TUGAS AKHIR – KI091391

MANAJEMEN KUALITAS ALIRAN DATA *AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST* (ADS-B) BANYAK TITIK DENGAN POHON KEPUTUSAN

ADRIANUS YOZA APRILIO
5110100085

Dosen Pembimbing I
Dr.tech. Ir. R.V.HARI GINARDI, M.Sc.

Dosen Pembimbing II
YUDHI PURWANANTO, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2014

FINAL PROJECT – KI091391

MULTIPLE NODE AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST (ADS-B) DATA STREAM QUALITY MANAGEMENT USING DECISION TREE

ADRIANUS YOZA APRILIO
5110100085

Supervisor
Dr.tech. Ir. R.V.HARI GINARDI, M.Sc.
YUDHI PURWANANTO, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
Sepuluh Nopember Institute of Technology
Surabaya, 2014

MANAJEMEN KUALITAS ALIRAN DATA *AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST* (ADS-B) BANYAK TITIK DENGAN POHON KEPUTUSAN

Nama Mahasiswa : Adrianus Yoza Aprilio
NRP : 5110 100 085
Jurusan : Teknik Informatika FTIF ITS
Dosen Pembimbing 1 : Dr.tech. Ir. R.V.Hari Ginardi, M.Sc.
Dosen Pembimbing 2 : Yudhi Purwananto, S.Kom., M.Kom.

Abstrak

Pohon keputusan adalah salah satu metode untuk melakukan pemilihan dan penyaringan data. Pemilihan dan penyaringan data, merupakan bagian dari usaha untuk menjaga kualitas data. Salah satu contoh dari permasalahan pentingnya kualitas data adalah data pada dunia transportasi, utamanya penerbangan. Data yang dihasilkan harus merupakan data yang memiliki tingkat validitas dan kepercayaan cukup tinggi, karena data yang dihasilkan mempengaruhi kehidupan banyak orang.

Pada tugas akhir ini diterapkan metode pohon keputusan (Decision Tree) untuk menunjang Manajemen Kualitas Aliran Data pada Automatic Dependent Surveillance Broadcast Banyak Titik. Data yang dikirim station-station yang ada akan dikonsolidasikan menjadi sebuah data yang tingkat validitas dan kepercayaannya terukur, sehingga data yang dikeluarkan dapat menjadi data yang valid untuk observasi dan simulasi, bahkan dalam kondisi terdesak, dapat digunakan untuk rekonstruksi.

Hasil uji coba terhadap ground-truth, intern data, dan ekstern data membuktikan bahwa pohon keputusan dapat menunjang proses manajemen kualitas aliran data pada ADS-B banyak titik, sehingga data yang terbentuk adalah valid dan terpercaya.

Kata kunci: ADS-B, Aliran data, Kualitas data, Banyak titik, Pohon keputusan

MULTIPLE NODE AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST (ADS-B) DATA STREAM QUALITY MANAGEMENT USING DECISION TREE

Student's Name : Adrianus Yoza Aprilio
Student's ID : 5110 100 085
Department : Informatics Engineering, FTIF-ITS
First Advisor : Dr.tech. Ir. R.V.Hari Ginardi, M.Sc.
Second Advisor : Yudhi Purwananto, S.Kom., M.Kom.

Abstract

Decision tree is one of the methods assigned for doing data selection and filtering. Data selection and filtering is part of managing data quality. One example of the problem is managing data quality in transportation data, mainly aviation data. Aviation data requires almost absolute validity and trustworthy, since these data will affect many people's live.

In this final project, Decision Tree is applied as a method in Multiple Node Automatic Dependent Surveillance-Broadcast Data Stream Quality Management. We need to consolidate data sent from stations to be valid and trustworthy data. So that the output data could be used in observation and simulation, even reconstruction in such of emergency case.

Based on trials on ground truth, internal data, and external data, it is proven that decision tree could be used to manage multiple node ADS-B Data Stream Quality, so that generated data is valid and trustworthy.

Keywords: ADS-B, Data quality, Data stream, Decision tree, Multiple node

LEMBAR PENGESAHAN

MANAJEMEN KUALITAS ALIRAN DATA *AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST* (ADS-B) BANYAK TITIK DENGAN POHON KEPUTUSAN

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
Pada
Bidang Studi Komputasi Cerdas Visual
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
ADRIANUS YOZA APRILIO
NRP: 5110 100 085

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Dr.tech. Ir. R.V.HARI GINARDI, M.Sc.
   NIP: 19650518 199203 1 003 (Pembimbing 1)

2. YUDHI PURWANANTO, S.Kom., M.Kom.
   NIP: 19700714 199703 1 002 (Pembimbing 2)

SURABAYA
JUNI 2014

KATA PENGANTAR

Segala puji dan syukur pada Tuhan YME yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "Manajemen Kualitas Aliran Data *Automatic Dependent Surveillance-Broadcast* (ADS-B) Banyak Titik Dengan Pohon Keputusan" dengan tepat waktu.

Harapan dari penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi yang nyata bagi kampus Teknik Informatika, ITS, dan bangsa Indonesia.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Bapak Dr.tech. Ir. R.V.Hari Ginardi, M.Sc. selaku dosen pembimbing penulis yang telah memberikan ide, kesempatan observasi, bimbingan, saran, kritik, dan ilmu yang sangat bermanfaat hingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Yudhi Purwananto, S.Kom., M.Kom., selaku dosen pembimbing yang telah memberikan nasihat, arahan, dan bimbingan sesuai dengan pengalaman dan kapasitasnya sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Ibu Dr. Nanik Suciati, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
4. Ayah, ibu, adik, dan keluarga besar penulis yang telah memberikan *support* baik material maupun pengorbanan waktu serta tenaga dan pikiran kepada penulis hingga akhirnya dapat menyelesaikan tugas akhir ini.

5. Tim dari laboratorium Sistem Cerdas (IF-304) yang memberikan saran dan masukan atas Tugas Akhir ini.
6. Mbak Noor Fitria Azzahra, yang menjadi rekan riset ADS-B dan meluangkan waktu serta memberikan lokasi yang sangat dekat dengan bandara Juanda untuk membantu proses *data logging*.
7. Tim dari Nokia/Microsoft Mobility Innovation Lab, yang memberikan akses, fasilitas, dan keleluasaan kepada penulis untuk menyelesaikan Tugas Akhir ini.
8. Bapak Erwin Tosca dan tim ISP Hypernet, yang memberikan *support* lokasi dan infrastruktur jaringan untuk melakukan *data logging*.
9. Bapak Taufan, Bapak N. Agustino, dan tim ISP Crossnet, yang memberikan akses infrastruktur jaringan untuk melakukan *data processing*.
10. Persaudaraan Eks Siswa-Siswi Kolese Santo Yusup (PEKSY), terutama Bapak Taro Lay dan Bapak Bambang Pranoto, yang memberikan akses ke *server* untuk melakukan *data processing*.
11. Bapak Gerry Soejatman, Bapak Vincent Herdison, Bapak Iwan Yunariawan, serta seluruh anggota forum penerbangan Ilmuterbang.com yang menjadi narasumber dan rekan diskusi, serta memberikan banyak literatur dalam riset mengenai ADS-B.
12. Bapak Stefendy Handoko dan bapak Bobby Tri, serta seluruh anggota komunitas Diskusi-SDR yang membantu proses pengenalan terhadap SDR dan memberikan banyak masukan untuk memperbaiki kualitas sinyal data ADS-B yang diterima.
13. Rekan-rekan penulis di forum Indoflyer, utamanya Indoflyer regional Surabaya (PK-SUB), yang menjadi rekan diskusi dan memberikan masukan serta *review* untuk tugas akhir ini.
14. Tidak lupa juga, teman-teman seangkatan TC 2010 yang telah berjuang bersama hingga saat ini dan terus saling

