CUBESAT TELEMETRY DECODER SOFTWARE

A thesis submitted in partial fulfillment of the requirement for the degree of
B.Sc. (HONS) Electrical and Electronic Engineering
(ELECTRONIC SYSTEMS SOFTWARE ENGINEERING)

BY:
SARA YOUSIF MOHAMED MAHMOUD
INDEX NO. 064031

Supervisor
Dr. Tahani Abdalla Attia

TO
Department of Electrical and Electronic Engineering
Faculty of Engineering
University of Khartoum

July 2011
Declaration of Originality

I declare that this report entitled “CubeSat Telemetry Decoder Software” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature: _________________________
Name: _________________________
Date: _________________________
Acknowledgment

From the very beginning to the end I thank Allah who provided me with health and strength and through whom a number of relatives and friends and many other helped me throughout this project.

I would like to express my deep gratitude to my supervisor Dr. Tahani Abdalla Attia for her valuable and patient through the project.

I acknowledge the deep gratitude and appreciation to Dr. Nader Omer and KNSAT Team for their help and encouragement during the project period.

I wish to express my appreciation to my project partner, Rayan Mohamed Elmubarak, for her hard work, team spirit and the impressive times we spent.

I am as ever, indebted to my family, especially my parents for supporting me throughout my life.
Abstract

The CubeSat accumulates and sends its housekeeping data to ground station in encoded format, the ground station needs a system to decode and display this data for the user.

This project aims to build a telemetry decoder software that decodes the data received from the CubeSat and displays it to the user in a friendly user interface program and gives the user the ability to do some process on the data.

To fulfill this aim a software program is written and implemented by java programming language to build the decoder system. The system consist of four modules: data decoding, chart drawing, data display and data storage.

The data decoding module gets the encoded data format and converts it to the original values that the CubeSat accumulated, some specific algorithms are used to decode the data.

The drawing chart module gives the user the ability to draw charts of some components during a specific time to note the variation of its value through that time.

The data should be displayed to the user in obvious way and allows the user to monitor the status of the CubeSat by an easy interface and gives alarm for danger values.

The decoded data can be stored in databases thus the user can review it later.

The system was tested in the KNSAT ground station and succeed to perform its objectives.
المستخلص

يقوم القمر الاصطناعي بتجميع بيانات وإرسالها بترميز محدد، تستقبل المحطة الأرضية هذه الإشارة

مرزمة لذا فإنها تحتاج إلى نظام يقوم بفك هذا الترميز وعرضها للمستخدم.

يهدف هذا المشروع لبناء برنامج فك ترميز بيانات تم قياسها عن بعد، يقوم البرنامج بفك البيانات المستلمة من القمر الاصطناعي المكعب وعرضها للمستخدم بصورة سهلة ومفهومة وإعطاء المستخدم إمكانية إجراء بعض العمليات على هذه البيانات.

لتحقيق هذا الهدف، تم كتابة وتصميم برنامج باستخدام لغة برمجة الجافا لبناء نظام فك الترميز، يحتوي النظام على أربع مهام أساسية: فك الترميز، الرسم البياني، عرض البيانات وتخزين البيانات.

فلك الترميز يقوم بإخراج البيانات المرزمة وتخويلها إلى قيمها الأصلية التي جمعها القمر الاصطناعي المكعب وفق نسق محدد يتم إتباعه لفك الترميز.

الرسم البياني يعني المستخدم إمكانية رسم تخطيطي لقيم بعض مكونات القمر الاصطناعي خلال فترة زمنية محددة، حتى يتمكن المستخدم من ملاحظة التغيرات في قيم هذه المكونات خلال تلك الفترة.

يتم عرض هذه البيانات للمستخدم بصورة واضحة بصورة مكتملة من مناحي حالة القمر الاصطناعي المكعب في واجهة سهلة الاستخدام مع إمكانية تتبع المستخدم لقيم البياني متى يظهر خطأ على القمر الاصطناعي. يمكن تخزين البيانات التي تم فك ترميزها في قواعد بيانات حتى يتمكن المستخدم من العودة إليها لاحقًا.

تم اختبار النظام وتشغيله في المحطة الأرضية وقد قام بتثبيته بنجاح.
# Table of Contents

Declaration of Originality .............................................................................................................. ii

Acknowledgment .......................................................................................................................... iii

Abstract ........................................................................................................................................ iv

المستخلص ...................................................................................................................................... v

Table of Contents .......................................................................................................................... vi

List of Figures ............................................................................................................................... x

List of Tables ................................................................................................................................... xii

List of Abbreviations ...................................................................................................................... xiii

1 CHAPTER ONE : INTRODUCTION ............................................................................................. 1

1.1 Overview .................................................................................................................................. 1

1.2 Problem Definition .................................................................................................................. 1

1.3 Background and Motivation .................................................................................................. 1

1.4 Objectives ............................................................................................................................... 2

1.5 Thesis Layout ......................................................................................................................... 2

2 CHAPTER TWO : CUBESAT COMMUNICATION SYSTEM ......................................................... 3

2.1 Introduction ............................................................................................................................ 3

2.2 About CubeSat ....................................................................................................................... 3

2.2.1 CubeSat Definition .......................................................................................................... 3

2.2.2 Standards And Usage ...................................................................................................... 3

2.2.3 CubeSat History .............................................................................................................. 4

2.3 Previous Software Evolution ................................................................................................. 4

2.3.1 AO-7 Telemetry RTTY decoder ...................................................................................... 4

2.3.2 CUTE –I CW Telemetry Decoder Software .................................................................... 5

2.3.3 O/OREOS Telemetry Decoder ....................................................................................... 6
2.3.4 HITSAT Simple CW Telemetry Decoder.......................6
2.3.5 COMPASS-1 CW Telemetry Decoder .........................7

2.4 CubeSat Communication System..................................8
  2.4.1 Ground Station..............................................8
  2.4.2 Amateur Radio ............................................8
  2.4.3 CubeSat Communication Link Types .....................9
  2.4.4 Limited Communication Window .........................9

2.5 Communication Protocols .......................................10
  2.5.1 Morse Code.................................................10
  2.5.2 Continuous Wave..........................................11
  2.5.3 AX.25 Protocol ...........................................11

2.6 CubeSat Telemetry...............................................12
  2.6.1 Telemetry History ..........................................12
  2.6.2 Tracking, Telemetry And Command (TT&C) .............13
  2.6.3 Telemetry Data ............................................14
  2.6.4 Command And Data Handling (C&DH) ...................14

2.7 Encoding and Decoding ...........................................15
  2.7.1 Encoding.....................................................15
  2.7.2 Decoding.....................................................16
  2.7.3 CubeSat’s Data Format ...................................16
  3.7.4 Sound Decoders ..........................................19

3  CHAPTER THREE : CUBESAT TELEMETRY DECODER SOFTWARE IMPLEMENTATION ......22

3.1 Introduction ......................................................22
3.2 System Requirements............................................22
  3.2.1 Functional Requirements.................................22
  3.2.2 Non-functional Requirements ............................23
3.3 Assumptions .....................................................23
3.4 Implementation Tools...........................................23
3.5 Data Format .....................................................24
  3.5.1 Alternatives ...............................................24
  3.5.2 KNSAT Data Format ......................................24
3.6 Java As Object Oriented Programming OOP..................26
  3.6.1 The need of OOP .........................................26
List of Figures

Figure 2.1 Standard CubeSat ................................................................. 4
Figure 2.2 AO-7 Telemetry RTTY Decoder ........................................... 5
Figure 2.3 CUTe-I CW Telemetry Decoder ........................................... 5
Figure 2.4 O/OREOS Telemetry Decoder .............................................. 6
Figure 2.5 HITSAT Simple CW Telemetry Decoder ................................. 7
Figure 2.6 COMPASS-1 CW Telemetry Decoder ................................... 7
Figure 2.7 Communication Window in Terms of AOS and LOS .................. 10
Figure 2.8 MixW Sound Decoder .......................................................... 20
Figure 2.9 MULTIPSK Sound Decoder .................................................. 21
Figure 3.1 Development Plan ............................................................... 28
Figure 3.2 Telemetry Decoder CLASS Diagram ..................................... 36
Figure 3.3 USE UASE Diagram ............................................................ 37
Figure 3.4 Activity Diagram .................................................................. 38
Figure 4.1 Program Start Tab ............................................................... 40
Figure 4.2 Beacon Tab .......................................................................... 41
Figure 4.3 Options Tab .......................................................................... 41
Figure 4.4 Beacon Text File ................................................................. 42
Figure 4.5 Import Beacon Text File ....................................................... 42
Figure 4.6 Decoded Values ................................................................. 43
Figure 4.7 Multiple Beacons Files ......................................................... 43
Figure 4.8 Next and Previous ............................................................... 44
Figure 4.9 Chart Display ..................................................................... 45
Figure 4.10 Change Chart Option ........................................................... 45
Figure 4.11 Chart Display in Separate Window ....................................... 46
Figure 4.12 Decoded with Alarm ............................................................ 46
Figure 4.13 Save Decoded Data In Local Database .................................. 47
Figure 4.14 Local Database Saving ....................................................... 47
Figure 4.15 The Save as Decoded Data .................................................. 48
Figure 4.16 Save As In Text File ............................................................. 48
Figure 4.17 Themes Options ................................................................. 49
Figure 4.18 Pink Theme ....................................................................... 49
Figure 4.19 Yellow Theme ................................................................... 50
Figure 4.20 Fahrenheit Options ............................................................ 50
Figure 4.21 Fahrenheit Unit Display ...................................................... 51
Figure 4.22 Import Help ...................................................................... 51
Figure 4.23 Save Help .......................................................................... 51
Figure 4.24 Not Beacon Text File ......................................................... 52
Figure 4.25 Importing Wrong Text File .................................................. 52
Figure 4.26 Wrong file Importing ......................................................... 53
Figure 4.27 Non-Importing Saving ......................................................... 54
FIGURE 4.28 WRONG SAVING .....................................................................................54
FIGURE 4.29 KNSAT TELEMETRY DECODER IN GROUND STATION UoFK ..............57
List of Tables

