri(1,R)==0
    bitstream{1,R}=[0 0];
    else if ri(1,R)>0
        ai(1,R)=ri(1,R);
        ai=de2bi(ai(1,R),'left-msb');
        bitstream{1,R}=[hi ai];
        else if ri(1,R)<0
            ai(1,R)=abs(ri(1,R));
            ai=de2bi(ai(1,R),'left-msb');
            bitstream{1,R}=[hi ~ai];
        end
    end
end
comp_size=comp_size+size(bitstream{1,R});
bs=[bs bitstream{1,R}];
end
comp_size(1,2)
```

Sequential Lossless Entropy Coding

```
ri = suhu(2:600)-suhu(1:599);
figure
  histogram(suhu)
figure
  histogram(ri)

sri=size(ri);
hi=[];
```

```
firs=de2bi(128+ri(1,1),8,'left-msb');
orig_size=size(firs);
comp_size=0;
% orig_size;
for R=2:sri(1,2)
raw{1,R}=de2bi(128+ri(1,R),8,'left-msb');
orig_size=orig_size+size(raw{1,R});
end
orig_size(1,2)

% Comp_size
% LEC
ni=ceil(log2(abs(ri(1,1)))+0.0001);
if ni>4
    hi=[ones(1,(ni-3)) 0];
     else
         switch ni
        case 1
            hi=[0 1 0];
        case 2
            hi=[0 1 1];
        case 3
            hi=[1 0 0];
        case 4
            hi=[1 0 1];
    end
end

%Pembentukan bitstream
if ri(1,1)==0
    bitstream{1,1}=[0 0];
    else if ri(1,1)>0
        ai(1,1)=ri(1,1);
        ai=de2bi(ai(1,1),'left-msb');
        bitstream{1,1}=[hi ai];
        else if ri(1,1)<0
            ai(1,1)=abs(ri(1,1));
            ai=de2bi(ai(1,1),'left-msb');
            bitstream{1,1}=[hi ~ai];
        end
```

```
    end
end
bs=bitstream{1,1};

% S-LEC
for R=2:sri(1,2)
if ri(1,R)==0
    ni(1,R)=0;
else
    ni(1,R)=ceil(log2(abs(ri(1,R)))+0.0001);
end

if ni(1,R)>4
    hi=[ones(1,(ni(1,R)-3)) 0];
     else
         switch ni(1,R)
        case 0
            hi=[0 0];
        case 1
            hi=[0 1 0];
        case 2
            hi=[0 1 1];
        case 3
            hi=[1 0 0];
        case 4
            hi=[1 0 1];
    end
end

%Pembentukan Kode Sequensial
if ni(1,R-1)==ni(1,R)
    si=[0 0];
    hi=[];
elseif ni(1,R-1)-ni(1,R)==1 || (ni(1,R)==2 &&
ni(1,R-1)==0)
        si=[0 1];
        hi=[];
elseif ni(1,R-1)-ni(1,R)==-1
            si=[1 0];
            hi=[];
        else
```

```
            si=[1 1];
end

%Group code reduction
if si==[1 1]
    if (ni(1,R)>ni(1,R-1))
        if (ni(1,R-1)>=0) && (ni(1,R-1)<=3)
            if ni(1,R)==2
                hi(2)=[];
            else
                hi(1)=[];
            end
        elseif (ni(1,R-1)>=4) && (ni(1,R-
1)<=5)
            hi=hi([3:end]);
        elseif (ni(1,R-1)>=6)
            hi=hi([4:end]);
        end
    end
end
%Pembentukan Bitstream
if ri(1,R)==0
    bitstream{1,R}=[si hi];
    else if ri(1,R)>0
        ai(1,R)=ri(1,R);
        ai=de2bi(ai(1,R),'left-msb');
        bitstream{1,R}=[si hi ai];
        else if ri(1,R)<0
            ai(1,R)=abs(ri(1,R));
            ai=de2bi(ai(1,R),'left-msb');
            bitstream{1,R}=[si hi ~ai];
            end
        end
end
bs=[bs bitstream{1,R}];
end
comp_size=comp_size+size(bitstream{1,R});
bs=[bs bitstream{1,R}];
end
comp_size(1,2)
```

LAMPIRAN D

KODE PEMROGRAMAN ARDUINO

Distribusi Uniform

```
double rand_uni (int rangeLow, int rangeHigh)
{
    double myRand = rand ()/ (1.0 + RAND_MAX);
    int range = rangeHigh - rangeLow + 1;
    double myRand_scaled = (myRand * range) +
rangeLow;
    return myRand_scaled;
}
```

Distribusi Normal

```
double ran_norm(double mu, double sigma){
  double U1, U2, W, mult;
  static double X1, X2;
  static int call = 0;
  if (call == 1)
    {
      call =! call;
      return (mu + sigma*(double) X2);
    }
  do
    {
      U1=-1+((double)rand()/RAND_MAX) *2;
      U2=-1+((double)rand()/RAND_MAX) *2;
      W = pow (U1, 2) + pow (U2, 2);
    }
  while (W >= 1 || W == 0);

  mult = sqrt ((-2 * log (W)) / W);
  X1 = U1 * mult;
  X2 = U2 * mult;
  call =! call;
  return (mu + sigma * (double) X1);
}
```

Distribusi Eksponensial

```
double ran_expo (double lambda) {
    double u;
    u = rand () / (RAND_MAX + 1.0);
    return (-log (1- u) / lambda);
}
```

Lossless Entropy Coding

```
String LEC (int ri_suhu) {
  String bitstream, ai;
  int base=2;
  if (ri_suhu==0) {
    ni=0;
    bitstream="00";
  }
  else {
    ni=0;
ni=ceil(log(abs(ri_suhu))/log(base)+0.001);
    ai=dec2bin(ri_suhu);
    bitstream=hi[ni]+ai;
  }
  return bitstream;
}
String dec2bin(int n) {
  int rem;
  String coba="";
  if (n>0) {
    while (n>=1) {
      rem=n%2;
      n/=2;
      if (rem==1) {
        coba="1"+coba;
      }
      else {
        coba="0"+coba;
      }
    }
  }
```

```
  else {
    n=abs(n);
    while (n>=1) {
      rem=n%2;
      n/=2;
      if (rem==1) {
        coba="0"+coba;
      }
      else {
        coba="1"+coba;
      }
    }
  }
return coba;
}
```

Sequential Lossless Entropy Coding