memotivasi untuk dapat segera menyelesaikan tahap sarjana di Teknik Informatika ITS.

15. Serta seluruh pihak-pihak lain yang belum sempat disebutkan satu per satu di sini yang telah membantu terselesaikannya tugas akhir ini.

Tugas Akhir ini merupakan persembahan penulis untuk kedua orang tua penulis yang selalu mengingatkan untuk menuntut ilmu setinggi-tingginya.

Kesempurnaan tentu masih jauh tercapai pada Tugas Akhir ini, maka penulis mengharapkan saran dan kritik yang membangun dari pembaca untuk perbaikan ke depan. Semoga Tugas Akhir ini dapat bermanfaat bagi perkembangan ilmu pengetahuan dan bagi semua pihak.

Surabaya, Juni 2014

Adrianus Yoza Aprilio

DAFTAR ISI

DAFTAR TABEL

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini dibahas hal-hal yang mendasari Tugas Akhir. Bahasan meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika laporan Tugas Akhir.

1.1 Latar Belakang

Saat ini, transportasi merupakan salah satu kebutuhan manusia, termasuk transportasi udara. Peningkatan jumlah pengguna transportasi udara ini mulai memperlihatkan dampaknya dengan kepadatan lalu lintas di udara [1]. Kepadatan lalu lintas udara, selama ini hanya dapat dipantau oleh radar yang dimiliki oleh regulator perhubungan udara, dalam hal ini *Air Traffic Control* (ATC). Pada kenyataannya, banyak data yang disiarkan oleh pesawat dapat diakses oleh publik, salah satunya adalah *Automatic Dependent Surveillance-Broadcast* (ADS-B).

Automatic Dependent Surveillance (ADS) adalah sebuah teknologi masa depan yang akan menggantikan radar sebagai cara untuk melakukan *tracking* pada pesawat udara. ADS adalah mekanisme komunikasi baik antar pesawat maupun antara pesawat dengan stasiun di darat (*ground*) [2]. Data yang dikomunikasikan meliputi lalu lintas dan pergerakan pesawat, cuaca, informasi geografis termasuk *terrain*, dan informasi penerbangan seperti *Notice to Airmen* (NOTAM) dan *Temporary Flights Restriction* (TFR).

Salah satu aspek dalam ADS adalah ADS-B yang dapat diakses secara bebas pada frekuensi 1.090MHz. Data ADS-B akan disiarkan oleh pesawat yang menggunakannya pada frekuensi tersebut [3]. Salah satu data terpenting dari ADS-B adalah data pergerakan masing-masing pesawat itu sendiri. Namun karena sensitivitas frekuensi ini terhadap halangan baik bangunan maupun sinyal radio, diperlukan lebih dari satu penerima (*receiver*) untuk

bisa meliputi sebuah area yang cukup besar dan menangkap sinyal dari seluruh pesawat yang berada pada area tersebut.

Dengan penggunaan beberapa penerima, sebuah data yang sama dapat diterima di dua penerima pada waktu yang sedikit berbeda, dengan kondisi (kelengkapan) yang berbeda pula. Karena kondisi inilah, dibutuhkan sebuah metode atau mekanisme yang dapat menyatukan aliran data (*data stream*) dari banyak penerima.

Diharapkan, metode penentuan kepastian data dengan pohon keputusan dapat menghasilkan sebuah data yang merepresentasikan detil pergerakan yang sebenarnya. Data yang dihasilkan pada Tugas Akhir ini dapat digunakan sebagai sumber data untuk melakukan observasi dan simulasi pergerakan pesawat. Selain itu, data ini juga dapat digunakan sebagai sumber data untuk sistem cerdas lain yang berhubungan dengan dunia penerbangan.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana metode penyaringan data yang efektif agar aliran data yang tidak diperlukan tidak mengganggu kinerja sistem?
2. Bagaimana proses pemilihan sebuah data dari dua atau lebih penerima (receiver) ADS-B dengan pohon keputusan sehingga dapat merepresentasikan pergerakan yang sebenarnya?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu:

1. Data ADS-B yang diproses dibatasi pada data:
    a. identitas pesawat (*hex code*)
    b. nomor penerbangan (*flight number - callsign*)
    c. lokasi pesawat (*coordinate*)

   d. arah pergerakan (*heading*)
   e. ketinggian (*flight level* dan *altitude*)
   f. kecepatan terukur di darat (*groundspeed)* maupun udara *(airspeed)*
2. Lingkup Tugas Akhir ini hingga menghasilkan sebuah data yang dapat digunakan sebagai acuan/sumber data untuk perangkat yang lain.
3. Format penerimaan aliran data ADS-B mengikuti standar manual Mode-S RTL1090 yang didapatkan dari halaman manual RTL1090 [4].
4. Data yang digunakan adalah data yang memang dikirim oleh *receiver* ADS-B, bukan merupakan perkiraan atau dugaan.
5. Data akhir yang dihasilkan berupa format *Keyhole Markup Language* (KML) dan *JavaScript Object Notation* (JSON).

1.4 Tujuan dan Manfaat

Tujuan pengerjaan Tugas Akhir ini adalah:
1. Menentukan metode penyaringan data yang efektif agar aliran data yang tidak diperlukan tidak mengganggu kinerja sistem.
2. Membuat mekanisme atau metode pemilihan sebuah data dari dua atau lebih penerima (*receiver*) ADS-B dengan pohon keputusan sehingga dapat merepresentasikan pergerakan yang sebenarnya.

Manfaat pengerjaan Tugas Akhir ini adalah:
1. Mendapatkan metode pemilihan dan penyaringan yang efektif untuk aliran data ADS-B di banyak titik.
2. Mendapatkan data gabungan dari beberapa penerima ADS-B yang dapat dimanfaatkan perangkat lain.
3. Mendapatkan arsip data yang valid, komprehensif, dan berkualitas baik sehingga dapat digunakan untuk observasi atau rekonstruksi rangkaian kejadian dalam penerbangan apabila dibutuhkan.