TABLE 2.1 LOW EARTH ORBIT CHARACTERISTICS .......................................................... 9
TABLE 2.2 COMPASS-1 CW TELEMETRY DECODER FORMAT .................................. 16
TABLE 2.3 O/OREOS TELEMETRY DECODER FORMAT .......................................... 17
TABLE 2.4 CUTE-1 - CW TELEMETRY FORMAT ...................................................... 18
TABLE 2.5 HIT-SAT CW FORMAT ........................................................................... 18
TABLE 3.1 KNSAT DATA FORMAT ........................................................................ 24
TABLE A.1 KNSAT DATA FORMAT ......................................................................... A-1
TABLE A.2 TRANSMISSION MODES CODES ............................................................. A-2
TABLE A.3 PAYLOAD REQUEST CODES ................................................................ A-2
TABLE A.4 STATUSES CODES ................................................................................ A-2
TABLE A.5 BATTERY VOLTAGE ............................................................................. A-2
TABLE A.6 TEMPERATURE AND VOLTAGE CODES ............................................. A-3
TABLE A.7 BUS VOLTAGE CODES ......................................................................... A-4
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFSK</td>
<td>Audio Frequency Shift Keying</td>
</tr>
<tr>
<td>AOS</td>
<td>Acquisition Of Signal</td>
</tr>
<tr>
<td>C&amp;DH</td>
<td>Command And Data Handling</td>
</tr>
<tr>
<td>CUTE-I</td>
<td>Cubical Titech Engineering satellite</td>
</tr>
<tr>
<td>CW</td>
<td>Continuous Wave</td>
</tr>
<tr>
<td>FSK</td>
<td>Frequency Shift Keying</td>
</tr>
<tr>
<td>HITSAT</td>
<td>Hokkaido Institute of Technology Satellite</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Run time Environment</td>
</tr>
<tr>
<td>KNSAT</td>
<td>Khartoum Nile Satellite</td>
</tr>
<tr>
<td>LEO</td>
<td>low-earth orbit</td>
</tr>
<tr>
<td>LOS</td>
<td>Loss Of Signal</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>OBC</td>
<td>Onboard Computer</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>P-POD</td>
<td>Poly- Picosatellite Orbital Deployer</td>
</tr>
<tr>
<td>TT&amp;C</td>
<td>Tracking, Telemetry And Command</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
</tbody>
</table>
1 CHAPTER ONE : INTRODUCTION

1.1 Overview

After the CubeSat is launched to the space, the ground station needs to know the status of the CubeSat, thus it accumulates the payload and its housekeeping then sends this data to ground station. There are many protocols of sending this data, so it is encoded to simpler format before sending. The ground station needs to decode this data and extract the original values to be displayed to the user.

A telemetry decoder software provides the ground station with capability to decode the received data and monitor the CubeSat status in easy and efficient way, the telemetry software includes many functions as decoding beacons data, draw charts of some component values and saving these data in databases.

The telemetry decoder is a unique system for the CubeSat, Every CubeSat has its own telemetry decoder since the data format is different from one to another.

1.2 Problem Definition

The CubeSat in the space is considered a piece of junk if no communication available. The telemetry decoder was constructed for the following reasons:

1. Without received stream is ambiguous string that ground station will not be able to make use of it.
2. With a user-friendly interface the user will be able to view the beacon values easily.
3. It is important to know the damaged component and determining the malfunctioning devices.
4. Another problem is that monitoring the deviation of values through time.

1.3 Background and Motivation

Every CubeSat has its own Telemetry decoder software. Without it the ground station will not be able to know the state of the CubeSat that have been sent. With the telemetry decoder software the ground station also is able no notice the variation of the CubeSat values of its components.
With the alarm the telemetry decoder software is considered a CubeSat monitoring system that can detect danger values in the CubeSat and problems occur in it. The Telemetry decoder provides the ability to go through many beacons at a time.

### 1.4 Objectives

The objective of this project is to develop a CubeSat telemetry decoder that is able to:

- Decode beacon that is received from the CubeSat to meaningful values that let the user know the CubeSat’s status.
- Display decoded values to the user in easy user-friendly interface window that allows the user to go through many beacon’s values.
- Construct charts of some values for several beacons.
- Give alarm for out of range decoded values, so the user is aware of the danger in the CubeSat.
- Save the decoded values in a format that the user can preview at any time later.

### 1.5 Thesis Layout

This thesis is presented in five chapters. The next chapter presents an introduction of the Cubesat, previous CubeSat telemetry decoders. Then it discuss the CubeSat communication system and overview of this communication protocols. As well as presenting the telemetry, encoding and decoding overview, Some CubeSats’ data formats were presented.

Chapter 3 details the objectives of the project, and focuses on how those objectives were achieved. Also, it presents the design methodology, and implementation of the system, and testing methods.

Chapter 4 contains the results of the system testing for all conditions, discussion of the results obtained and Special features in the system.

Chapter 5 ends the thesis with a conclusion and future work.

Appendix A shows the data format tables of the KNSAT CubeSat.
2 CHAPTER TWO : CUBESAT COMMUNICATION SYSTEM

2.1 Introduction
Arguably, one of the most important parts of any satellite is the communications subsystem. Without any way to communicate, the CubeSat would quickly become space junk[1].
The CubeSat communication system is composed primarily of the telemetry and command systems, which send and receive data, respectively. Analog and digital data collected by the sensors and payload of the satellite must be relayed to the ground station via the telemetry system, which is composed of a transmitter that acts much like a “modem in a computer”[2].

2.2 About CubeSat
2.2.1 CubeSat Definition
The CubeSat Project was a project started by a partnership between the California Polytechnique University in San Luis Obispo and Stanford University in Palo Alto to develop a standardized space platform for academic satellite projects.
Compared to traditional multi-million-dollar satellite missions, CubeSat projects have the potential to educate the participants and implement successful and useful missions in science and industry at much lower costs[3].

2.2.2 Standards And Usage
A standard CubeSat is a 10 cm cube with a mass of up to 1 kg (Figure 2.1). Developers benefit from the sharing of information within the community. Resources are available by communicating directly with other developers and attending CubeSat workshops[3].
With this many institutes participating in the CubeSat program, the educational benefits are tremendous. Students, through hands on work, will develop the necessary skills and experience needed to succeed in industry after graduation. The CubeSat
program also benefits private firms and government by providing a low-cost way of flying payloads in space. All while creating important educational opportunities for future leaders of industry[3].

CubeSat missions still require considerable planning and many man-hours of work to maximize the chances for success. By employing as much standardization as possible you can concentrate on your project’s mission-specific goals.

![Standard CubeSat](image)

**Figure 2.1 Standard CubeSat**

### 2.2.3 CubeSat History

It was a joint project - begun in 1999 - of Cal Poly and Stanford University’s Space Systems Development Laboratory that developed standards for the design of picosatellites. As a result, picosatellites built by students at universities worldwide are launched using a common deployer known as a “P-POD,” developed at Cal Poly.

The term “CubeSat” was coined to denote nano-satellites that adhere to the standards described in the CubeSat design specifications which were laid out by aerospace engineering professor Jordi Puig-Suari and Bob Twiggs, from the Department of Aeronautics & Astronautics at Stanford University[4].

### 2.3 Previous Software Evolution

#### 2.3.1 AO-7 Telemetry RTTY decoder

Overall sent 34 telemetry channels, with some channels (+ X, -X, + Y, -Y Current, + / - repeat Z Axis Sensors and Battery Voltage The input is an ASCII file required are
evaluated only complete lines (. . . 60 characters per line). Be displayed graphically and stored in the decoded data telemetry in an ASCII file, decoder Figure 2.2

![Decoder Figure 2.2 AO-7 Telemetry RTTY decoder](image)

**Figure 2.2 AO-7 Telemetry RTTY decoder**

### 2.3.2 CUTE –I CW Telemetry Decoder Software

The nanosatellite CUTE-I [NKS+03] has been developed by the Laboratory for Space Systems of the Tokyo Institute of Technology and was launched into orbit by a Russian Rokot rocket together with other micro- and nanosatellites on June 30th, 2003 from Plesetsk. It was developed as part of the international CubeSat-project proposed by Stanford University in November 1999.

CUTE-I (Cubical Titech Engineering satellite) was built by using COTS components only in order to reduce total development costs. CUTE-I works for CW, FM Packet telemetry[5]. Decoder Figure 2.3

![Decoder Figure 2.3 CUTE-I CW Telemetry Decoder](image)
2.3.3 O/OREOS Telemetry Decoder

The data portion of a standard AX.25 packet is 64 bytes long, although only the first 62 bytes are considered for the O/OREOS packet. The beacon packet contains only standard ASCII characters. Each character represents a HEX value. Decoder in Figure 2.4

The values in each field are coded in a ‘Little Endian by Pairs’ fashion, also known as Middle Endian. In this coding, each pair of ASCII characters form a byte and those bytes are written from MSB to LSB[6].

2.3.4 HITSAT Simple CW Telemetry Decoder

It is a satellite of the developed amateur radio according to the Hokkaido Cubsat development ham club (The representative: Hokkaido Institute of Technology information design subject assistant professor Mitsuhashi Ryuichi). As for HIT-SAT, it was launched as a sub-payload in solar observation satellite "SOLAR-B" with the seventh M-V rocket machine of JAXA on September 23, 2006.[7] Decoder Figure 2.5
The COMPASS-1 CubeSat was designed and built by students from the Aachen University of Applied Sciences in Aachen, Germany. More than four years were needed to realize this 1 kilogram Picosatellite from scratch into a space-ready flight model. The launch took place in April 2008 from the Indian space port Sriharikota. The mission received (and still receives) tremendous support by the radio amateur community, which helped to collect a large amount of data and images from the satellite.