```
String S_LEC (int ri_suhu) {

  int nis, decision1;
  String si[4]={"00","01","10","11"};
  String bitstream,si_tx,hi_tx,ai;

  nis=ni;
  ni=0;
 ni=ceil(log(abs(ri_suhu))/log(base)+0.001);
  decision1=nis-ni;
  if (decision1>1 || decision1<-1) {
    if (ni==2 && nis==0) {
      si_tx=si[1];
    }
    else {
    si_tx=si[3];
    }
  }
  else {
    switch (decision1) {
    case 0 :
```

```
    si_tx=si[0];
    break;
    case 1 :
    si_tx=si[1];
    break;
    case -1 :
    si_tx=si[2];
    break;
    }
  }
  hi_tx=hi[ni];
  if (si_tx=="11" && ni>nis) {
    if (nis>=0 && nis<=3) {
        hi_tx.remove(0,1);
    }
    else {
      if (nis==4 || nis==5) {
        hi_tx.remove(0,2);
      }
      else {
        if (nis>=6) {
          hi_tx.remove(0,3);
        }
      }
    }
  }
    if (ri_suhu==0) {
      ai="";
    }
    if (si_tx=="11") {
      ai=dec2bin(ri_suhu);
      bitstream=si_tx+hi_tx+ai;
    }
    else {
      ai=dec2bin(ri_suhu);
      bitstream=si_tx+ai;
    }
  return bitstream;
}
```

```
String dec2bin(int n) {
  int rem;
  String coba="";
  if (n>0) {
    while (n>=1) {
      rem=n%2;
      n/=2;
      if (rem==1) {
        coba="1"+coba;
      }
      else {
        coba="0"+coba;
      }
    }
  }
  else {
    n=abs(n);
    while (n>=1) {
      rem=n%2;
      n/=2;
      if (rem==1) {
        coba="0"+coba;
      }
      else {
        coba="1"+coba;
      }
    }
  }
return coba;
}
```

Pengirim

```
#include <dht.h>
dht DHT;
const int iPinDHT11 = 5;
double ri_suhu;
int ni;
String tx;
```

```
String hi[16] = {"00", "010", "011", "100",
   "101", "110", "1110", "11110", "111110",
   "1111110", "11111110", "111111110",
   "1111111110", "11111111110",
   "111111111110", "1111111111110"};

void setup() {
  //Start the serial communication
  Serial.begin(9600); //Baud rate must be the
   same as is on xBee module
  pinMode(iPinDHT11, INPUT);
}

void loop() {
  int suhu_skrg, suhu_sblm, suhu_awal = 0,
   chk;
  int  i, j, lop;
  double inttx;
  String kuy;
  //Send the message:
  delay(10000);
  chk = DHT.read11(iPinDHT11);
  suhu_skrg = DHT.humidity;
  Serial.write('<');
  Serial.print((int)suhu_skrg);
  Serial.write('>');//Ending symbol
  while (1) {
    tx = "";
    for (j = 0; j < 40; j++) {
      chk = DHT.read11(iPinDHT11);
      delay(1000);
      suhu_sblm = suhu_skrg;
      suhu_skrg = DHT.humidity;
      ri_suhu = (int)suhu_skrg -
   (int)suhu_sblm;
      tx = tx + LEC((int)ri_suhu);
    }
    tx = "1" + tx;
    lop = tx.length();
```

```
    lop = lop % 8;
    if (lop != 0) {
      for (j = 0; j < 8 - lop; j++) {
        tx = "0" + tx;
      }
    }
    Serial.write('<');  //Starting symbol
    for (j = 0; j < tx.length() / 8; j++) {
      kuy = (String)tx[j * 8] + (String)tx[j *
   8 + 1] + (String)tx[j * 8 + 2] +
   (String)tx[j * 8 + 3] + (String)tx[j * 8 +
   4] + (String)tx[j * 8 + 5] + (String)tx[j *
   8 + 6] + (String)tx[j * 8 + 7];
      inttx = bi2de(kuy);
      //Serial.println((int)inttx);
      if ((int)inttx == 60) {
        Serial.write('0');
        Serial.write('0');
      }
      else if ((int)inttx == 62) {
        Serial.write('1');
        Serial.write('1');
      }
      else
        Serial.write((int)inttx);
    }
    Serial.write('>');//Ending symbol
  }
}
double bi2de (String binary) {
  int i;
  double coba;
  coba = 0;
  for (i = 0; i < 8; i++) {
    if (binary[7 - i] == '1') {
      coba += 1 * pow((double)2, (double)i);
    }
  }
  coba = ceil(coba);
```

```
  return coba;
}
```

Penerima

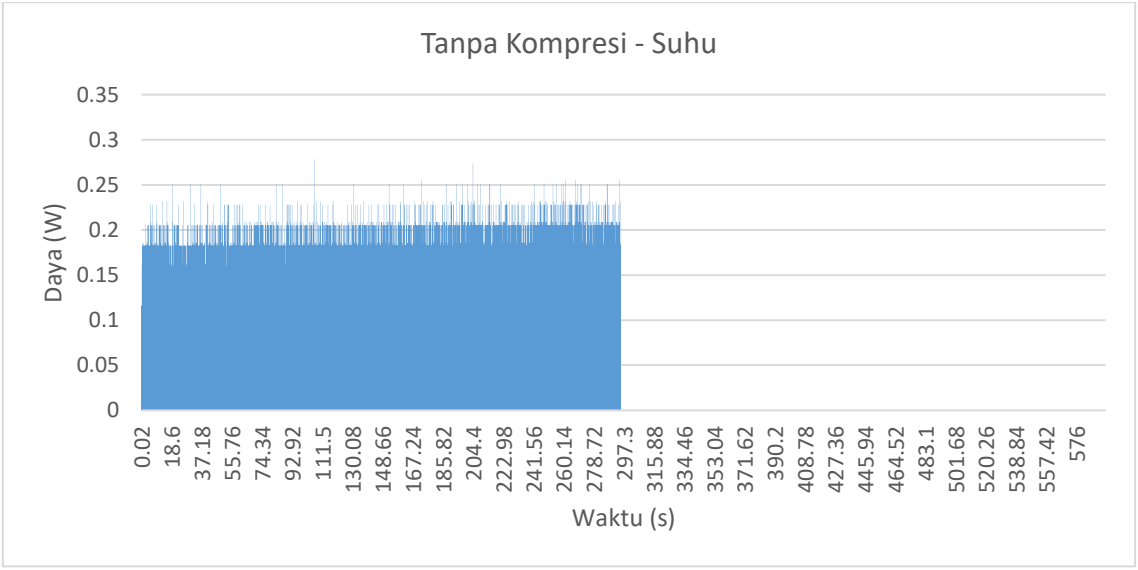
```
//Variables
bool started = false;
bool ended  = false;
char incomingByte ;
char msg[60];
byte index;