1.5 Metodologi

Metodologi yang digunakan pada pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Studi Literatur
   Pada proses ini dilakukan studi lebih lanjut terhadap konsep-konsep yang terdapat pada jurnal, buku, artikel, dan literatur lain yang menunjang. Studi dilakukan untuk mendalami konsep mengenai standar komunikasi Mode-S, ADS-B, konsep algoritma pohon keputusan, proses seleksi data, dan struktur data KML serta JSON yang diharapkan menjadi keluaran aplikasi ini.
2. Implementasi Algoritma
   Implementasi merupakan tahap untuk membangun sistem pohon keputusan dan seleksi data. Sistem akan diimplementasikan dengan bahasa pemrograman Python.
3. Uji Coba dan Evaluasi
   Uji coba dan evaluasi dilakukan dengan tahap:
   Performa dari algoritma yang diterapkan akan dievaluasi dengan pembandingan hasil program dengan aplikasi atau portal yang menyediakan layanan serupa, yakni PlaneFinder dan flightradar24.com.
4. Penyusunan Laporan Tugas Akhir
   Pada tahap ini, disusun buku berisi metode, dasar teori, hasil uji coba, dan hasil evaluasi dari Tugas Akhir.

1.6 Sistematika Laporan

Buku Tugas Akhir ini disusun dengan sistematika sebagai berikut:

1. Bab I Pendahuluan
   Pada bab ini dibahas latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika laporan Tugas Akhir.
2. Bab II Tinjauan Pustaka

Pada bab ini dibahas dasar teori yang berkaitan dengan topik Tugas Akhir.

3. Bab III Perancangan
   Pada bab ini dibahas perancangan sistem konsolidasi data dan cara pengujian.
4. Bab IV Implementasi
   Pada bab ini dibahas implementasi rancangan yang dibuat pada tahap perancangan.
5. Bab V Uji Coba dan Evaluasi
   Pada bab ini dibahas tahap-tahap uji coba. Kemudian hasil uji coba dievaluasi untuk mengetahui validitas data yang dihasilkan oleh sistem.
6. Bab VI Kesimpulan dan Saran
   Pada bab ini disimpulkan hasil perbandingan. Dibahas juga saran untuk pengembangan lebih lanjut.

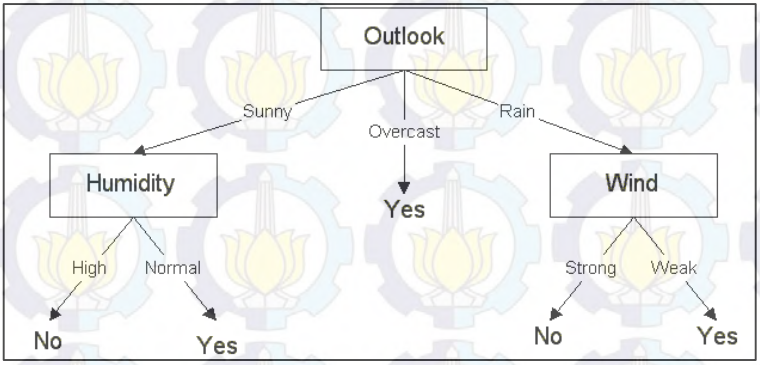
BAB II
TINJAUAN PUSTAKA

Pada bab ini dijelaskan dasar teori yang digunakan dalam Tugas Akhir. Dasar teori yang dijelaskan mengenai metode, alat, serta definisi data yang akan digunakan.

2.1. Pohon Keputusan

Pohon keputusan (*decision tree*) adalah salah satu metode klasifikasi yang paling populer karena mudah dimengerti oleh manusia. Pohon keputusan adalah model prediksi menggunakan struktur pohon atau struktur hierarki. Konsep dari pohon keputusan adalah mengubah data menjadi pohon keputusan dan aturan-aturan keputusan [5].

Manfaat utama dari penggunaan pohon keputusan adalah kemampuannya untuk membuat proses pengambilan keputusan yang kompleks menjadi lebih simpel sehingga pengambil keputusan akan lebih menginterpretasikan solusi dari permasalahan. Pohon keputusan juga berguna untuk mengeksplorasi data, menemukan hubungan tersembunyi antara sejumlah calon variabel masukan dengan sebuah variabel target.



Gambar 2-1 Contoh Pohon Keputusan Sederhana

Pohon keputusan juga dapat digunakan untuk melakukan proteksi dan penyaringan terhadap kualitas data. Artinya data yang masuk namun tidak memiliki spesifikasi dan variabel sesuai dengan batasan tertentu, dapat dibuang agar tidak mengganggu data yang lain.

Pada **Gambar 2-1** ditunjukkan contoh pohon keputusan sederhana mengenai faktor-faktor yang dipertimbangkan seseorang untuk bermain tenis [6].

2.2. Kualitas Data

Kualitas data adalah bagian dari tata kelola data. Utamanya, untuk data yang diperoleh dari pengukuran, atau merupakan hasil perkiraan / perhitungan. Kualitas data mempunyai pengertian tentang kelengkapan dan keakuratan data, juga berhubungan dengan konsistensi dan ketepatan waktu [7]. Kelengkapan itu sendiri mengandung pengertian informasi sebagai keluaran dari proses pengolahan data mewakili setiap keadaan sebenarnya [8]. Keakuratan mengandung pengertian sejauh mana data tersebut benar, dapat diandalkan [9].

2.3. Aliran Data

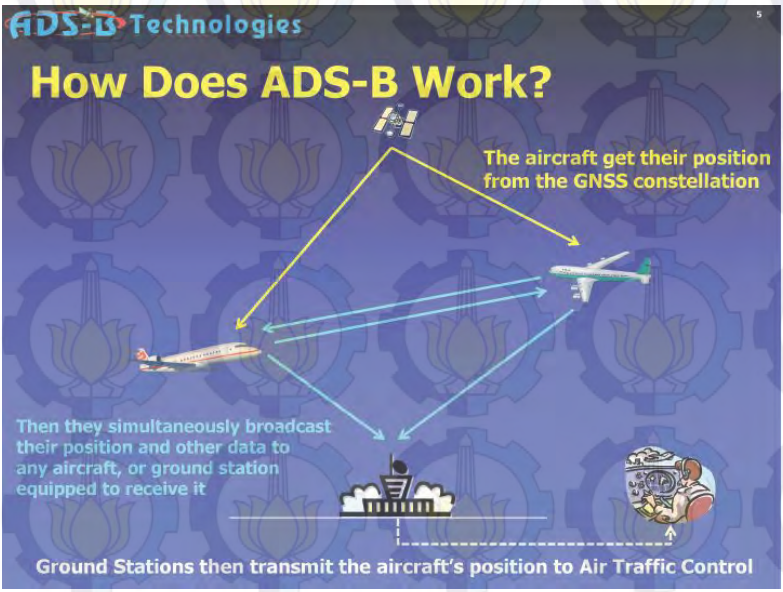
Model data aliran (*stream data*) adalah sebuah model lalu lintas data yang sebagian atau seluruh masukannya tidak berasal dari media penyimpanan, melainkan berasal dari sumber-sumber lain yang diterima secara *real-time* dan *Online* [10]. Biasanya, tipe data aliran / *stream* memiliki ciri pemrosesan sebagai berikut:

1. Tidak dibatasi ukuran.
2. Sistem tidak memiliki kuasa untuk mengurutkan data, melainkan hanya memproses apapun data yang diterimanya pada saat itu
3. Setelah diproses, kemudian data tersebut dapat diarsipkan maupun dibuang.