The received string (26 hex values) is entered into the text box and the corresponding telemetry values are clearly displayed. [8] Decoder shown in Figure 2.6

Figure 2.5 HITSAT Simple CW Telemetry Decoder

Figure 2.6 COMPASS-1 CW Telemetry Decoder
2.4 CubeSat Communication System

The primary goal of the communications subsystem is to provide a link to relay data to ground station and send commands to and from the CubeSat and as such will primarily consist of telemetry and command sequences that send and receive data respectively. Data is collected by the payload and sensors on the satellite in the form of analogue or digital data which must be sent to the ground station[9].

2.4.1 Ground Station

A ground station is basically an earth based point of communication with the space segment. They are our source of interaction with the satellite; hence play an important part for any satellite related operation and it is very important to have a very good communication link between the ground station and the satellite/space segment[10]. Usually a ground segment/ground system involves following tasks:

- Tracking and determine the position of satellite orbit.
- Telemetry operation to acquire and record satellite data and status.
- Controlling operation to determine orbital parameters, to schedule all satellite passes and to monitor and load the on-board computer.
- Data processing operations to present all the engineering and the scientific data in formats required for the successful progress of the mission.
- Voice and data links to the other worldwide ground station and processing centres.

2.4.2 Amateur Radio

(Also, ham radio) is the licensed and private use of designated radio bands, for purposes of private recreation, non-commercial exchange of messages, experimentation, self-training, and emergency communication.

Amateur radio, like other regulated radio services, operates under rules that limit the maximum power and the technical and operational characteristics of transmissions. Amateur radio stations are issued with a designated call sign to allow identification of stations.

The power of amateur radio equipment is restricted, and operators must not cause interference to other authorized radio users. They may not broadcast to or
communicate with the public with their equipment. They are only allowed to communicate with other licensed operators.

National regulations governing amateur radio use are coordinated under international agreements since radio frequency transmissions can cross multiple national boundaries[11].

### 2.4.3 CubeSat Communication Link Types

Communication with the spacecraft takes place over two types of links:

1. The uplink: carries commands from a ground station to the spacecraft.
2. The downlink carries data, which consists of two different types of information:
   a) Payload data is optional to the mission specified by the CubeSat construction.
   b) Housekeeping data: is information about the spacecraft's vital characteristics.

### 2.4.4 Limited Communication Window

The geometry of a satellite's orbit dictates a schedule of when, and for how long, the satellite is able to communicate with a fixed ground station. Cubesats are typically launched in what is called a low-earth orbit (LEO). Low earth orbits are characterized by their short range, high orbital velocity and non-geosynchronous nature, Table 2.1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (US)</th>
<th>Value (metric)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elevation</td>
<td>400-435 mi</td>
<td>650-700 km</td>
</tr>
<tr>
<td>Orbital Velocity</td>
<td>≈17,000 mi/h</td>
<td>≈27,000 km/h</td>
</tr>
</tbody>
</table>

The communication window for a satellite is the amount of time that a fixed ground command station can transmit to and receive signals from a satellite. The duration of this window is determined by the orbital parameters, and is defined as the length of time between AOS (acquisition of signal) and LOS (loss of signal).[12] Figure 2.7
2.5 Communication Protocols

2.5.1 Morse Code

Morse code is a method of sending text messages by keying in a series of electronic pulses, usually represented as a short pulse (called a "dot") and a long pulse (a "dash"). Figure 2.8 and Figure 2.9

Morse code offers a slow but reliable means of transmitting and receiving wireless text messages through conditions involving noise, fading, or interference. This is primarily because it's simple binary code (key down or key up) allows for an extremely narrow bandwidth. In addition, the human ear and brain make a remarkably good digital receiving device. Nowadays, Morse code is used to a limited extent by amateur radio operators, landline telegraphers, and military radio operators[13].

```
A .-  J .---  S ...  2 ..---
B ---.  K .-..  T -  3 ...-
C --.  L -. .  U ..-  4 .---
D -.  M .-..  V ..  5 ---
E .  N -  W .--  6 -.--
F ---  O ..-  X --.  7 --
G --.  P .--.  Y -. -  8 -.--
H ....  Q .--.  Z --.  9 -.--
I .  R .-.
```

Figure 2.8 Morce Code Table
2.5.2 Continuous Wave

A continuous wave or continuous waveform (CW) is an electromagnetic wave of constant amplitude and frequency; and in mathematical analysis, of infinite duration. Continuous wave is also the name given to an early method of radio transmission, in which a carrier wave is switched on and off. Information is carried in the varying duration of the on and off periods of the signal, for example by Morse code in early radio. In early wireless telegraphy radio transmission, CW waves were also known as "undamped waves", to distinguish this method from damped wave transmission. Continuous-wave radio was called radiotelegraphy because like the telegraph, it worked by means of a simple switch to transmit Morse code. However, instead of controlling the electricity in a cross-country wire, the switch controlled the power sent to a radio transmitter. This mode is still in common use by amateur radio operators[14].

2.5.3 AX.25 Protocol

AX.25 is an amateur radio specification which describes how to encode digital data in order to transmit it over radio frequencies. The AX.25 specification mandates a bit rate of 1200 baud and uses AFSK (audio frequency shift keying) encoding to represent binary values 0 and 1 with audio tones of 1200 Hz and 2200 Hz, respectively.
AX.25 is a full featured, stateful communications protocol that has been used for many years by ham radio enthusiasts worldwide.

Frequency-shift keying (FSK) is another method of transmitting digital signals. The two binary states, logic 0 (low) and 1 (high), are each represented by an analog waveform. Logic 0 is represented by a wave at a specific frequency, and logic 1 is represented by a wave at a different frequency. A modem converts the binary data from a computer to FSK for transmission over telephone lines, cables, optical fiber, or wireless media. The modem also converts incoming FSK signals to digital low and high states, which the device can process[15].

2.6 Cubesat Telemetry

Telemetry is a technology that allows remote measurement and reporting of information. The word is derived from Greek roots tele = remote, and metron = measure. Systems that need external instructions and data to operate require the counterpart of telemetry, telecommand.

Although the term commonly refers to wireless data transfer mechanisms (e.g. using radio or infrared systems), it also encompasses data transferred over other media, such as a telephone or computer network, optical link or other wired communications[16].

2.6.1 Telemetry History

Telemetering information over wire had its origins in the 19th century. One of the first data transmission circuits was developed in 1845 between the Russian Tsar's Winter Palace and the army's headquarters. In 1874, French engineers built a system of weather and snow-depth sensors on Mont Blanc that transmitted real-time information to Paris. In 1901 the American inventor C. Michalke patented the selsyn, a circuit for sending synchronized rotation information over distances. In 1906, a set of seismic stations were built with telemetering to the Pulkovo Observatory in Russia. In 1912, Commonwealth Edison developed a system of telemetry to monitor electrical loads on its power grid. The Panama Canal (completed 1913-1914) used extensive telemetry systems to monitor locks and water levels.

Wireless telemetry made early appearances in the radiosonde developed concurrently in 1930 by Robert Bureau in France and Pavel Molchanov in Russia. Molchanov's system modulated temperature and pressure measurements by converting them into
wireless Morse code. The German V-2 rocket used a system of primitive multiplexed radio signals called "Messina" to report 4 rocket parameters, but it was so unreliable that Von Braun once claimed it was more useful to watch the rocket through binoculars. In both the USA and USSR, the Messina system was quickly replaced with better systems, in both cases based on pulse-position modulation.

Early Soviet missile and space telemetry systems developed in the late 1940s used either pulse-position modulation (e.g., the Tral telemetry system developed by OKB-MEI) or pulse-duration modulation (e.g., the RTS-5 system developed by NII-885).

2.6.2 Tracking, Telemetry And Command (TT&C)

The TT&C subsystem performs several routine functions aboard the spacecraft. The telemetry, or telemetering, function could be interpreted as measurement at a distance. Specifically, it refers to the overall operation of generating an electrical signal proportional to the quantity being measured and encoding and transmitting this to a distant station, which for the satellite is one of the earth stations. Data which are transmitted as telemetry signals include attitude information such as that obtained from sun and earth sensors; environmental information such as the magnetic field intensity and direction, the frequency of meteorite impact, and so on; and spacecraft information such as temperatures, power supply voltages, and stored-fuel pressure. Certain frequencies have been designated by international agreement for satellite telemetry transmissions.

Telemetry and command may be thought of as complementary functions. The telemetry subsystem transmits information about the satellite to the earth station, while the command subsystem receives command signals from the earth station, often in response to telemetered information.

The command subsystem demodulates and, if necessary, decodes the command signals and routes these to the appropriate equipment needed to execute the necessary action. Thus attitude changes may be made, communication transponders switched in and out of circuits, antennas redirected, and station-keeping manoeuvres carried out on command. It is clearly important to prevent unauthorized commands from being received and decoded, and for this reason, the command signals are often encrypted.
It is clear that the telemetry, tracking, and command functions are complex operations which require special ground facilities in addition to the TT&C subsystems aboard the satellite[17].

2.6.3 Telemetry Data

The CubeSat’s house-keeping telemetry is comprised of readings from various sensors (voltage, current and thermal sensors) housed on the satellite that will be transmitted to the ground station. At the ground station the house-keeping telemetry will be compared with those readings taken during environmental and functionality testing of the satellite prior to launch.

2.6.4 Command And Data Handling (C&DH)

The Command and Data Handling Subsystem is the ‘brain’ of the whole autonomous CubeSat. The C&DH system consists of an Onboard Computer, OBC, which controls the operation of the CubeSat. The OBC has software installed that manages the programs written to handle various tasks; for example, a program whose function is to create a telemetry stream will read the status of the payload sensors and then encode the telemetry stream. The same program can further control the flow of the data from sensors to the temporary memories inside the microcontroller in the event of communication restrictions, such as the blocking of communication signals between the CubeSat and the ground station.

Under normal conditions, the CubeSat’s Nominal Orbital Modes will perform the basic day-to-day activities.