void setup() {
  Serial.begin(9600);
}

void loop() {
  receive();
}

void receive() {
  int  i, j, emp;
  String rx;
  inpu = "";
  while (Serial.available() > 0)
    incomingByte = Serial.read();
    if (incomingByte == '<')
    {
      started = true;
      index = 0;
      msg[index] = '\0';
    }
    else if (incomingByte == '>')
    {
      ended = true;
      break;
    }
    else
    {
      if (index < 59)
```

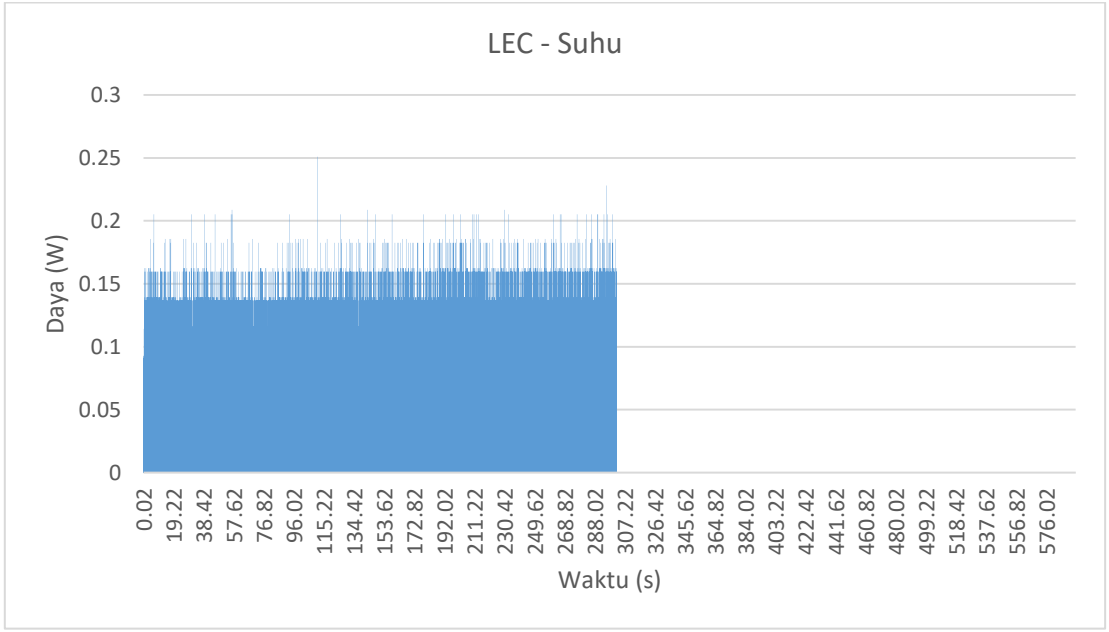
```
    {
      msg[index] = incomingByte;
      index++;
      msg[index] = '\0';
    }
  }
}

if (started && ended && s == 0)
{
  suhu_skrg = String(msg).toInt();
  Serial.println(msg);
  index = 0;
  msg[index] = '\0';
  started = false;
  ended = false;
  s = 1;
}
if (started && ended && s == 1) {