2.4. *Automatic Dependent Surveillance Broadcast* (ADS-B)

Automatic Dependent Surveillance-Broadcast (ADS-B) adalah sebuah teknologi yang memungkinkan pesawat yang memilikinya, memancarkan data pada frekuensi tertentu. Pada saat ini, frekuensi yang umum digunakan adalah 1.090MHz.

Data yang dipancarkan, antara lain, identitas pesawat, nomor penerbangan, lokasi pesawat, arah pergerakan, ketinggian, kecepatan, perhitungan arah angin, dan beberapa informasi lainnya. Pancaran data ini dapat diterima oleh stasiun darat dan pesawat lainnya. Informasi yang dikirimkan oleh sistem ini didapat dari informasi *Global Positioning System* (GPS) atau *Flight Management System* (FMS) yang ada di pesawat [2].



Gambar 2-2 Cara Kerja ADS-B

International Civil Aviation Organization (ICAO) dan *International Air Transport Association* (IATA) sebagai regulator internasional telah menerbitkan beberapa dokumen mengenai protokol dan perencanaan implementasi ADS-B di masa depan sebagai sarana navigasi sekunder. Cara kerja ADS-B secara umum ditunjukkan oleh **Gambar 2-2** [11].

2.5. SDR

Software-Defined Radio (SDR) adalah adalah sebuah sistem radio yang fleksibel, *multiservice*, *multistandard*, *multiband*, *reconfigurable*, dan *reprogrammable* dengan menggunakan *software*. Tidak seperti radio analog, SDR dapat diubah-ubah karakteristiknya sesuai dengan sistem radio yang dikehendaki.

Gambar 2-3 Contoh SDR RTL2832U/R820T

SDR dapat mengaplikasikan berbagai pelayanan atau servis berupa suara, teks dan data. SDR mengandalkan penggerak (*driver*),

dalam hal ini adalah sebuah perangkat lunak, untuk memilih frekuensi, standar, servis, dan konfigurasi yang dikehendaki [12].

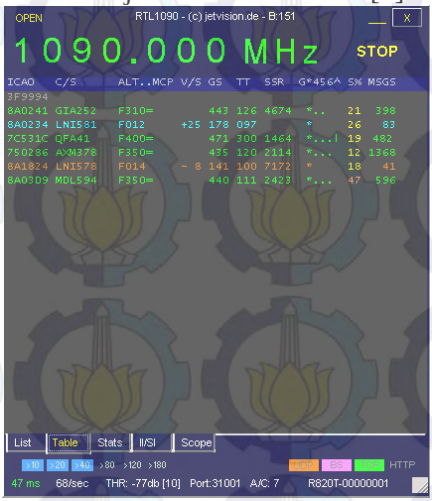
SDR yang umum digunakan untuk menangkap sinyal ADS-B adalah SDR dengan *chipset* RTL2832U/R820T seperti ditunjukkan pada **Gambar 2-3**.

2.6. Mode-S

Mode-S adalah standar pertukaran data pada dunia aviasi. Standar ini berdasarkan identifikasi pesawat sesuai standar ICAO. Standar ini juga diratifikasi oleh FAA [13]. Setiap pesawat memiliki identitas unik sepanjang 24 bit yang dapat membantu proses identifikasi masing-masing pesawat yang terlibat dalam komunikasi data.

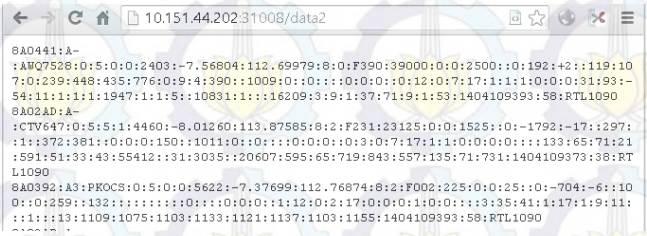
2.7. RTL1090

RTL1090 adalah sebuah aplikasi bantu yang menjadi penggerak dan mampu mengubah data pada frekuensi 1.090MHz dari bentuk aliran data dari SDR menjadi standar Mode-S [4].



Gambar 2-4 Tampilan Aplikasi RTL1090

Masukannya berupa sinyal analog dari perangkat SDR, dengan keluaran aliran data TCP, UDP, Basestation, maupun HTTP. RTL1090 dibuat oleh komunitas hobi jetvision.de. Contoh tampilan RTL1090 yang sedang berjalan ditunjukkan pada **Gambar 2-4**. Standar keluaran HTTP (*record* data) memiliki 84 variabel yang tidak selalu terisi. Daftar variabel tersebut dapat dilihat pada **Tabel 8-2**. Contoh tampilan HTTP dapat dilihat pada **Gambar 2-5**.



Gambar 2-5 Contoh *Output* HTTP RTL1090

2.8. ICAO Hex

ICAO Hex adalah identitas unik sebuah pesawat yang dibuat oleh ICAO berdasarkan alokasi negara. Sebuah pesawat akan memiliki ICAO Hex yang sama, selama pesawat tersebut tidak berpindah kepemilikan (dalam hal ini, maskapai penerbangan). Alokasi ICAO Hex dapat dilihat pada **Tabel 8-1**.

2.9. JSON

JSON (JavaScript *Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat oleh komputer [14]. Format ini dibuat berdasarkan bagian dari bahasa pemrograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemrograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh keluarga bahasa C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran data.

JSON terbuat dari dua struktur:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).
3. Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman modern mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemrograman yang juga berdasarkan pada struktur data ini.