The primary state of these Orbital Modes will be the Beaconing mode. When in Beaconing mode, C&DH will perform a series of fundamental orbital tasks to ensure the CubeSat is running optimally. A list of these fundamental tasks can be seen below:

a) Ensure the antenna is deployed and check for command to deploy gravity gradient boom
b) Take sensor data at designated orbital intervals from the voltage, current and thermal sensors
c) Manage the storage of house-keeping data
d) Perform nominal communication through beaconing
e) Monitor telecommunications unit for inbound telecommand from ground system to switch over to telemetry mode and begin transferring data
f) Monitor the battery charger and transition to Recharge mode shall the battery capacity shall fall below 60%

The CubeSat will release short, periodic Morse code beacons which will serve as an identifier, a way to track the CubeSat, and a way to provide the ground system with some critical house-keeping data or space telemetry should telecommunication links with the ground station become unattainable. Space telemetry is a one-way transmission from a space station of measurements made from the measuring instruments in a spacecraft, including those relating to the functioning of the spacecraft. Upon the completion of the fundamental orbital tasks, the system will go into a sleep state, in an attempt to conserve power until it’s time to complete another set of tasks[2].

2.7 Encoding and Decoding

One reason for coding is to enable communication in places where ordinary spoken or written language is difficult or impossible. Encoding and decoding are used in data communications, networking, and storage. The term is especially applicable to radio (wireless) communications systems[18].

2.7.1 Encoding

In communications and information processing, encoding is the process by which information from a source is converted into symbols to be communicated. Encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. Encoding should not be confused with encryption, a process in which data is deliberately altered so as to conceal its content. Encryption can be done without changing the particular code that the content is in, and encoding can be done without deliberately concealing the content[18].
2.7.2 Decoding

Decoding is the reverse process, converting these code symbols back into information understandable by a receiver.

Decoding is the conversion of an encoded format back into the original sequence of characters[18].

2.7.3 CubeSat’s Data Format

2.7.3.1 COMPASS-1 CW Telemetry Decoder Format

COMPASS-1 CW Telemetry Decoder as mentioned in 2.3.5 data format Table 2.2 [8]:

COMPASS AABBBBBBBBBCDEFFGHHIIJ

Table 2.2 COMPASS-1 CW Telemetry Decoder Format

<table>
<thead>
<tr>
<th>AA</th>
<th>Solar Cells Voltage</th>
<th>N * 5/255 [Volts]</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBBB BBBB</td>
<td>Solar Panel 1..5 Current</td>
<td>N * 1.6/255*1000 [mA]</td>
</tr>
<tr>
<td>CC</td>
<td>EPS reset counter</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Power Level</td>
<td></td>
</tr>
<tr>
<td>0 = Battery Capacity OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Low Battery Capacity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 = Critical Battery Capacity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 = Battery Charging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Heater Active</td>
<td></td>
</tr>
<tr>
<td>0 = Heater OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Heater ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>Powersafe Counter</td>
<td></td>
</tr>
<tr>
<td>GG</td>
<td>Emergency Mode Counter</td>
<td></td>
</tr>
<tr>
<td>HH</td>
<td>Battery Voltage</td>
<td>N * 5/255 [Volts]</td>
</tr>
<tr>
<td>II</td>
<td>Battery Current</td>
<td>N * 1.6/255*1000 [mA]</td>
</tr>
<tr>
<td>JJ</td>
<td>Battery Temperature</td>
<td>[C]</td>
</tr>
</tbody>
</table>

2.7.3.2 O/OREOS Telemetry Decoder Format

As mentioned in section 2.3.3, The data portion of a standard AX.25 packet is 64 bytes long, although only the first 62 bytes are considered for the O/OREOS packet. The beacon packet contains only standard ASCII characters. Each character represents a HEX value. Below is an example of the 64-byte long raw data:

OOREOS.org 086501010400B2020AB202A201210004020D5F094300204B46

(note: in this packet, the last two bytes, 46, will not be considered for parsing)
Depending on the TNC being used, the data string could be preceded by the following set of characters:

KF6JBP>UNDEF,TELEM/1: <<UI>>:

which denotes sender and recipient of the packet. Ignore this when decoding the packet. Table 2.3

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (bytes)</th>
<th>Offset (bytes)</th>
<th>Description</th>
<th>Valid for Packet Type</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Website</td>
<td>30</td>
<td>0</td>
<td>O/OREOS seq</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>4</td>
<td>10</td>
<td>Unassigned</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>BusTime</td>
<td>6</td>
<td>14</td>
<td>BusTime</td>
<td>All</td>
<td>Seconds</td>
</tr>
<tr>
<td>PacketType</td>
<td>2</td>
<td>20</td>
<td>Packet type (0,1,2, or 3)</td>
<td>All</td>
<td>Integer</td>
</tr>
<tr>
<td>Solar1</td>
<td>4</td>
<td>22</td>
<td>Solar Panel 1 Electrical Current</td>
<td>ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar Panel 2 Electrical Current</td>
<td>ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar Panel 3 Electrical Current</td>
<td>ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar Panel 4 Electrical Current</td>
<td>ADC beams</td>
<td></td>
</tr>
<tr>
<td>Solar2</td>
<td>4</td>
<td>26</td>
<td>Solar Panel 1 Temperature</td>
<td>Celsius degrees C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar Panel 2 Temperature</td>
<td>Celsius degrees C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar Panel 3 Temperature</td>
<td>Celsius degrees C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar Panel 4 Temperature</td>
<td>Celsius degrees C</td>
<td></td>
</tr>
<tr>
<td>Health0</td>
<td>2</td>
<td>30</td>
<td>Bus Power Fault Status</td>
<td>Bit Field</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Startup Counter</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>spacecraft to Ground ID</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Health1</td>
<td>4</td>
<td>32</td>
<td>FLT Current Phase</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FLT Temperature</td>
<td>Celsius degrees C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MinO electrical current</td>
<td>ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MinO Voltage</td>
<td>ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Battery Voltage</td>
<td>0 ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Common Voltage</td>
<td>1 ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sensors Voltage</td>
<td>2 ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bus Voltage</td>
<td>3 ADC beams</td>
<td></td>
</tr>
<tr>
<td>Health2</td>
<td>4</td>
<td>38</td>
<td>ProviderName (bytes 2-3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BatteryChargeRate (bytes 0-1)</td>
<td>see section 4 for more information</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Payload Electrical Current</td>
<td>1 ADC beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bus Data Page</td>
<td>2 Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Register File Wrap Count</td>
<td>3 Integer</td>
<td></td>
</tr>
<tr>
<td>Health3</td>
<td>4</td>
<td>40</td>
<td>FLT3 Health Data Page</td>
<td>0 Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FLT3 Science Data Page</td>
<td>1 Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FLT2 Health Data Page</td>
<td>2 Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FLT2 Science Data Page</td>
<td>3 Integer</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>4</td>
<td>44</td>
<td>Unassigned</td>
<td>All</td>
<td></td>
</tr>
</tbody>
</table>

| Total Bytes | 62          |                |                                                  |                       |          |

2.7.3.3 **Cute-1 - CW Telemetry Format**

This frame consists of housekeeping data of the satellite.[5]
Table 2.4 Cute-1 - CW Telemetry Format

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>bit</th>
<th>letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA Command</td>
<td>DTMF</td>
<td>24bit</td>
<td>6</td>
</tr>
<tr>
<td>BB  V3.3</td>
<td>3.3V</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>CC  V5</td>
<td>5V</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>DD  VBatt</td>
<td>Battery Voltage</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>EE  VMain</td>
<td>Main Bus Voltage</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>FF  SatStatus</td>
<td>Satellite Status</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>GG  TComm</td>
<td>Temperature of Comm Board</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>HH  TBatt</td>
<td>Temperature of Battery</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>II  Current</td>
<td>Battery Current</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>JJ  S144</td>
<td>144MHz Smeter</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>KK  S1200</td>
<td>1.2GHz Smeter</td>
<td>8bit</td>
<td>2</td>
</tr>
<tr>
<td>LL  FETStatus</td>
<td>FET status</td>
<td>8bit</td>
<td>2</td>
</tr>
</tbody>
</table>

2.7.3.4 HITSAT Simple CW Telemetry Decoder

As mentioned in 2.3.4, the decoder’s data format [7]

Format Description of HIT-SAT CW. Table 2.5

Table 2.5 HIT-SAT CW Format

<table>
<thead>
<tr>
<th>HIT1</th>
<th>Callsign</th>
<th>JR8YJT</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIT4</td>
<td>Power source</td>
<td>[OO]Primary battery,[PP]Secondary battery,[QQ]5V,[RR]5.5V</td>
</tr>
<tr>
<td>HIT5</td>
<td>Amateur Radio Service</td>
<td>TNX[Callsign1],TNX[CallSign2],</td>
</tr>
</tbody>
</table>

HIT1@@Callsign

18
This is JR8YJT.

**HIT2® HIT-SAT time and RSSI(Receive Signal Strength Indication)**

[A]: Month of hex number. If it is 1, it is January. If it is A, it is October.

[BB]: Day. 01 to 31.

[CC]: Hour. 00 to 23.

[DD]: Minute. 00 to 59.

[EE]: Second. 00 to 59.

[FF]: AD conversion value of RSSI.

Data is given in hexadecimal. Change the hexadecimal number to the decimal number, and substitute the X into the following equation.

\[
\text{RSSI[V]} = X \times \frac{5}{256}
\]

**HIT3® Temperature Data**

[GG][HH][II][JJ][KK][LL][MM][NN]: AD conversion value of Temperature sensor.

Data is given in hexadecimal. Change the hexadecimal number to the decimal number, and substitute the Y into the following equation.

\[
\text{TEMP[C]} = (Y \times \frac{5}{256}) \times (-122.99) + 214.94
\]

Temperature Data with an error margin of 20[C]

**HIT4® Power source Data**

[OO][PP]: AD conversion value of Battery.

Data is given in hexadecimal. Change the hexadecimal number to the decimal number, and substitute the X into the following equation.

\[
E[V] = X \times \frac{10}{256}
\]

[QQ][RR]: It is 5V and 5.5V supplied in the satellite. AD conversion value of Voltage.