  for (i = 0; i < index; i++) {
    if (msg[i] == '0') {
      if (msg[i + 1] == '0') {
        msg[i] = 60;
        for (j = i + 1; j < index; j++) {
          msg[j] = msg[j + 1];
        }
        index = index - 1;
      }
    }
    if (y [i] == '1') {
      if (y [i + 1] == '1') {
        msg[i] = 62;
        for (j = i + 1; j < index; j++) {
          msg[j] = msg[j + 1];
        }
        index = index - 1;
      }
    }
```

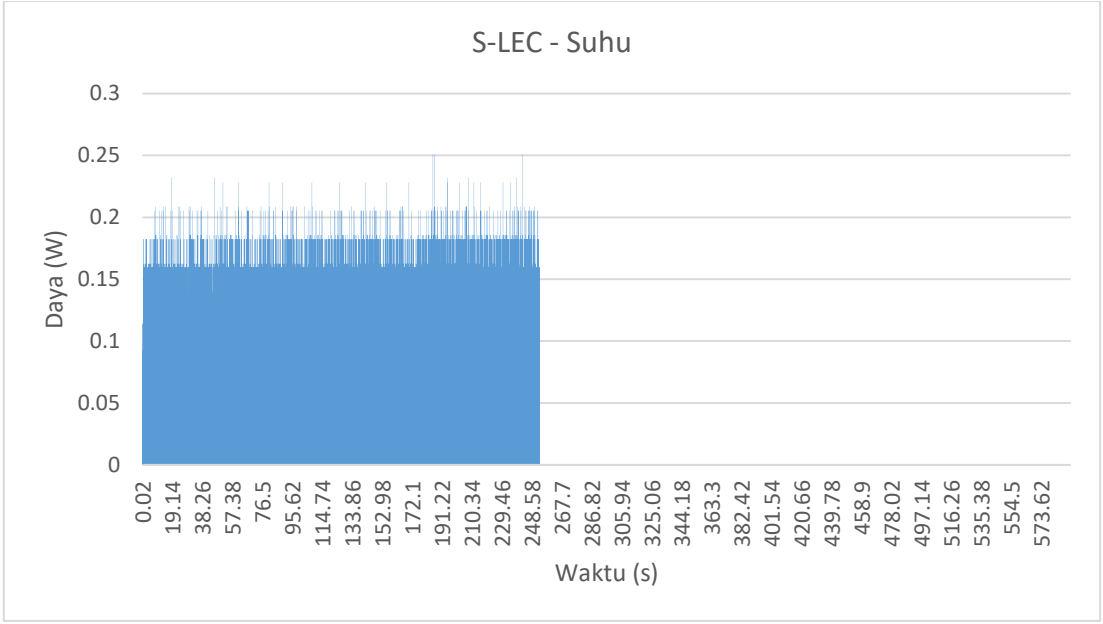
```
    }
    for (i = 0; i < index; i++) {
      if (msg[i] < 0) {
        nggon[i] = (int)msg[i] + 256;
      }
      else
        nggon[i] = (int)msg[i];
      Serial.print(nggon[i]);
      Serial.print(',');
    }
    Serial.println();
    index = 0;
    msg[index] = '\0';
    started = false;
    ended = false;
    s = 1;
  }
}
```

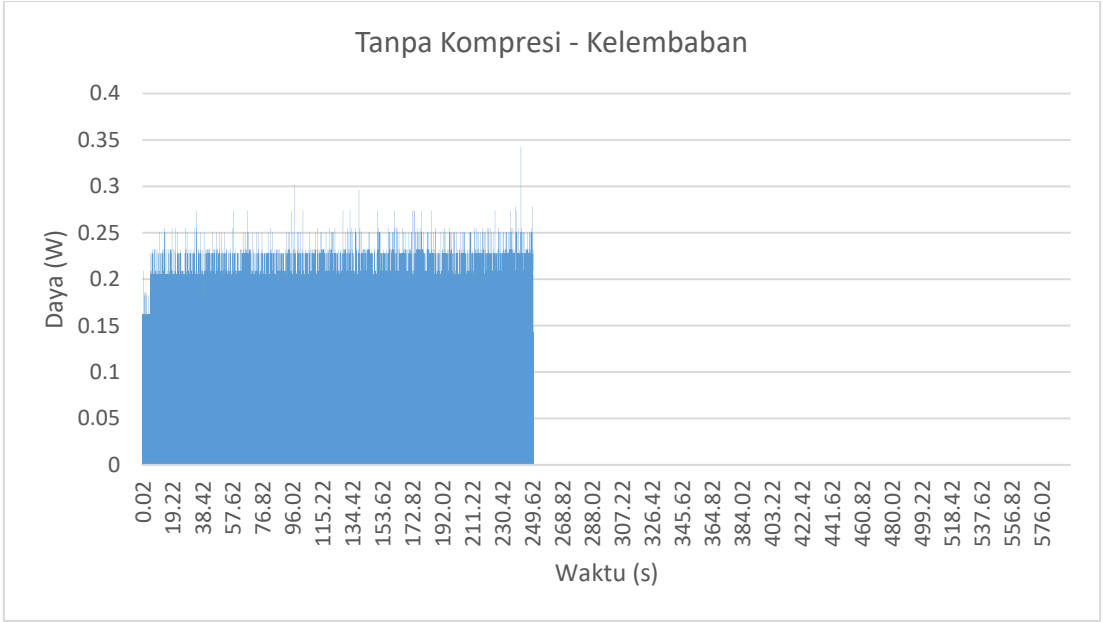
LAMPIRAN E

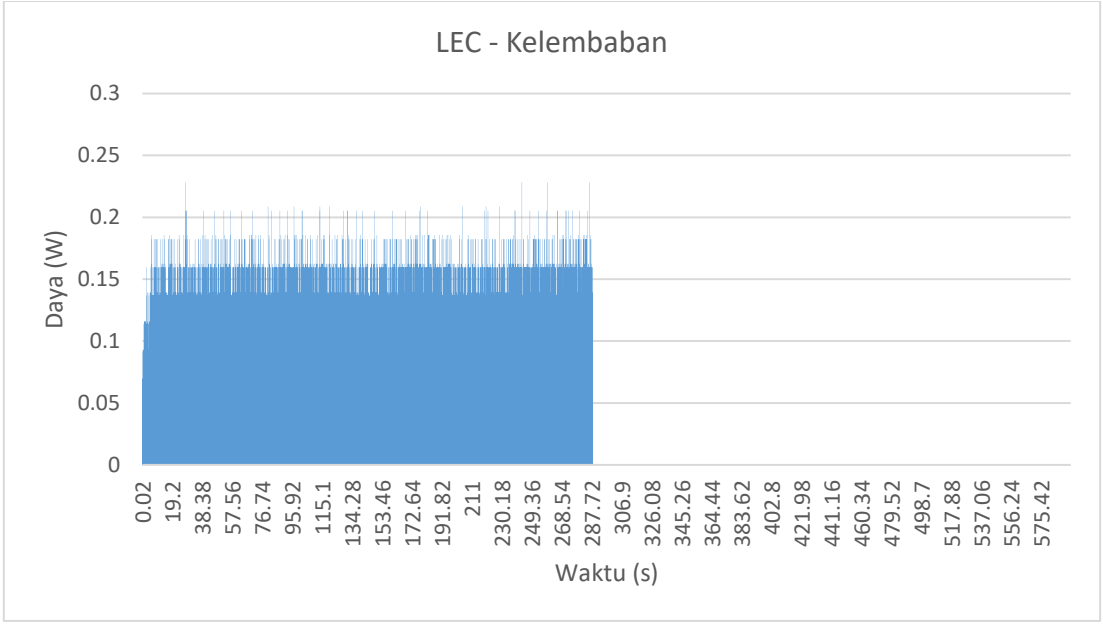
HASIL PENGUKURAN

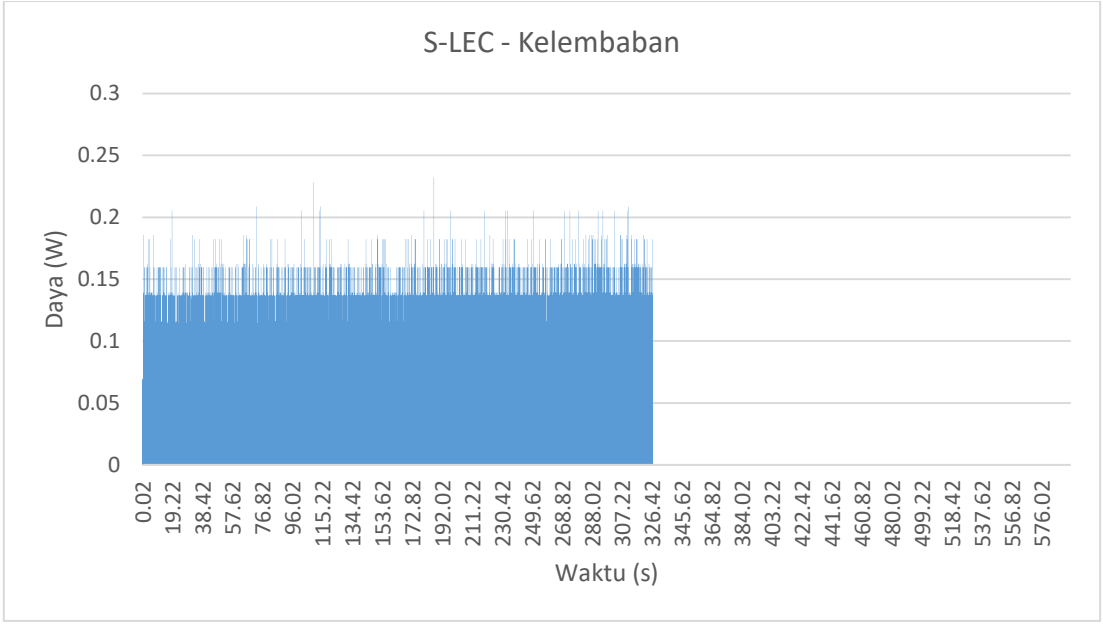


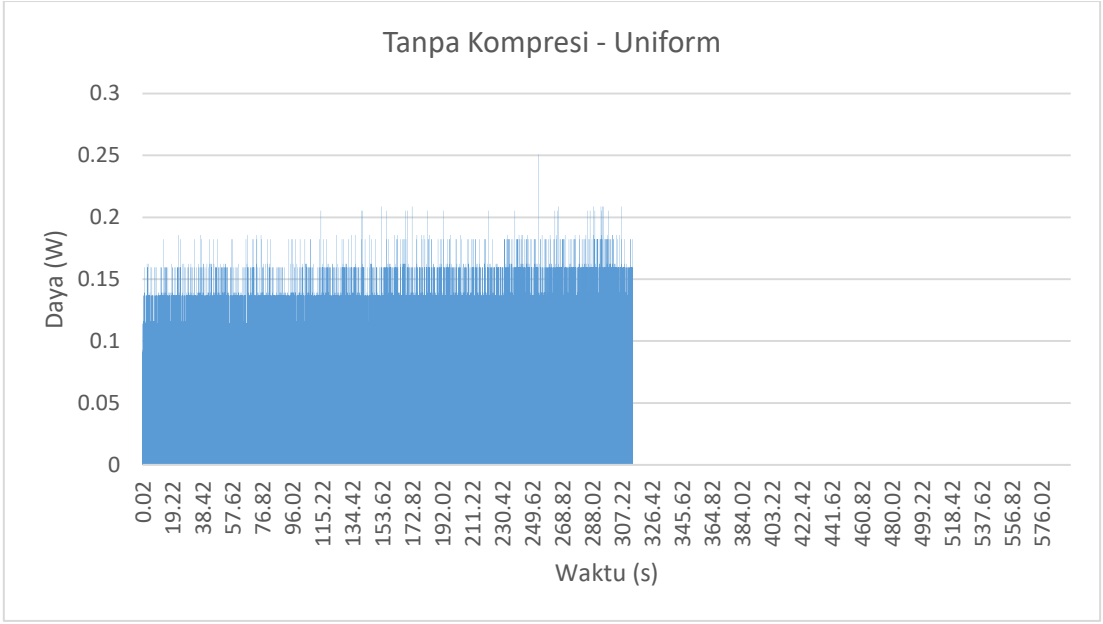

LEC - Suhu
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.22
38.42
57.62
76.82
96.02
115.22
134.42
153.62
172.82
192.02
211.22
230.42
249.62
268.82