Contoh skema JSON ditunjukkan pada **Gambar 2-6**.

```
[
  {
    hex: "8991B7",
    squawk: "----",
    flight: "CAL752",
    lat: 0,
    lon: 0,
    validposition: 0,
    altitude: "N/A",
    vert_rate: "N/A",
    track: "N/A",
    validtrack: 1,
    speed: "N/A",
    messages: 2959,
    seen: 3,
    station: "T-WARR7"
  }
]
```

Gambar 2-6 Contoh Skema JSON

2.10. KML

KML (*Keyhole Markup Language*) adalah format berkas yang digunakan untuk menampilkan data geografis pada rumpun aplikasi GIS [15]. Contoh aplikasi yang dapat menampilkan KML antara lain Google Earth dan Google Maps. Data yang dapat dimasukkan ke dalam KML antara lain informasi koordinat, arah dan sejarah pergerakan termasuk ketinggian, serta skema-skema geografis lainnya. KML adalah standar pertukaran data internasional yang dikelola oleh Open Geospatial Consortium, Inc. (OGC). Contoh skema KML ditunjukkan **Gambar 2-7**.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
  <Document>
    <Placemark>
      <description>
        Flight : AXM308
        Reg :
        Hex : 7501E5
        Type :
        Flt Level : F318
      </description>
      <name>AXM308 7501E5  F318 </name>
      <visibility>1</visibility>
      <Point>
        <altitudeMode> absolute </altitudeMode>
        <coordinates>114.75944,-
7.60035,9692</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

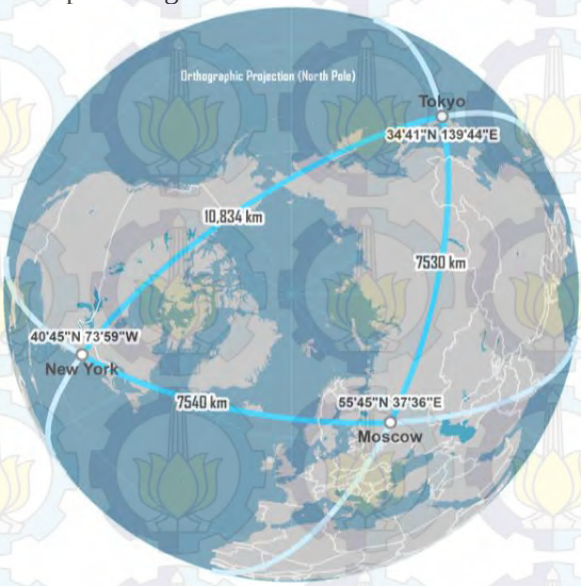
Gambar 2-7 Contoh Skema KML

2.11. Sudut antara dua titik

Pengukuran sudut antara dua titik (dalam studi kasus ini adalah koordinat lintang dan bujur), diperlukan untuk memeriksa apakah pergerakan yang dilakukan sebuah pesawat adalah wajar. Pengukuran sudut dilakukan dengan fungsi atan2 (arc tan dengan dua variabel: y dan x).

2.12. *Great Circle Distance*

Great circle distance adalah salah satu cara perhitungan dua titik di permukaan bumi yang memasukkan unsur lingkaran (bola) bumi sebagai dasar perhitungan.



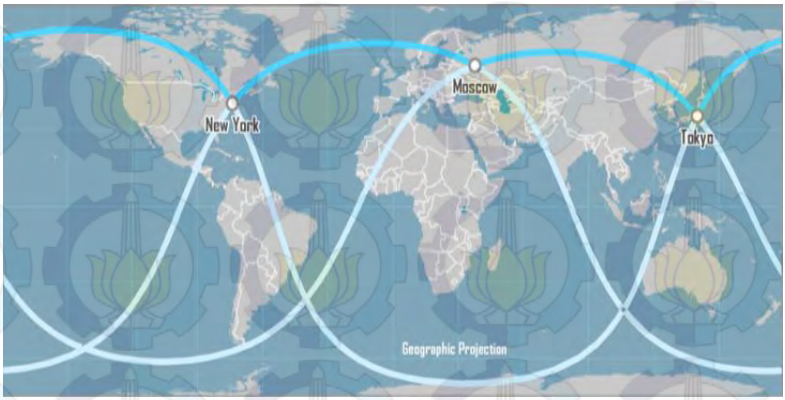
Gambar 2-8 Contoh Proyeksi Ortografik sebagai dasar perhitungan Great Circle Distance

Perhitungan GCD sedikit berbeda dengan *euclidean distance*, karena GCD menghitung jarak terdekat dua titik melalui selubung dalam (mendekati nilai keliling) pada bola bumi, sementara

euclidean distance hanya menghitung jarak dua titik pada bidang dua dimensi tanpa menghitung lengkung (*arc*) bumi. Setiap derajat pada perhitungan GCD, setara dengan 111.23 KM di darat [16]. Perhitungan dengan great circle distance ditunjukkan persamaan **(2.1)**

$$D = \cos^{-1}((sin\ a\ sin\ b) + (cos\ a\ cos\ b\ cos\ |\Delta long|)) \quad \textbf{(2-1)}$$

D adalah jarak dua titik dengan metode GCD, a dan b merupakan nilai lintang (*latitude*), sementara $|\Delta long|$ adalah nilai absolut dari perbedaan bujur (*longitude*). Contoh perbedaan proyeksi ortografis dan geografis ditunjukkan pada **Gambar 2-8** dan **Gambar 2-9.**



Gambar 2-9 Contoh Proyeksi Geografis untuk Rute yang sama dengan Gambar 2-8

2.13. Python

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif [17].

Python mendukung berbagai teknik pemrograman, utamanya namun tidak dibatasi pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Salah satu fitur yang tersedia pada python adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya, python umumnya digunakan sebagai bahasa skrip meski pada praktiknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakukan dengan menggunakan bahasa skrip. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi.

Saat ini kode python dapat dijalankan di berbagai platform sistem operasi, beberapa diantaranya adalah:

1. Linux/Unix
2. Windows
3. Mac OS X
4. Java Virtual Machine
5. OS/2
6. Amiga
7. Palm
8. Symbian (untuk produk-produk Nokia)

Python didistribusikan dengan beberapa lisensi yang berbeda dari beberapa versi. Namun pada prinsipnya Python dapat diperoleh dan dipergunakan secara bebas, bahkan untuk kepentingan komersial. Lisensi Python tidak bertentangan baik menurut definisi Open Source maupun General Public License (GPL)

Beberapa fitur yang dimiliki Python adalah:

1. memiliki kepustakaan yang luas; dalam distribusi Python telah disediakan modul-modul 'siap pakai' untuk berbagai keperluan.
2. memiliki tata bahasa yang jernih dan mudah dipelajari.

3. memiliki aturan layout kode sumber yang memudahkan pengecekan, pembacaan kembali dan penulisan ulang kode sumber.
4. memiliki sistem pengelolaan memori otomatis (garbage collection, seperti java)
5. modular, mudah dikembangkan dengan menciptakan modul-modul baru; modul-modul tersebut dapat dibangun dengan bahasa Python maupun C/C++.
6. memiliki fasilitas pengaturan penggunaan ingatan komputer sehingga para pemrogram tidak perlu melakukan pengaturan ingatan komputer secara langsung.
7. memiliki banyak fasilitas pendukung sehingga mudah dalam pengoperasiannya.