Data is given in hexadecimal. Change the hexadecimal number to the decimal number, and substitute the Y into the following equation.

\[
E[V] = X \times \frac{5 \times 1.21276}{256}
\]

### 3.7.4 Sound Decoders

Other software should be used to convert the sound signal received to a text format, some software are concerned with that such as:
MixW
The MixW can be used to decode the received signal from either CW or Packet to text in real time decoding. Decoder is shown in Figure 2.10.

![MixW sound decoder](image)

Figure 2.10 MixW sound decoder

MULTIPSK
Another sound decoder is used is the MULTIPSK to decode the received sound signal into text. The decoder is shown in Figure 2.11.
Figure 2.11 MULTIPSK sound decoder
CHAPTER THREE : CUBESAT TELEMETRY DECODER SOFTWARE IMPLEMENTATION

3.1 Introduction

This chapter presents the design and implementation of the objectives stated in chapter 1 section 1.4.

The requirements, alternatives assumptions for the system are stated to be considered in the design of the system.

In order to implement the software system, Object Oriented Programming methods are used. The system consist of four classes that are implemented to accomplish the system. An easy graphical user interface is designed to be clear to the user with addition to a help tool for the user. More features are added to the system to have a complete telemetry decoder software.

3.2 System Requirements

3.2.1 Functional Requirements

The main objective of this project is to develop a software system that can be used to read a beacon data that is saved in a text file as a KNSAT data format and decode it to its values according to known tables of values, the system should be able to detect out of range values and alarm the user for each beacon.

By making the system achieve the objectives stated in chapter 1 section 1.4 will then be achieved as consequence. The system must perform the following:

- Read the file chosen by the user that should be a text file containing the beacon format data.
- Check that the chosen file is a text file and that it contains the beacon data in the specified format.
- Decode each beacon in the file and assign each value of it to the equivalent value.
- Display the decoded value in a friendly user interface that is easily used to go through many beacons data at the same time.
Provide graphic charts that shows the variation of some values over time.
Save the decoded values in a local database or in separate file as specified by
the user.
Detect danger values of CubeSat component and alarm the user for higher
from the normal limit values and lower from the normal limit.
The user can change sitting of the software as the user specify.

3.2.2 Non-functional Requirements

Performance, the program has no delays in performing methods.
Maintainability, the program is easily maintained and the source code is
readable.
Reliability, decoding is done with less errors as possible.
Ease of modification, any future change in the data format should be easily
modified in the source code.

3.3 Assumptions

The telemetry decoder software is a subsystem from the ground station software, so
some assumptions and consideration should be taken about the whole system.
These considerations are mentioned as follow:

The data is assumed to be received by antenna and converted to text using
some other software such as MixW, MULTIPSK or any other software.
The received data is saved in a text file.
Each line in the text file is a beacon and begins with the callsign KNSAT
The beacon consist of 5 parts separated by space between them.
The value of each field is encoded by one character.

3.4 Implementation Tools

Some java tools where used to implement the telemetry decoder software:

- Java Netbeans, which is the text editor used to write the program and compile
  (downloaded from [19]).
- Java Run time Environment, the java plat form that java JAR file run on
  (downloaded from [20]).
- JDK (Java Development Kit), To develop Java applications and
  applets(downloaded from [21]).
• JFreeChart class, open-source framework for the programming language Java, which allows the creation of complex charts in a simple way (downloaded from [22]).

3.5 Data Format

3.5.1 Alternatives

Based on the experience of other Cubesat projects (most notably, CUTE-1) it was determined that CubeSats with audible beacons were much easier to locate and contact after deployment.

As mentioned in section 2.7.3, there many ways to construct the data format, the main rule is to save power, there is no standard for the format of the beacon.

From the observation of the other CubeSats. The best format is that each field is divided into range of values, the CubeSat with chose the value from a look up table rather than sending the actual measured value.

This method has the following advantages:

➢ The values in space is not necessary to be exact, and an approximate value is Adequate.
➢ Using look up table is the faster method in calculations, the CubeSat processor is not compelled to convert the value to hexadecimal or using any calculations.
➢ By reducing the process the power consumption is reduced and more power will be saved.
➢ For future changes it is easier and faster to modify the transmitting software and the telemetry decoder software to fit any changes.
➢ Easy modifications can be done to use this software for any other CubeSat that uses this format.

3.5.2 KNSAT Data Format

The KNSAT beacon data format is shown in Table 3.1:

<table>
<thead>
<tr>
<th>Table 3.1 KNSAT Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN0</td>
</tr>
<tr>
<td>KNSAT</td>
</tr>
</tbody>
</table>
Each beacon data consist of 5 parts,

- First, the call sign KNSAT,
- Second [AAAAA] The CubeSat time.
- Third [BCDEF] The statuses of some devices that contains:
  - [B] The Cubesat mode that illustrate the mode of the CubeSat that is changed with time.
  - [C] The payload request that,
  - [D] The payload status,
  - [E] OBC status
  - [F] The transmitter device status.
- Forth [GHIJ] The voltages that contains:
  - [G] Battery voltage
  - [H] Battery Temperature
  - [I] Bus 3 Voltage
  - [J] Bus 5 Voltage
- Fifth [KLMNOPQRS] The Temperature cells and other devices that contains:
  - [K] Payload Temperature
  - [L] Antenna Temperature
  - [M] OBC Temperature
  - [N] Temperature of +X Panel
  - [O] Temperature of –X Panel
  - [P] Temperature of +Y Panel
  - [Q] Temperature of –Y Panel
  - [R] Temperature of +Z Panel
  - [S] Temperature of –Z Panel

Buses’ Currents and its powers are calculated from their voltages and known load for it.

List of look up table are shown in Appendix A.
3.6 Java As Object Oriented Programming OOP

3.6.1 The need of OOP

3.6.1.1 OOP Definition
Object Oriented Programming or OOP is the technique to create programs based on the real world. Unlike procedural programming, here in the OOP programming model programs are organized around objects and data rather than actions and logic. Objects represent some concepts or things and like any other objects in the real world, objects in programming language have certain behavior, properties, type, and identity. In OOP based language the principal aim is to find out the objects to manipulate and their relation between each other. OOP offers greater flexibility and compatibility and is popular in developing larger application. Another important work in OOP is to classify objects into different types according to their properties and behavior. So OOP based software application development includes the analysis of the problem, preparing a solution, coding and finally its maintenance.

Java is a object oriented programming and to understand the functionality of OOP in Java, we first need to understand several fundamentals related to objects.

3.6.1.2 OOP Principles

Class - It is the central point of OOP and that contains data and codes with behavior. In Java everything happens within class and it describes a set of objects with common behavior. The class definition describes all the properties, behavior, and identity of objects present within that class. As far as types of classes are concerned, there are predefined classes in languages like C++ and Pascal. But in Java one can define his/her own types with data and code.

Object - Objects are the basic unit of object orientation with behavior, identity. As we mentioned above, these are part of a class but are not the same. An object is expressed by the variable and methods within the objects. Again these variables and methods are distinguished from each other as instant variables, instant methods and class variable and class methods.

Methods - We know that a class can define both attributes and behaviors. Again attributes are defined by variables and behaviors are represented by methods. In other words, methods define the abilities of an object.[23]
3.6.2 OOP In Telemetry decoder Software

The telemetry decoder software is an object oriented programming, the usage of OOP gives the ability of declaring many objects of the same entity with different values. This feature of OOP is used in the telemetry decoder to allow the user to open more than text file at the same with each containing many beacons data. With this ability it make it easy for the user to open and compare beacons data from different files received at different duration time without closing the opened file each time.

The separation of objects in classes make the use of applying methods on and process without affecting other objects.

The java is the best programming language for dealing with String data type, as the main element in the telemetry decoder is the beacon String types. Also it considered a secure programming language because of the encapsulation of the objects.

The most feature in Java is the General-purpose programs that the built program can run in any machine that installs the Java-Run time – Environment, independent on the operating system and the hardware.

3.7 Development Plan

The methods were implemented by the plan of V-Model.

The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

The whole system was implemented by the plan of Agile development. Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. The development plan is shown in Figure 3.1.
The CubeSat telemetry decoder software was implemented by the Java programming language. The software composed of four classes: First, The main class that runs the GUI part. Second, The pack class that represents the files objects and contains the beacon objects, this class deals with the whole beacon file and construct chats and saves decoded data in another text file. Third, the convert class which convert each data to its corresponding value. Fourth, the option class that the user can change its settings.

The relation between classes is shown in the class diagram in Figure 3.2 section 3.11.1.

3.8 NewJFram Class

The main class that implement the main function where the program start. At beginning of the program, the main class initialize the GUI components by initComponents() function by declaring it and adding it to the frame.

After the frame be initialized the user imports the text file that is wanted to be decoded. The file is read from its direction and should be checked by the checked() function to insure that it is text file and contains the beacon data that begins with
KNSAT. If it is a wrong file, the program will reject it. Otherwise, an object of the class pack will be defined for that file by the `newpack()`.

The Frame class then displays the decoded data by `displayValues()` function and display red alarm by `alarm()` for out of range values. As the program has the ability to change the temperature unit, so the frame class has the `displayUnit()` function to display the chosen temperature unit.

Also it contains bottoms for displaying the next by `next()` function and previous beacons by `previous()` function and the next and previous files by `nextFile()` and `previousFile()` functions respectively.

This class also contains an easy help tool bar that displays a help tip when mouse places over a component.

The class also contains the charts buttons that calls the `drawChart()` function. The frame has many themes that the user choose, the frame class change the display the chosen theme by the `theme()` function.

The unwanted file can be closed by the `close()` function to close an opened file without closing the program.

The Frame class should manage the array of files so no array index out of bound exceptions occurs.

### 3.8.2 Pack Class

The `pack` class contains the `beacons` of each file and returns `beacons` to the `Frame` to display it.

When the `Frame` class declare a new file, the `pack object` will read each line in the text file and create a `convert beacon object` for each `beacon`, the `beacon object` is an array of `convert class`.

When the user ask to draw a chart, This class will get the specified element from the Frame class and the area to display the chart from the options class. The pack class creates the database of the chosen element by the `dabaseElement()` function that each element has its own function, then construct the chart by the `drawChart()` function.