288.02
307.22
326.42
345.62
364.82
384.02
403.22
422.42
441.62
460.82
480.02
499.22
518.42
537.62
556.82
576.02
Waktu (s)

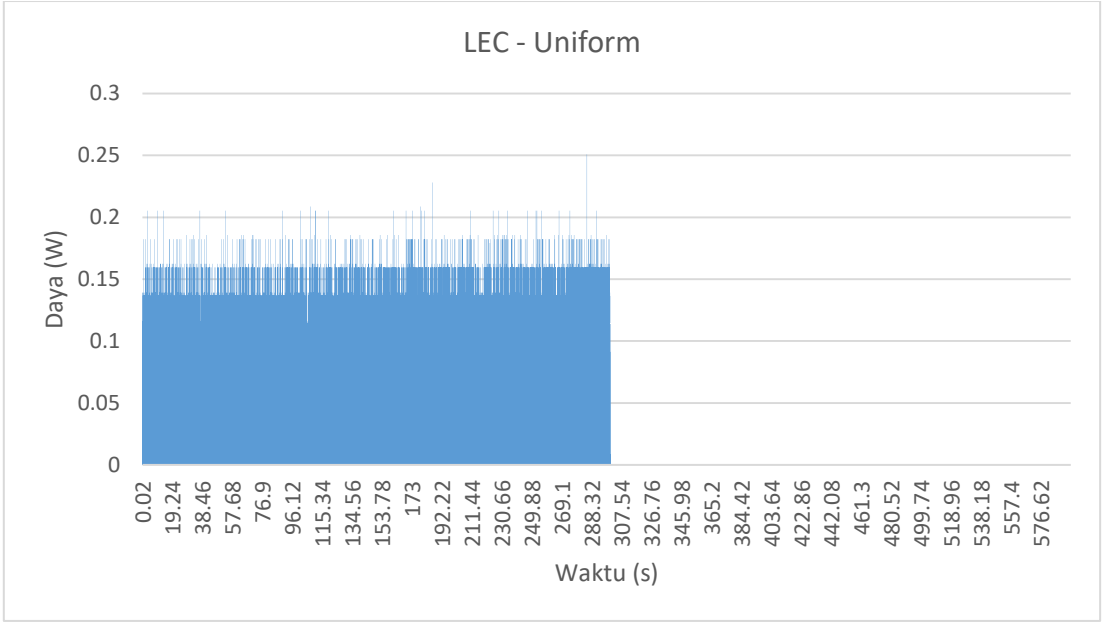

S-LEC - Suhu
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.14
38.26
57.38
76.5
95.62
114.74
133.86
152.98
172.1
191.22
210.34
229.46
248.58
267.7
286.82
305.94
325.06
344.18
363.3
382.42
401.54
420.66
439.78
458.9
478.02
497.14
516.26
535.38
554.5
573.62
Waktu (s)

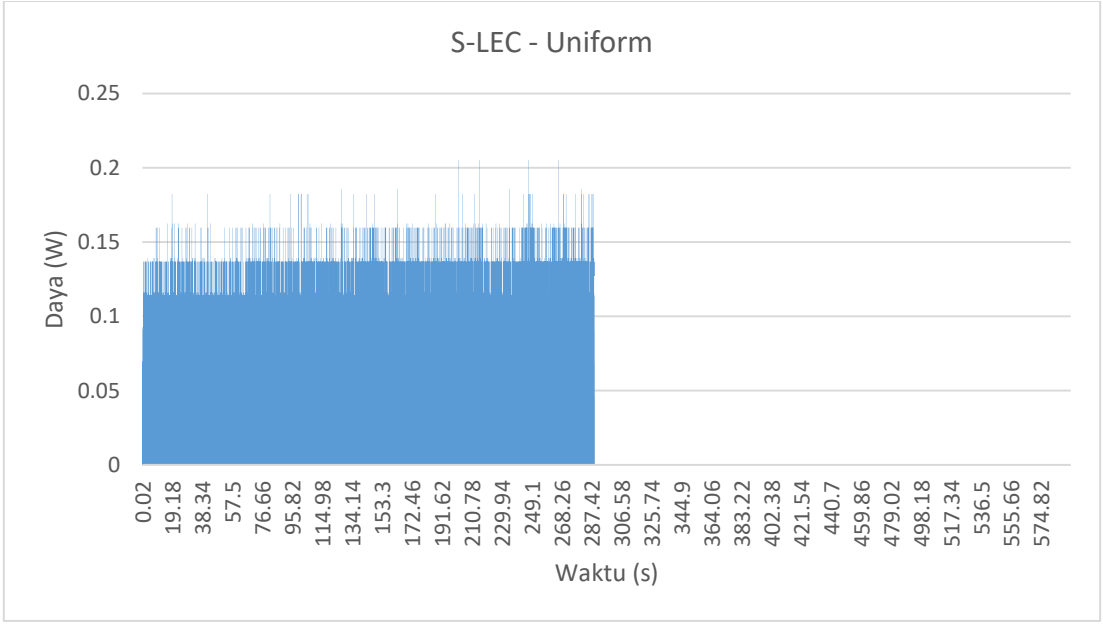

Tanpa Kompresi - Kelembaban
Daya (W)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.22
38.42
57.62
76.82
96.02
115.22
134.42
153.62
172.82
192.02
211.22
230.42
249.62
268.82
288.02
307.22
326.42
345.62
364.82
384.02
403.22
422.42
441.62
460.82
480.02
499.22
518.42
537.62
556.82
576.02
Waktu (s)

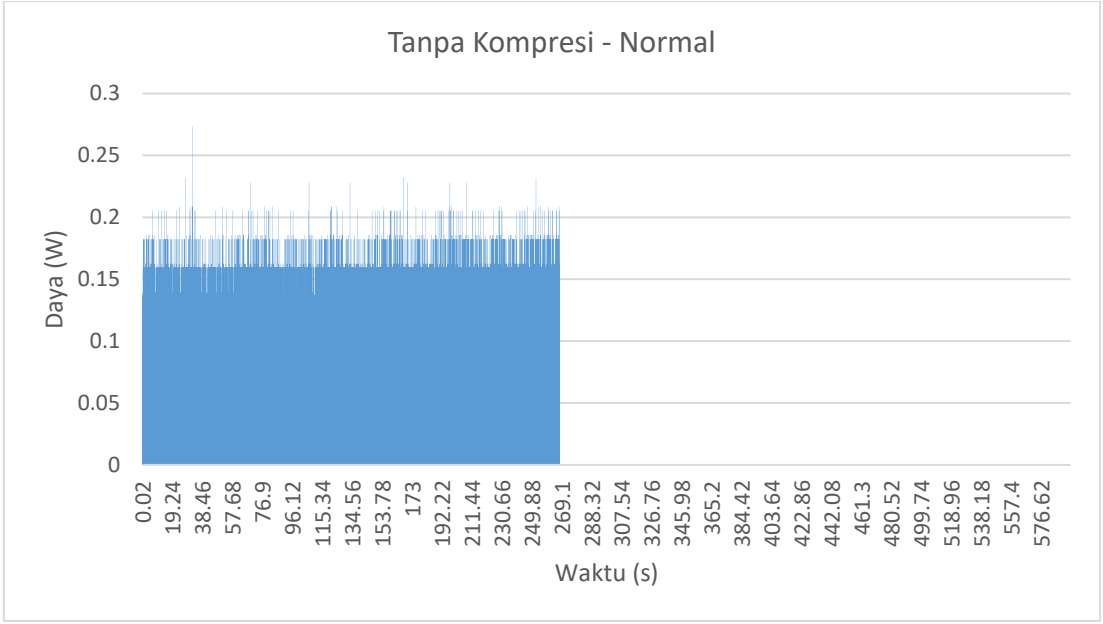

LEC - Kelembaban
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.2
38.38
57.56
76.74
95.92
115.1
134.28
153.46
172.64
191.82
211
230.18
249.36
268.54
287.72
306.9
326.08
345.26
364.44
383.62
402.8
421.98
441.16
460.34
479.52
498.7
517.88
537.06
556.24
575.42
Waktu (s)

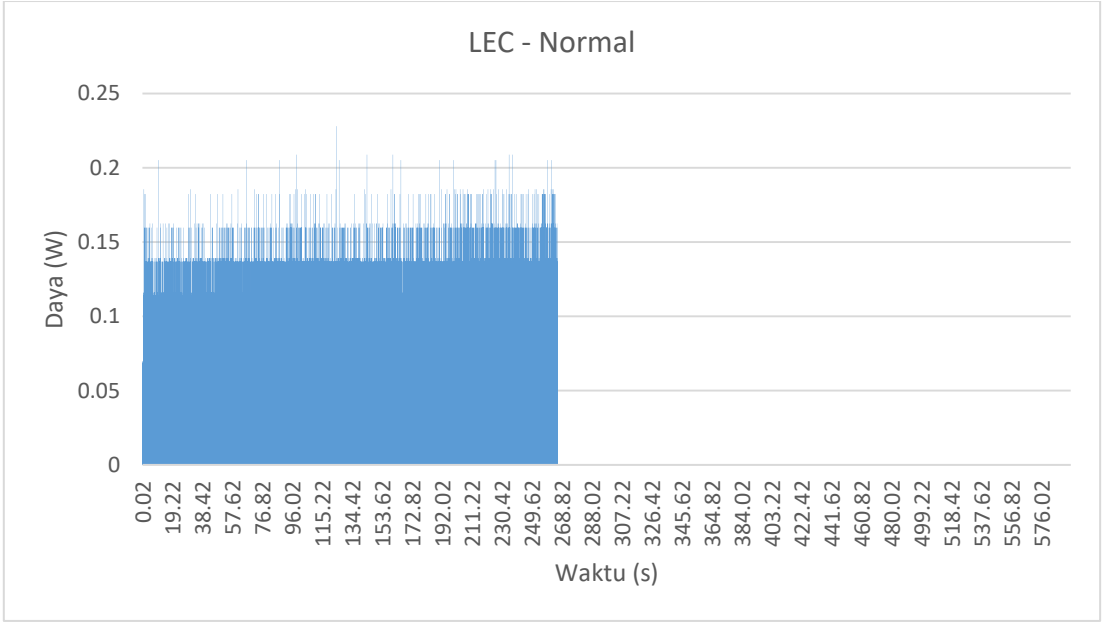

S-LEC - Kelembaban