2.14. *Network Time Protocol* (NTP)

Secara umum, *Network Time Protocol* (NTP) adalah protokol untuk melakukan sinkronisasi sistem waktu (*clock*) pada komputer terhadap sumber yang akurat, melalui jaringan intranet atau internet. Terdapat beberapa situs NTP "Stratum 1" (situs NTP dengan sumber waktu dari *atomic clock*) and "Stratum 2" (situs NTP dengan sumber waktu dari situs NTP lain, dengan sedikit penurunan tingkat akurasi) yang dapat digunakan oleh publik [18].

BAB III
PERANCANGAN

Pada bab ini dibahas mengenai observasi, skenario studi kasus yang dikehendaki, perancangan metode berdasarkan studi kasus, serta perancangan skema aliran data. Metode yang digunakan adalah pohon keputusan.

3.1. Observasi

Sebelum menyusun rancangan skema aliran data yang menghubungkan >1 *station*, telah dilakukan observasi dan proses *logging* (pencatatan) data pada dua *stadion* yang berjalan masing-masing secara mandiri selama $\pm$ 4 bulan (November 2013 – Februari 2014). Dalam pengamatan tersebut, tampak perbedaan utama, yakni cakupan antara 2 *station* pada waktu yang sama, seringkali berbeda. Baik dalam hal kuantitas (banyak pesawat yang tertangkap), maupun kualitas (*field* yang terisi).

Terdapat beberapa kasus atau kondisi yang ditemukan selama observasi, antara lain:

1. Terdapat tipe-tipe pesawat tertentu yang tidak terdeteksi ADS-B, terutama pesawat dengan tahun pembuatan dibawah tahun 2000.
2. Terdapat tipe-tipe pesawat tertentu yang tidak memiliki informasi lokasi, hanya memuat informasi ketinggian dan arah saja.
3. Masih terdapat pesawat yang tidak valid (tidak ada penerbangan yang dimaksud pada keadaan *real*), yang dimungkinkan karena kesalahan penerimaan atau penerimaan yang kurang sempurna.
4. Kualitas perangkat SDR, *port* USB, jenis dan skala antena, mempengaruhi jangkauan (luasan) penerimaan sinyal.

Beberapa indikator validitas data yang dapat disimpulkan selama observasi ini, antara lain:

1. Aspek valid/*invalid* data: altitude
   Pada pesawat-pesawat tertentu yang tidak mengirimkan data ADS-B secara penuh, altitude yang terisi adalah indikator bahwa data/*record* tersebut valid. Contoh kasus real untuk indikator ini adalah data yang dikirimkan oleh armada Bombardier CRJ-1000 Garuda Indonesia dan beberapa armada Boeing *Classic* Sriwijaya Air.
2. Aspek valid/*invalid* data: variabel-variabel terisi
   Variabel yang terisi pada data-data valid, cenderung sama, minimal sekitar 20 variabel. Sementara data data yang kurang/tidak valid, cenderung terisi secara acak, dengan banyak variabel terisi bervariasi namun dibawah 20 variabel. 20 Variabel terisi yang menandakan validitas data, dapat dilihat pada **Tabel 3-1**.
3. Aspek validitas identitas: identitas pesawat
   Identitas pesawat, dalam hal ini diwakili oleh ICAO Hex, adalah salah satu indikator validitas data. Beberapa data yang masuk, tercatat menggunakan identitas yang tidak dikenal.
4. Aspek redundansi data: Data dikelompokkan berdasarkan identitas pesawat
   Record variabel yang diterima dari >1 *station/node* perlu diasosiasikan dengan identitas pesawat. Ini untuk menghindari kemungkinan data yang sama diterima oleh kedua *station*, kemudian memenuhi memori sistem. *Record* yang persis sama cukup dicatat sebanyak satu kali.

Tabel 3-1 Daftar 20 Variabel Terisi

No	Urutan pada *record*	Variabel	Keterangan
1	0	AA	**ICAO Hex** Identitas pesawat
2	1	CAT	**Kategori Pesawat**
3	2	CS	**Callsign** Nama panggil pesawat, biasanya memuat registrasi pesawat, atau nomor penerbangan.
4	3	FS	**Flight Status**
5	4	CA	**Capability**
6	6	UM	**Utility Message**
7	7	ID / SQUAWK	Kode kondisi penerbangan (*takeoff* / *landing* / *cruising* / *emergency*)
8	10	NIC	**Navigation Integrity Category**
9	11	NUCR	**Navigation Uncertainty Category**
10	12	FL	**Flight Level** Ketinggian pesawat dalam ekuivalen 100 kaki
11	13	AC	**Altitude Code** Ketinggian pesawat dalam ekuivalen 25 kaki
12	19	VR	**Vertical Rate** Kecepatan vertikal pesawat, dalam ekuivalen 25 kaki
13	22	TT	**True Track** Arah pesawat saat ini

No	Urutan pada *record*	Variabel	Keterangan
14	23	HDG	**Heading** Setting arah pesawat
15	25	IAS	**Indicated Airspeed**
16	26	TAS	**True Airspeed**
17	27	GS	**Ground Speed**
18	81	TIMESTAMP	
19	82	TIMEOUT	
20	83	STATION ID	

5. Aspek *continous* data: banyak variabel terisi
   Dari observasi yang dilakukan, data yang diterima oleh RTL1090 tidak selalu lengkap. Namun RTL1090 menutup lubang ketidaklengkapan tersebut dengan mengubah nilai-nilai variabel yang mengalami perubahan dari data baru yang masuk. Sehingga, secara desain, sebuah pesawat yang telah masuk ke RTL1090 akan setidaknya memiliki variabel terisi yang minimal sama pada setiap iterasinya (apabila variabel yang diupdate hanya variabel-variabel tertentu dan tidak ada penambahan variabel terisi).