The `dabaseElement()` function represents the functions of some elements such as the voltages, currents and powers of the battery, also the temperature of some components such as the 6 Panels cells, the OBC, antenna, payload and the battery temperature.

The `drawChart()` function will display the chart in separate window or returns the constructed panel to the `Frame class` to display it in the main frame.
As this represents a file object, the `save()` and `saveAs()` functions are implemented in it. The `save()` function saves the decoded values in a fixed text file as a local database file. For each save request from the user, the `save()` function will add the decoded beacons to the same file. The number of beacons added to the file are preserved in a variable in the `Frame class` and passed to the `save()` function to continue from it and return the last position to the `Frame class`.

On the other hand, the `saveAs()` function asks the user to choose a directory to save the text file in it and a file name. The function creates a text file in the specified directory with the entered name and adds the header label of the database file, then the decoded beacons are added to this text file by numbering begins by one till the number of beacons.

This class also manages the beacon returned to the `Frame class` to be displayed to the user. Whenever the user asks for the next or previous beacon, the `pack class` will return the wanted `beacon object`. Also when the user moves from file to file, the class should return the same object where the user moved from.

Moreover, The `Pack class` deals with `convert objects` and implements functions that changes and gets values from it.

### 3.8.3 Convert Class

This class represents the `beacon objects`, and it is created from the `pack class` Each `beacon object`.

In this class, variables of the beacon’s elements are declared and values are given. Also the alarm variable of each element is determined in this class and the `Frame class` gets this variable to display the alarm.

The `beacon object` reads the line and divides it to segments then, reads the beacon’s characters of segments using function for each segment `toKN()` and convert each one according to its corresponding value by using `toValues()` functions by the beacon format as mentioned in section 3.5.2. Each variable has its own function to convert values.

In order to return values to `pack class` and then to `Frame class`, the variables should be converted to text by using a function `getValue()` for each element to be displayed.
In addition, the calculations of temperature conversion unit is done by \texttt{changeTemp()} function to change temperature values from Fahrenheit to Celsius and vice versa.

The function \texttt{toString()} is used to return the text format of the whole beacon to the \textit{pack class} to be used in \texttt{save()} or \texttt{saveAs()} functions.

This class deals with the low level data structure and manage the detailed values.

### 3.8.4 Options Class

Some options are available to the user to change its settings.

These changes are performed in the \textit{options class} to be returned to other classes for changes.

\textit{Option class} preserves the change of option of temperature unit in a variable and it is changed by \texttt{setTempUnit()} function. \textit{Beacon objects} get this value by \texttt{getTempUnit()} function to convert the temperature values.

Also, when the user changes the theme color, new colors should be preserved and returned to \textit{Frame class} in \textit{option class} by using \texttt{setColor()} and \texttt{getColor()} functions.

Furthermore, the option of displaying the charts are preserved in this class, the chart can be either displayed in the same main frame or in a separate external frame. This options should be returned to \textit{pack class} by \texttt{setChartDisplay()} and \texttt{getChartDisplay()} functions.

### 3.8.5 JFreeChart External Class

\textit{JFreeChart} is an open-source framework for the programming language Java, which allows the creation of complex charts in a simple way.

\textit{JFreeChart} also works with GNU Class path, a free software implementation of the standard class library for the Java programming language.

This class should be linked to the program to be included in classes and perform the chart methods.

### 3.9 Methods

#### 3.9.1 Decoding Method

The essential method in the program and the most important is the decoding method, the user presses the import button and chooses the text file that contains the beacon data.
The program reads the text file to check if it is the beacon file and creates a *pack object* for that file, the number of *pack objects* is incremented in the pack array.

The new *pack object* contains the array of *beacon objects*. The *pack object* reads the file line by line and creates a beacon object for each line, the number of beacons is the size of the array it should be known.

For each *beacon object*, the beacon divides the string into segments then convert each segment by reading its characters and convert each character by its function using the specific format shown in section 3.5.2 and save each value in its variable.

The bus current and power are calculated from a known load value.

After that, the alarm is checked by comparing the values with a safe values, the alarm variables are marked in out of range variables.

Then the temperature unit is get from the *options class* and according to it the temperature is converted to Fahrenheit or not.

The beacon object returns its values to the *Frame class* through the *pack class* using getters function. The *Frame class* displays the values of the current beacon then displays the alarm and the temperature unit.

### 3.9.2 Chart Method

The user requests the *Frame class* to draw a chart, the frame class determines the pack object that the user wants to draw chart for it to get its chart panel. The pack class adds the value of the specified element to a database, this database is used in the draw chart function. The function gets the details of the chart that determines the X-axis, Y-axis, the chart title and the chart type. After the chart is constructed, the color of the bars is set.

The display area is checked from the options class either in the same frame or in external frame. If the option is set to the same frame then the function returns the chart panel to the *Frame class* to be displayed.

Otherwise, the function creates a new external frame and displays the chart on it, and returns an empty panel to the *Frame class*.

This method is used to draw chart of some values such as:

- Battery Voltage
- 3.3v Bus Voltage
- 3.3v Bus Current
- 3.3v Power
 5v Bus Voltage
 5v Bus Current
 5v Power
 Cell 1 Temperature
 Cell 2 Temperature
 Cell 3 Temperature
 Cell 4 Temperature
 Cell 5 Temperature
 Cell 6 Temperature
 Payload Temperature
 Antenna Temperature
 OBC Temperature
 Battery Temperature

The chart is shown either in separate window or in a panel in the same frame. This is done by the display method mentioned in section 3.9.4

3.9.3 Save & Save As Methods

The save & save as method are used to save the decoded data to another text file in a format of table with beacons values.

The user requests the Frame class to save or save us the pack object.

The Frame class should first check if the user imported a file or no file is available to be saved. The user should import at least one file to be saved.

The saving method also checks the availability of the file where the data will be saved. For the save() method, the database file is fixed for every time, and the counter is incremented for each beacon added, no need to determine the file location.

On the other hand, the saveAs() function is used to save the decoded data in a new text file, where the user specifies the file direction and the file name. The saveAs() method creates a new text file and inserts the header that contains the labels of the elements in the top of the file.

For all the beacon objects in the pack file, the elements are added to the text file one by one with numbering starts at one to the last beacon object, Then the program closes the text file and notify the user by a message that the data is added.
3.9.4 Display Method

The display method is concerned with the interaction of the user with the data. This method contains many functions to manage the data being displayed to the user. When the user imports a file the display method should display the first beacon object in that file, the display method displays the next beacon object when the user presses the next button, and displays the previous beacon object when the user presses the previous button.

Moreover, the display method displays the next and previous pack object in its beacon object where the user current go through by the next file and previous file buttons. The display method should take care of the boundaries of the array for the beacon objects and the array for the pack objects.

In addition, the display method closes the pack object that the user current displaying when the user presses the close button without closing the whole program, and then displays the previous pack object, the closed pack object should be deleted from the pack objects array and the new boundary is saved.

The display method gets the element wanted to be displayed from the current pack file object to get its current beacon object to be displayed.

The display method also gets the chart panel from the pack class to be displayed in the specified area either in the internal Frame or in external frame, as the user sets the options.

The GUI is provided with an easy help tool bar in the bottom of the window, when the user moves the cursor over a component the display method displays the help tip of it in the help tool bar, and it is erases when the user moves the cursor out.

The display method also displays the alarm red color for the elements that their alarm variables are true.

The program has the option of changing the theme color, the user can choose the theme from choices and the display method deals with displaying the chosen theme. The options class should preserve the color theme to keep the un alarmed elements in the theme color after modification of the alarm, The display method also gets the temperature unit from the option class to change the unit being displayed whenever the user changes its setting, The display method also displays the file’s path being displayed and its order in the pack objects array and the beacon object array.
3.10 User Implementation
As shown in section 3.8 and 3.9, many classes and methods are used to perform the task for the user. The user starts at the start window which contains the KNSAT logo, then the user can moves to the beacon tab to import the beacon text file and then the program performs the decoding method as mentioned in section 3.9.1 and the displaying method as mentioned in section 3.9.4.
The user can import as much as want by performing decoding method and displaying method.
After that, the user can draw charts from the buttons and perform the chart method as mentioned in section 3.9.2. Also the user can save the charts and change its color and scale as the want of the user. The options of the temperature unit can be changed from Fahrenheit to Celsius and vice versa. Also there are three themes available for the user to choose, pink, gray and yellow.
Furthermore, the user can change the area of displaying the charts either in the same frame or in separate window.
For any ambiguous button or tool the user can use the help tool bar to get tips for using the buttons.
The about panel is al shown in the options tab shows some details of the version, the copy right, the operating system, the KNSAT web and the developers.

3.11 Implementation UML Diagrams
Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.
UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

3.11.1 Class Diagram
Describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
As shown is diagram Figure 3.2, the frame class can have many packets files objects or non. And each packet object has a beacon object or more.
In addition, the frame class has one options object that contains the detail of the options of the whole program.
3.11.2 Use Case Diagram

Describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.

The user can choose a beacon text file to be decoded, request drawing a chart of some values or save the decoded values to another text file.

The use case diagram is shown in Figure 3.3

---

1 **Extend** is a use case that shows functions beyond those found in the base case. The execution of one Use Case may enclose the *extending* Use Case.  
**Include** When a use case can be part of multiple use cases (similar to subroutine or sub module), the execution of one Use Case always encloses the *included* Use Case.

---

**Figure 3.2 Telemetry Decoder Class Diagram**
Figure 3.3 Use Uase Diagram

### 3.11.3 Activity Diagram

Describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. The activity diagram of the system for the decoding operation is shown in Figure 3.4.
Figure 3.4 Activity Diagram
3.12 Data Integrity

The data should be checked before using it, the program performs that in many parts, all these conditions should be checked in order to prevent system crash and insure robust and reliable system.

3.12.1 Input Integrity

The program checks the imported data before creating a pack file and decode it, the file imported should has the following conditions:

- It should be a text file.
- The file has no blank lines in it.
- Each beacon should contains the first part KNSAT.