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.22
38.42
57.62
76.82
96.02
115.22
134.42
153.62
172.82
192.02
211.22
230.42
249.62
268.82
288.02
307.22
326.42
345.62
364.82
384.02
403.22
422.42
441.62
460.82
480.02
499.22
518.42
537.62
556.82
576.02
Waktu (s)

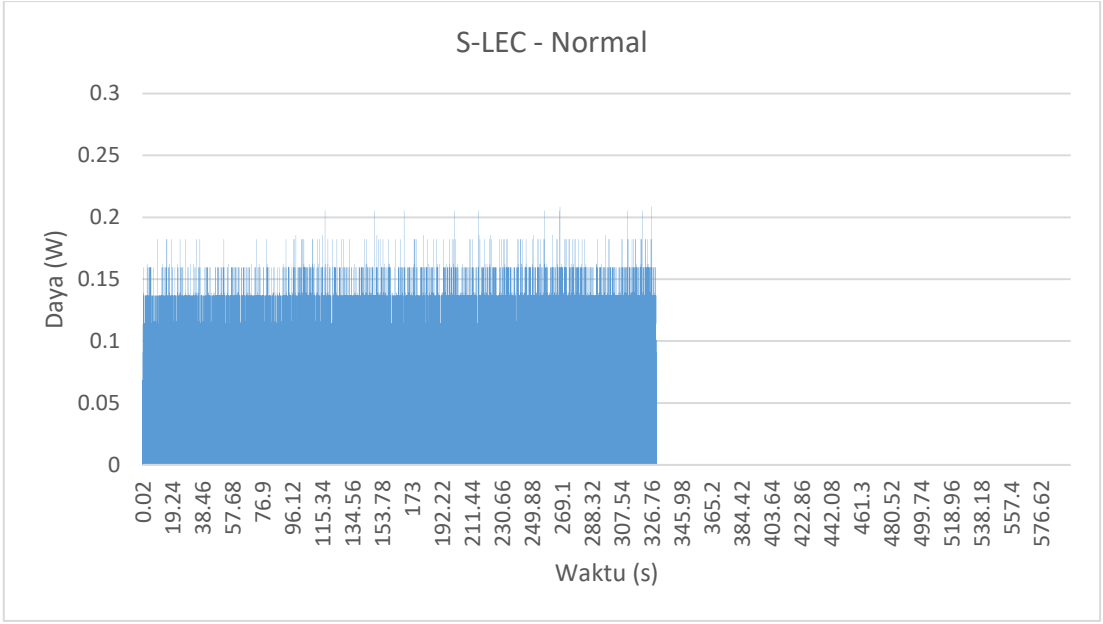

Tanpa Kompresi - Uniform
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.22
38.42
57.62
76.82
96.02
115.22
134.42
153.62
172.82
192.02
211.22
230.42
249.62
268.82
288.02
307.22
326.42
345.62
364.82
384.02
403.22
422.42
441.62
460.82
480.02
499.22
518.42
537.62
556.82
576.02
Waktu (s)

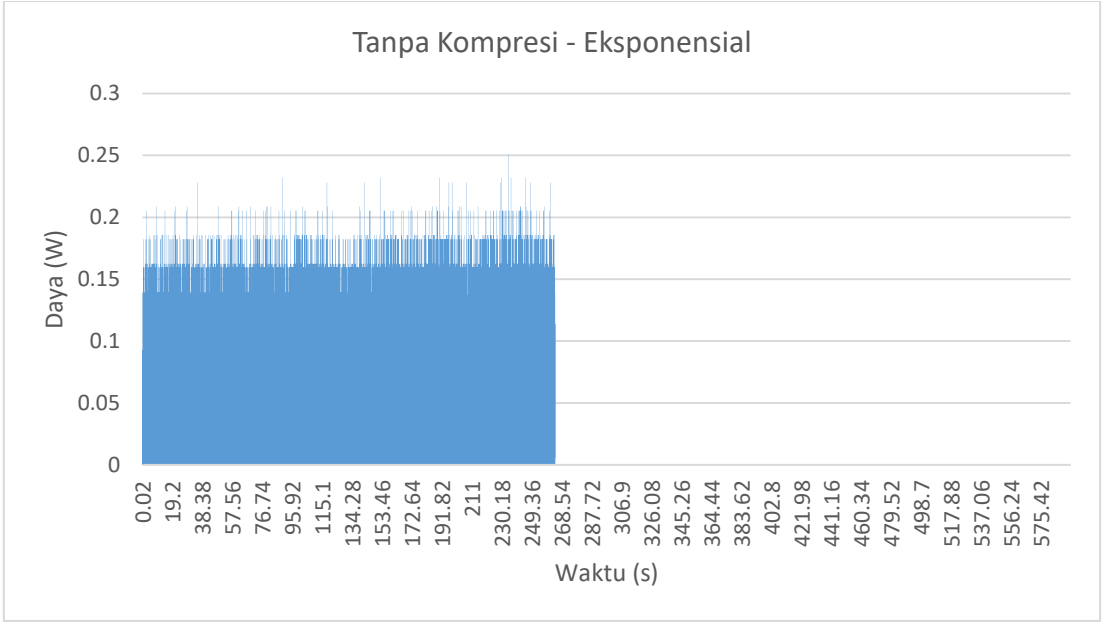

LEC - Uniform
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.24
38.46
57.68
76.9
96.12
115.34
134.56
153.78
173
192.22
211.44
230.66
249.88
269.1
288.32
307.54
326.76
345.98
365.2
384.42
403.64
422.86
442.08
461.3
480.52
499.74
518.96
538.18
557.4
576.62
Waktu (s)

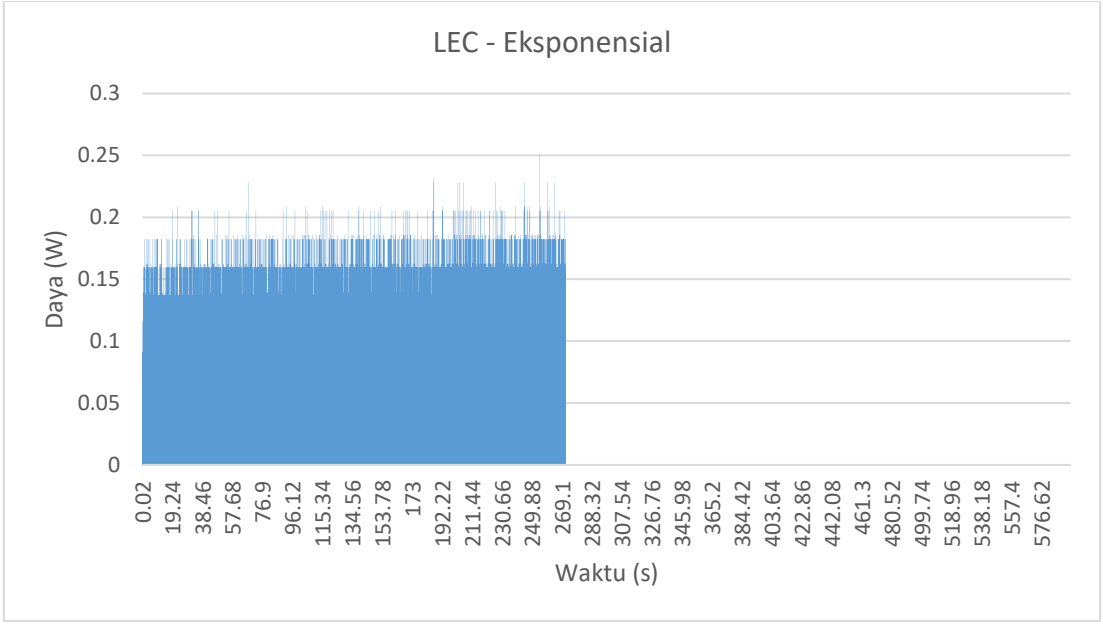

S-LEC - Uniform
0.25
0.2
0.15
0.1
0.05
0
Daya (W)
0.02
19.18
38.34
57.5
76.66
95.82
114.98
134.14
153.3
172.46
191.62
210.78
229.94
249.1
268.26
287.42
306.58
325.74
344.9
364.06
383.22
402.38
421.54
440.7
459.86
479.02