Tabel 3-2 Contoh Record Data yang Tidak Konsisten

Detik ke	Nilai Koordinat		Ket
	Lat	Lon	
0	-5.90654	112.77527	Data inisial
1	-5.96827	112.83131	
2	-5.97628	112.83862	Konsisten terhadap [0..1]
3	-6.00328	112.86311	Konsisten terhadap [1..2]
4	-6.01129	112.87036	Konsisten terhadap [2..3]
5	-5.98896	112.85011	Tidak konsisten

6. Aspek *continous* data: check heading
   Arah pesawat (ditunjukkan pada variabel Heading dan True Track pada **Tabel 3-1**) perlu diperiksa pada setiap iterasi untuk menjamin data yang diproses bukan merupakan data yang terlambat masuk. Sebagai permisalan dapat dilihat pada **Tabel 3-2**. Dapat dilihat pada tabel tersebut, *record* data pada detik kelima secara tiba-tiba kembali ke koordinat lintang -5,98896, setelah sebelumnya secara konsisten berubah dari -5 ke -6. Begitu pula koordinat bujur yang menampakkan data pencilan yang tidak sesuai dengan urutan *record* yang telah ada sebelumnya.
7. Aspek *garbage* data: check area
   Secara acak (belum ditemukan pola pasti) pada pesawat-pesawat yang memiliki nilai *altitude* 0 (berada di darat), memiliki lokasi koordinat yang sangat jauh dari *receiver*. Hal ini merupakan *bug* pada sistem RTL1090 yang telah diketahui oleh pengembang sistem. Untuk itu, data dengan kondisi ini harus dibuang. Contoh data yang menunjukkan perubahan lokasi secara drastis ditunjukkan pada **Gambar 3-1**

```
1  Hex   |CAT|FLTNUM|      |squawk| Lat    | Lon      |   |FLevel|Alt|
2  8A0367:A-:LNI570:0:7:0:3:7156:-7.37132:112.73429:8:0:F008:825:0:
3  8A0367:A-:LNI570:0:7:0:3:7156:-7.37252:112.74132:8:0:F007:675:0:
4  8A0367:A-:LNI570:0:5:0:3:7156:-7.37294:112.74392:8:0:F006:625:0:
5  8A0367:A-:LNI570:0:5:0:3:7156:-7.37345:112.74705:8:0:F006:575:0:
6  8A0367:A-:LNI570:0:5:0:3:7156:-7.37359:112.74799:8:0:F006:550:0:
7  8A0367:A-:LNI570:0:5:0:3:7156:-7.37373:112.74894:8:0:F006:550:0:
8  8A0367:A-:LNI570:0:5:0:3:7156:-7.37410:112.75135:8:0:F005:500:0:
9  8A0367:A-:LNI570:0:5:0:3:7156:-7.37503:112.75724:8:0:F004:400:0:
10 8A0367:A-:LNI570:0:5:0:3:7156:-7.37769:112.77366:8:0:G---:0:0:0:
11 8A0367:A-:LNI570:0:4:0:3:7156:-7.37984:112.78686:8:0:G---:0:0:0:
12 8A0367:A-:LNI570:0:4:0:3:7156:82.61959:24.947695:8:0:G---:0:0:0:
13 8A0367:A-:LNI570:0:4:0:3:7156:82.61947:24.953776:8:0:G---:0:0:0:
14 8A0367:A-:LNI570:0:4:0:3:7156:82.61939:24.957994:8:0:G---:0:0:0:
```

Gambar 3-1 Contoh Data yang Menunjukkan Perubahan Lokasi Secara Drastis

3.2. Skenario Studi Kasus

Penggunaan SDR dan RTL1090 sebagai receiver ADS-B adalah salah satu cara termudah mendapatkan data ADS-B. SDR dan RTL1090 sendiri memiliki kelemahan, yakni cakupan wilayah yang kecil. Salah satu upaya memperbesar cakupan wilayah adalah dengan menambah receiver yang tersebar pada beberapa wilayah, kemudian dilakukan penggabungan data.

Data yang dikirimkan sebuah pesawat belum tentu diterima oleh semua *receiver*, ataupun bisa diterima dalam bentuk paket yang tidak lengkap. Data yang dijadikan acuan ini bisa disebabkan oleh:

1. Data hanya diterima oleh satu receiver.
2. Seluruh receiver menerima data yang sama.
3. Data yang diterima receiver lain tidak selengkap data acuan.
4. Data yang diterima dianggap tidak merepresentasikan kondisi sesungguhnya.
5. Data yang diterima receiver lain terlambat dikirimkan.

3.3. Skema Penyimpanan Data

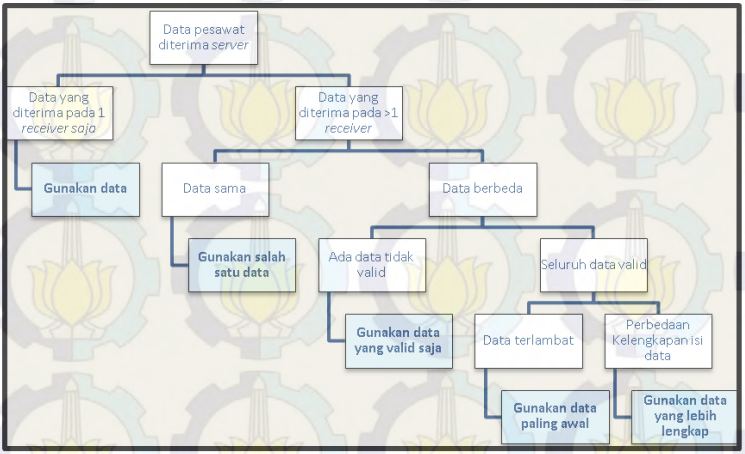
Dari observasi tersebut, ditentukan bahwa skema penyimpanan data pada program adalah sebagai berikut:

1. *List* of (valid) hex, menyimpan data-data ICAO Hex yang pernah diperiksa dan hasilnya valid. Memudahkan dan mengurangi waktu komputasi untuk pesawat-pesawat yang memang sering berada di atas langit Surabaya.
2. *List* of *invalid* hex, menyimpan data-data ICAO Hex yang pernah diperiksa dan hasilnya tidak valid. Memudahkan dan mengurangi waktu komputasi untuk pesawat-pesawat yang memang tidak valid.
3. Kelas ADSBData yang dimasukkan ke dalam LIST. Kelas ADSBData sendiri memuat:
    a. String planeHex (Identitas / ICAO Hex)

b. String flightNumber (nomor penerbangan)
c. *LIST* dataRecord (*list record* dari *receiver*)
d. *LIST* stationRecord (*list station* mana saja yang mengirim data sebuah pesawat)
e. Datetime lastUpdate
f. Int lastUsedDataRecord

3.4. Perancangan Metode

Berdasarkan skenario studi kasus penerimaan ADS-B dengan banyak *receiver/station*, dapat dibuat pohon keputusan sederhana seperti terlihat pada **Gambar 3-2**.