3.12.2 Display Integrity

The display method mentioned in section 3.9.4 should check the array boundaries of the pack file array and the beacon object array, the program should insure the following:

- Not to access an element after the last packet file object.
- Not to access an element before the first pack file object.
- Keep the user in the range of beacons object, without accessing a beacon before or after the boundaries object.

Another Display integrity requirements is the buttons press integrity which should insure the following:

- Prevent drawing chart without importing a file, at least one file should be imported to draw chart.
- Notifying the user when saving without importing a file, at least one file should be imported to be saved.
- The program make sure of the saving path before saving it
- Handel any un expected error in reaching un known location.
4 CHAPTER FOUR : IMPLEMENTATION RESULTS AND DISCUSSION

4.1 Results Obtained

4.1.1 CubeSat Telemetry Decoder Software Interface Tabs

The software has three tabs in its interface:

4.1.1.1 Start Tab

When the user runs the program, the start tab is displayed which contains the title and the KNSAT logo. The start tab is shown in Figure 4.1

![Figure 4.1 Program Start Tab](image)

4.1.1.2 The Beacon Tab

The main tab is the beacon tab, the user’s most methods are in the beacon Tab, it contains the methods of decoding, drawing charts, saving and going through beacons imported, Also the chart diagram is shown in this tab.

The beacon tab is shown in Figure 4.2
4.1.1.3 Options Tab

The options tab allows the user to modify some settings, such as the temperature unit, the theme and the chart display area, the options tab is shown in Figure 4.3.
4.1.2 Decoding Results

4.1.2.1 Single File

First the user has the beacon data in a text file such as in Figure 4.4

![Figure 4.4 Beacon Text File](image1)

7 beacons in this file

Then the user imports this file from the *Import button* and choose the text file where it is located as shown in Figure 4.5

![Figure 4.5 Import Beacon Text File](image2)

The program displays the decoded values as shown in Figure 4.6

The beacon text file

Press Open
The user can have more than text file to view at the same time, text files are shown in Figure 4.7.

**Figure 4.6 Decoded Values**

**4.1.2.2 Multiple Files**

The user can have more than text file to view at the same time, text files are shown in Figure 4.7.

**Figure 4.7 Multiple beacons files**
The user can import more than a file at the same time and goes through it using the next and previous buttons to display the next and previous beacons and file, Figure 4.8

### 4.1.3 Chart Drawing Results

For the elements of the Voltages, Currents, Powers and Temperatures buttons the user presses it to draw chart in the Chart area,

#### 4.1.3.1 Same Frame Display Chart

The user presses the chart buttons, Figure 4.9
4.1.3.2 Separate Chart Frame

To display the chart in separate window, the user changes the options from the options tab and check the view in separate window option, Figure 4.10 and press the chart buttons to view Charts, Figure 4.11.
4.1.4 Alarm Results

The program may display Alarm for out of range values, Figure 4.12.
4.1.5 Save And Save As Results

4.1.5.1 Save File

The save button saves the decoded data of the file in a local database, then the user is notified the user with “New data is added to file” message, Figure 4.13

The local database is shown in Figure 4.14
4.1.5.2 **Save As File**

The user can save the pack file in a separate file with a specific name in a specific directory, Figure 4.15.

![Figure 4.15 The Save As Decoded Data](image1)

The file is saved in the desired location with the desired name, Figure 4.16.

![Figure 4.16 Save As In Text File](image2)

The user is notified by a message as in Figure 4.13.

4.1.6 **Options Results**

4.1.6.1 **Themes**

The user three options of colors, Figure 4.17 such as Pink Figure 4.18, Gray Figure 4.2 and Yellow Figure 4.19.
Figure 4.17 Themes Options

Figure 4.18 Pink Theme
4.1.6.2 Temperature Unit

The User can change the temperature unit from the options tab, the default received temperature is Celsius, Figure 4.20.

Then the temperature is changed as in Figure 4.21
4.1.7 Help toolbar

When the user move the cursor over the display component, the help toolbar gives help tips about the button. Help tip Figure 4.22 is shown when cursor is over the import button, Help tip Figure 4.23 is shown when moving the cursor over the save button.

![Figure 4.22 Import Help](image)

![Figure 4.23 Save Help](image)

4.1.8 Data Integrity Results

4.1.8.1 Wrong Importing

If the user import a file that contains a wrong text file Figure 4.24, Figure 4.25 the following message “Wrong File!” is shown to the user, as shown in Figure 4.26.
IMPLEMENTATION RESULTS AND DISCUSSION

CHAPTER 4

Figure 4.24 Not Beacon Text File

Figure 4.25 Importing Wrong Text File
Importing file other than text file gives the “Wrong file” message, Figure 4.26.

**4.1.8.2 Saving Without Importing**

Pressing Save or Save as buttons decoded data without having a pack file gives “Please Import file to be saved” message, Figure 4.27.
4.1.8.3 Wrong Saving

Any failing in the saving process gives “Something is wrong, please try again!” message, Figure 4.28.

Figure 4.27 Non-Importing Saving

Figure 4.28 Wrong Saving
4.2 Results Discussion

4.2.1 Results Analysis

When the software was tested, it achieved the objectives listed in chapter 1 section 1.4 and met the requirements mentioned in section 3.2 of functional and non-functional requirements.

The system was designed to decode the KNSAT beacon data, and it does it successfully, the beacon format was the best format, it is easy, fast and less power consumption for the CubeSat, no calculations needed. Using a look up table made it easy for any modification in values’ code. It is compatible for any decoder using the same decoding method.

The OOP was the best choice for the system implementation and was successfully managed to has many beacons data in many file at the same time without mix in any, that would never be made without OOP.

Classes was coherent and interact as designed, the connectivity was well designed and work as specified, each class was able to implements its method without errors.

The decoding was successfully obtained as designed and the results matched the data format as mentioned in section 3.5.2, the feature of multiple importing files made the program more easier for the user to use, thus, no need to delete the previous file each time the user intend to decode another file.

Charts was clear to the user and details was illustrated in it, some features are available in the chart diagram as saving, coping, printing the image and axis scaling, the ability to view charts in separate Frame give the user the ability to compare between values in the same file or other files without erasing the chart each time.

The alarm method was successful in detecting danger values and alert the user for these values, This made the software not just a decoder software but also monitoring software, were the user can monitor every behavior in the CubeSat and be aware of the danger occurs in it.

The saving feature allows the user to save the decoded data so no need to decode the file each time the user wants to check it, that increase the performance of the software, and make it easy for the user to view more data, the decoded data can be stored in a local database where the whole decoded data is saved together and each decoded file is separated from the next file by a blank line, the user can view the whole decoded data in a single file, the user can also save the data in a separate file
different from the local data wherever the user wants to save it and with a chosen name for the file, saving the decoded data in a text file make it easy for the user to use any spreadsheet software to view data.

The options setting give the user the freedom to choose a comfortable theme, temperature unit and chart display area.

The software was able to handle errors successfully without causing a system crash, the software was able to detect the wrong file before creating a pack class for it, this protects the software from falling down.

The program filters the files shown in the importing window and only shows the text files so the user not choose non-text file, also even if the user choose a non-text file the program detects it and shows the wrong file message.

Saving decoded data should also be checked so any errors occurs while saving should be handled and prevent the program from crash.

The KNSAT data format was chosen carefully, E was not used in Morse code because it can be lost with noise since it is just a dot, For elements, the most likely values to be sent was given a short Morse code values so it will not take long time in sending when using CW protocol.

Although the format of the look up table may cause decrease of accuracy of values, the values was carefully chosen to get high accuracy as possible, the steps in the battery voltage step is 1v step, the temperature is only 5C steps, the bus voltage is just 0.25v step.

The software was successfully implemented and tested in KNSAT ground station of University of Khartoum, Figure 4.29.
Figure 4.29 KNSAT Telemetry Decoder in Ground Station UofK

4.2.2 Special features in the software

The KNSAT Telemetry Decoder Software has many features that other decoders doesn’t:

- Importing more than file at the same time.
- Saving decoded values in local database.
- Saving decoded values in separate file.
- Flexibility of viewing the charts.
- Options of temperature unit.
- Many themes available.
- Easy help tool.
- Flexible encoding values changing.
- Running in any machine that has the Java Run Time Environment.
5 CHAPTER FIVE : CONCLUSION AND FUTURE WORK

5.1 Conclusion

The telemetry decoder software that decodes the KNSAT beacon data was successfully implemented. The system has four main methods; decoding, drawing charts, saving and displaying.

The decoding method is the most important method that is the system main usage of decoding the file that the user import which contains the beacon data to understandable values according to specific beacon format.

The drawing chart method allows the user to view charts of some components in bar charts so the user can see the variation of a component during time, the chart make sense of changing components’ value with time.

Saving the decoded data is an imported method that the user needs to return for going through large amount of data without consuming process each time for large amount of data, the data can be saved in the same directory or in separate file.

The display methods which the user interact with, it gets the user requests and ask other classes for implementation, this class manages the object displayed for the user according to the user usage.
5.2 Future work

The system can be automated more than this, it could be a real time system where the antenna is connected to the telemetry decoder which decodes from sound signal to decoded values for the user without the manual process by the user.

The system can be connected with other parts of ground station such as the tracking software to get the AOS and LOS to open and close the port, for receiving data signal, also the commanding process could be used to send commands to control the CubeSat.

In addition, the decoder could be implemented in a server PC, and the users gets the decoded data Online and import the files in servers, this feature needs distributed system concepts, Also the system could be extended for phone use and ipad versions.

The system can be improved so that users around the world upload telemetry directly to a server rather than sending them by mail.