498.18
517.34
536.5
555.66
574.82
Waktu (s)

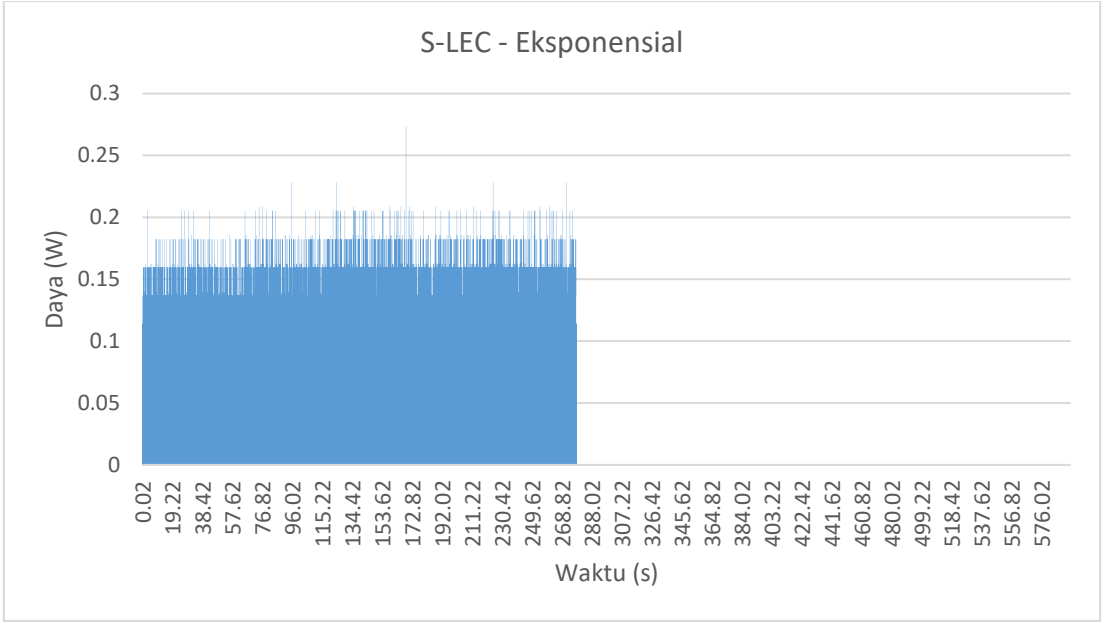

Tanpa Kompresi - Normal
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.24
38.46
57.68
76.9
96.12
115.34
134.56
153.78
173
192.22
211.44
230.66
249.88
269.1
288.32
307.54
326.76
345.98
365.2
384.42
403.64
422.86
442.08
461.3
480.52
499.74
518.96
538.18
557.4
576.62
Waktu (s)


LEC - Normal
0.25
0.2
0.15
0.1
0.05
0
Daya (W)
0.02
19.22
38.42
57.62
76.82
96.02
115.22
134.42
153.62
172.82
192.02
211.22
230.42
249.62
268.82
288.02
307.22
326.42
345.62
364.82
384.02
403.22
422.42
441.62
460.82
480.02
499.22
518.42
537.62
556.82
576.02
Waktu (s)


S-LEC - Normal
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.24
38.46
57.68
76.9
96.12
115.34
134.56
153.78
173
192.22
211.44
230.66
249.88
269.1
288.32
307.54
326.76
345.98
365.2
384.42
403.64
422.86
442.08
461.3
480.52
499.74
518.96
538.18
557.4
576.62
Waktu (s)


Tanpa Kompresi - Eksponensial
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.2
38.38
57.56
76.74
95.92
115.1
134.28
153.46
172.64
191.82
211
230.18
249.36
268.54
287.72
306.9
326.08
345.26
364.44
383.62
402.8
421.98
441.16
460.34
479.52
498.7