Also the data can be saved in Database (SQL for example).
References

Appendix A: KNSAT DATA FORMATs

The KNSAT data format is presented in Table A.1

<table>
<thead>
<tr>
<th>Header</th>
<th>No. of chars</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN0</td>
<td>X chars</td>
<td>Call sign</td>
</tr>
<tr>
<td>KN1</td>
<td>5 chars</td>
<td>Time</td>
</tr>
<tr>
<td>KN2</td>
<td>5 chars</td>
<td>Mode + status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Transmission mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Payload request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Status(On/Off)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ OBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ Payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ Transmitter</td>
</tr>
<tr>
<td>KN3</td>
<td>4 chars</td>
<td>Bus + battery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Battery:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ Voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ Temperature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bus 3.3:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ Voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bus 5:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◆ Voltage</td>
</tr>
<tr>
<td>KN4</td>
<td>9 chars</td>
<td>Temperatures:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Antenna</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cells (6 cells)</td>
</tr>
</tbody>
</table>
Transmission Modes Table A.2:

**Table A.2 Transmission Modes Codes**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value</th>
<th>In Morse</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>I</td>
<td>..</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>.-</td>
</tr>
<tr>
<td>D</td>
<td>N</td>
<td>-.</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>-..</td>
</tr>
</tbody>
</table>

Payload request Table A.3:

**Table A.3 Payload request Codes**

<table>
<thead>
<tr>
<th>Value</th>
<th>In Morse</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>-</td>
<td>No payload data to be sent</td>
</tr>
<tr>
<td>I</td>
<td>..</td>
<td>Payload data ready to be sent</td>
</tr>
</tbody>
</table>

Cubesat Status Table A.4:

**Table A.4 Statuses Codes**

<table>
<thead>
<tr>
<th>Value</th>
<th>In Morse</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>-</td>
<td>Is ON</td>
</tr>
<tr>
<td>I</td>
<td>..</td>
<td>Is OFF</td>
</tr>
</tbody>
</table>

Battery Voltage Table A.5:

**Table A.5 Battery Voltage**

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Value</th>
<th>In Morse</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>I</td>
<td>..</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>.-</td>
</tr>
</tbody>
</table>
Temperature and voltage Table A.6:

### Table A.6 Temperature and voltage Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Temperature</th>
<th>Bus Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-40</td>
<td>&lt;0.00</td>
</tr>
<tr>
<td>B</td>
<td>-35</td>
<td>0.00</td>
</tr>
<tr>
<td>C</td>
<td>-30</td>
<td>0.25</td>
</tr>
<tr>
<td>D</td>
<td>-25</td>
<td>0.50</td>
</tr>
<tr>
<td>F</td>
<td>-15</td>
<td>1.00</td>
</tr>
<tr>
<td>G</td>
<td>-10</td>
<td>1.25</td>
</tr>
<tr>
<td>H</td>
<td>-5</td>
<td>1.50</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1.75</td>
</tr>
<tr>
<td>J</td>
<td>5</td>
<td>2.00</td>
</tr>
<tr>
<td>K</td>
<td>10</td>
<td>2.25</td>
</tr>
<tr>
<td>L</td>
<td>15</td>
<td>2.50</td>
</tr>
<tr>
<td>M</td>
<td>20</td>
<td>2.75</td>
</tr>
<tr>
<td>N</td>
<td>25</td>
<td>3.00</td>
</tr>
<tr>
<td>O</td>
<td>30</td>
<td>3.25</td>
</tr>
<tr>
<td>P</td>
<td>35</td>
<td>3.50</td>
</tr>
<tr>
<td>Q</td>
<td>40</td>
<td>3.75</td>
</tr>
<tr>
<td>R</td>
<td>45</td>
<td>4.00</td>
</tr>
<tr>
<td>S</td>
<td>50</td>
<td>4.25</td>
</tr>
<tr>
<td>T</td>
<td>55</td>
<td>4.50</td>
</tr>
</tbody>
</table>
Bus Voltage Table A.7:

Table A.7 Bus Voltage Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Bus Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.00</td>
</tr>
<tr>
<td>B</td>
<td>0.25</td>
</tr>
<tr>
<td>C</td>
<td>0.50</td>
</tr>
<tr>
<td>D</td>
<td>0.75</td>
</tr>
<tr>
<td>F</td>
<td>1.25</td>
</tr>
<tr>
<td>G</td>
<td>1.50</td>
</tr>
<tr>
<td>H</td>
<td>1.75</td>
</tr>
<tr>
<td>I</td>
<td>2.00</td>
</tr>
<tr>
<td>J</td>
<td>2.25</td>
</tr>
<tr>
<td>K</td>
<td>2.50</td>
</tr>
<tr>
<td>L</td>
<td>2.75</td>
</tr>
<tr>
<td>M</td>
<td>3.00</td>
</tr>
<tr>
<td>N</td>
<td>3.25</td>
</tr>
<tr>
<td>O</td>
<td>3.50</td>
</tr>
<tr>
<td>P</td>
<td>3.75</td>
</tr>
<tr>
<td>Q</td>
<td>4.00</td>
</tr>
<tr>
<td>R</td>
<td>4.25</td>
</tr>
<tr>
<td>S</td>
<td>4.50</td>
</tr>
<tr>
<td>T</td>
<td>4.75</td>
</tr>
<tr>
<td>U</td>
<td>5.00</td>
</tr>
<tr>
<td>V</td>
<td>5.25</td>
</tr>
<tr>
<td>W</td>
<td>5.50</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>X</td>
<td>5.75</td>
</tr>
<tr>
<td>Y</td>
<td>6.00</td>
</tr>
<tr>
<td>Z</td>
<td>6.25</td>
</tr>
</tbody>
</table>
APPENDIX B: CONFIGURATIONS

The package contains three files:

First file is the sources file, To install Java programming language follow the instructions:

- First run the “jdk-6-windows-i586” the java development kit.
- Then install the java NetBeans programming language that is used for coding.
- Run the JRE - Java Run time Environment that allows you to run JAR java execution files.

To open the Decoder source code

- After installing the NetBeans open project choose the decoder file
- You should link the JFREECHART:

  Right click on the project name and choose properties

  Choose Libraries and add JAR for these files: jfreechart-1.0.13 & jcommon-1.0.16 from the jfreechart file >> Lib

Now the source code is ready to be used.

Also, an EXE file is included and can be run easily.
APPENDIX C: CODE

Piece of codes for converting method:

Convert statuses:

```java
private void toKN2(String kn2)
{
    try{
        int k2=0;
        mode = toMode(kn2.charAt(k2++));
        payloadRequest = toPayloadRequest(kn2.charAt(k2++));
        statusPayload = toStatus(kn2.charAt(k2++));
        statusOBC = toStatus(kn2.charAt(k2++));
        statusTx = toStatus(kn2.charAt(k2++));
        status = toStatus(kn2.charAt(k2++));
    } catch(Exception e){}
}
```

Convert voltages:

```java
private void toKN3(String kn3)
{
    try{
        int k3=0;
        batteryVoltage = toBatteryVoltage(kn3.charAt(k3++));
        batteryTemp = toTemp(kn3.charAt(k3++));
        bus3Voltage = toBusVoltage(kn3.charAt(k3++));
        bus3Current = toBusCurrent(bus3Voltage);
        bus5Voltage = toBusVoltage(kn3.charAt(k3++));
        bus5Current = toBusCurrent(bus5Voltage);
        pwrBus3 = power(bus3Voltage,bus3Current);
        pwrBus5 = power(bus5Voltage,bus5Current);
    } catch(Exception e){}
}
```

Convert Temperatures:

```java
private void toKN4(String kn4)
{
    try{
        int k4=0;
        tempPayload = toTemp(kn4.charAt(k4++));
        tempAntenna = toTemp(kn4.charAt(k4++));
    }
```
tempOBC = toTemp(kn4.charAt(k4++));
tempCell1 = toTemp(kn4.charAt(k4++));
tempCell2 = toTemp(kn4.charAt(k4++));
tempCell3 = toTemp(kn4.charAt(k4++));
tempCell4 = toTemp(kn4.charAt(k4++));
tempCell5 = toTemp(kn4.charAt(k4++));
tempCell6 = toTemp(kn4.charAt(k4++));
} catch(Exception e) {} 

The Alarm method:

```java
public void alarm(){
    if(mode.equals("F")) modeAlarm=true;
    if(payloadRequest.equals("error")||payloadRequest.equals("Payload data ready to be sent")) payloadRequestAlarm=true ;
    if(statusPayload.equals("OFF")) statusPayloadAlarm=true;
    if(statusOBC.equals("OFF")) statusOBCAlarm=true;
    if(statusTx.equals("OFF")) statusTxAlarm=true;
    if(batteryVoltage<11) batteryVoltageAlarm=true;
    if(batteryTemp<0) batteryTempAlarm=true;
    if(bus3Voltage<3) bus3VoltageAlarm=true;
    if(bus3Current<0) bus3CurrentAlarm=true;
    if(bus5Voltage<4) bus5VoltageAlarm=true;
    if(bus5Current<0) bus5CurrentAlarm=true;
    if(pwrBus3<0) pwrBus3Alarm=true;
    if(pwrBus5<0) pwrBus5Alarm=true;
    if(tempPayload<0||tempPayload>60) tempPayloadAlarm=true;
    if(tempAntenna<0||tempAntenna>60) tempAntennaAlarm=true;
    if(tempOBC<0||tempOBC>60) tempOBCAlarm=true;
    if(tempCell1<0||tempCell1>60) tempCell1Alarm=true;
    if(tempCell2<0||tempCell2>60) tempCell2Alarm=true;
    if(tempCell3<0||tempCell3>60) tempCell3Alarm=true;
    if(tempCell4<0||tempCell4>60) tempCell4Alarm=true;
    if(tempCell5<0||tempCell5>60) tempCell5Alarm=true;
    if(tempCell6<0||tempCell6>60) tempCell6Alarm=true;
}
```

Check function:

```java
public boolean check(String fileName ){
    /* The method is used to check the validation of the imported file
     * It should be text file and each line should start by KNSAT call sign
     */
    boolean checked=true;
```
String line;
try{
    BufferedReader data = new BufferedReader(new FileReader(fileName));
    while(((line=data.readLine())!=null)& checked ){
        String [] KN =line.split(" ");

        if(KN[0].toUpperCase().equals("KNSAT")){
            checked=true;
        }
        else{
            checked=false;
        }
    }
}