517.88
537.06
556.24
575.42
Waktu (s)


LEC - Eksponensial
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.24
38.46
57.68
76.9
96.12
115.34
134.56
153.78
173
192.22
211.44
230.66
249.88
269.1
288.32
307.54
326.76
345.98
365.2
384.42
403.64
422.86
442.08
461.3
480.52
499.74
518.96
538.18
557.4
576.62
Waktu (s)


S-LEC - Eksponensial
Daya (W)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
19.22
38.42
57.62
76.82
96.02
115.22
134.42
153.62
172.82
192.02
211.22
230.42
249.62
268.82
288.02
307.22
326.42
345.62
364.82
384.02
403.22
422.42
441.62
460.82
480.02
499.22
518.42
537.62
556.82
576.02
Waktu (s)

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Fachri Akbar Rafsanzani, lahir 16 Juli 1998 di Kediri, merupakan anak pertama dari 3 bersaudara. Penulis mempunyai kegemaran berolahraga dan menonton film. Setelah lulus dari SMAN 2 Kediri pada tahun 2015, penulis melanjutkan pendidikan Departemen Teknik Elektro program sarjana, bidang studi Telekomunikasi Multimedia, Institut Teknologi Sepuluh Nopember hingga tahun 2019 menyelesaikan studinya dengan mengikuti seminar dan sidang tugas akhir sebagai syarat kelulusan dan mendapatkan gelar sarjana.