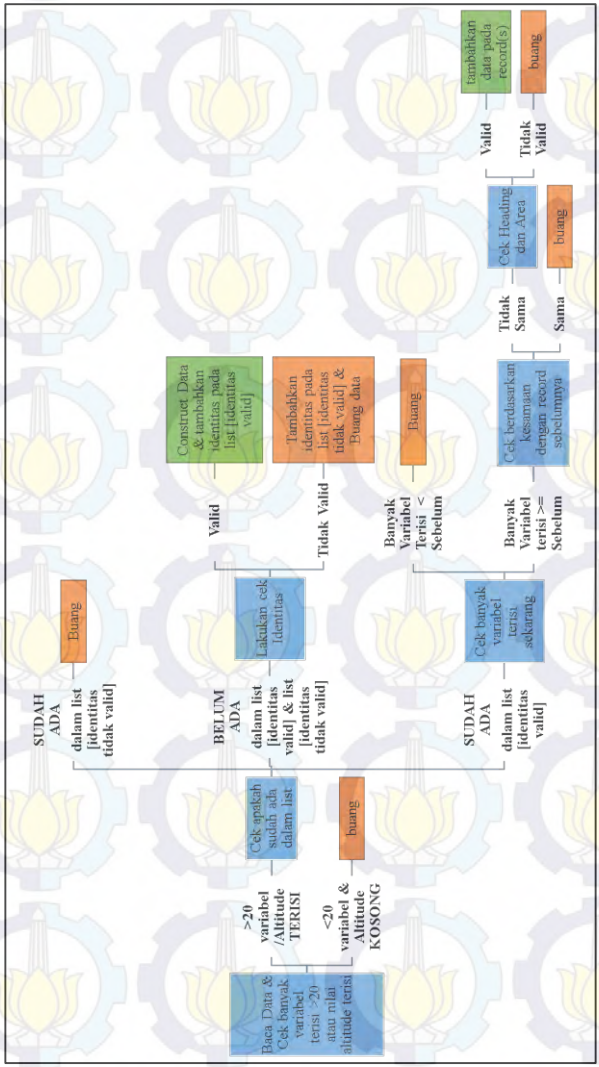


Gambar 3-2 Pohon Keputusan Sederhana

Temuan-temuan pada observasi tersebut, direpresentasikan pada *rules* yang lebih kompleks untuk diimplementasikan sebagai berikut:

1. Sebagai saringan awal, diperlukan pengecekan banyak variabel terisi minimal 20 atau nilai altitude terisi.
2. Untuk melakukan pemeriksaan validitas, digunakan tabel alokasi ICAO Hex, sehingga identitas pesawat

yang tidak tercakup pada tabel, otomatis dibuang dari sistem dan tidak perlu diproses lebih lanjut.



Gambar 3-3 *Decision Tree* dengan *Rules* yang Telah Ditentukan

3. Identitas yang sudah pernah diperiksa, tidak perlu diperiksa kembali. Baik yang valid maupun yang tidak.
4. Jika poin 1 dan 2 terpenuhi, lakukan pengecekan, apakah banyak variabel terisi adalah sama atau lebih banyak dari *record* sebelumnya, jika kurang dari sebelumnya, buang.
5. Jika poin 4 terpenuhi, cek apakah *record* yang sama telah ada dalam sistem. Jika ada, buang data yang baru.
6. Jika poin 5 terpenuhi, Cek konsistensi data dari heading dan area.
7. Jika poin 6 terpenuhi, masukkan data ke dalam sistem.

Ketujuh *rule* ini membentuk sebuah pohon keputusan, seperti ditunjukkan pada **Gambar 3-3**.

3.5. Pemrosesan Teks

Data yang masuk ke sistem adalah data teks. Selain baris identitas dan waktu pengiriman/penerimaan yang memang ditambahkan secara mandiri, setiap baris berisi 84 variabel sesuai **Tabel 8-2** yang dipisahkan oleh tanda titik dua / *colon* (":"). Contoh data dapat dilihat pada **Gambar 3-4**.

```
8A0367:A-:LNI570:0:7:0:3:7156:-
7.37132:112.73429:8:0:F008:825:0:0:75::0:-704:-
6::100:96:1:142:148:130:220::0:0::::::0::::0:0:0::0:2:0:
2:17:0:1:1:0:0:0:70:20:25:5:21:21:1:17:5:7:7:441:2367:5:
38403::34059:21:2043:21:107:27:3957:21:61:1399158695:58:
T-WARR7
```

Gambar 3-4 Contoh Data Teks Masukan Sistem

Data ini kemudian dipisahkan dengan fungsi text.split(":") pada bahasa Python sehingga dapat dihasilkan *list*/*array* yang dapat diakses, dibaca dan dimodifikasi dengan mudah. Contoh operasi teks yang dilakukan dalam bahasa Python ditunjukkan pada **Gambar 3-5**.

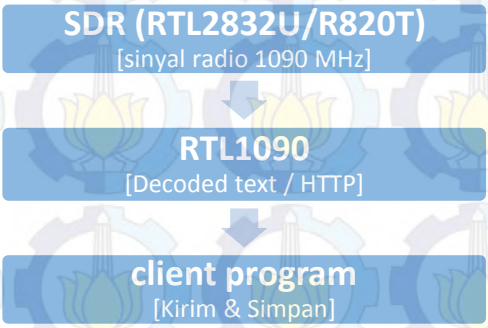
```
C:\Users\yoza>python
Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC
v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for
more information.
>>> dataADSB = "8A0367:A-:LNI570:0:7:0:3:7156:-
7.37132:112.73429:8:0:F008:825:0:0:75::0:-704:-
6::100:96:1:142:148:130:220::0:0:::::0:::0:0:0::0:2:0:
2:17:0:1:1
:0:0:0:70:20:25:5:21:21:1:17:5:7:7:441:2367:5:38403::340
59:21:2043:21:107:27:3957:21:61:1399158695:58:T-WARR7"
>>> type(dataADSB)
<type 'str'>
>>> dataADSBSplit = dataADSB.split(":")
>>> dataADSBSplit
['8A0367', 'A-', 'LNI570', '0', '7', '0', '3', '7156',
'-7.37132', '112.73429', '8', '0', 'F008', '825', '0',
'0', '75', '', '0', '-704', '-6', '', '100', '96',
 '1', '142', '148', '130', '220', '', '0', '0', '', '',
'', '', '', '0', '', '', '', '0', '0', '0', '', '0',
'2', '0', '2', '17', '0', '1', '1', '0', '0', '0',
'70', '20', '25', '5', '21', '21', '1', '17', '5', '7',
'7', '441', '2367', '5', '38403', '', '34059', '21',
'2043', '21', '107', '27', '3957', '21', '61', '139
9158695', '58', 'T-WARR7']
>>> type(dataADSBSplit)
<type 'list'>
>>> dataADSBSplit[0]
'8A0367'
>>> dataADSBSplit[1]
'A-'
>>> dataADSBSplit[82]
'58'
>>> dataADSBSplit[83]
'T-WARR7'
>>>
```

Gambar 3-5 Contoh Operasi Teks Masukan pada Bahasa Python

3.6. Perancangan Skema Aliran Data

3.6.1. Skema Aliran Data pada *Station*

Pada setiap *node*/*station* yang menerima sinyal ADS-B, diperlukan beberapa alat dan program.



Gambar 3-6 Skema Aliran Data pada Station

Data yang diterima oleh SDR pada frekuensi 1090 harus diterjemahkan terlebih dahulu oleh aplikasi RTL1090. Hasil yang diharapkan dari terjemahan ini adalah teks dalam protokol HTTP.

Kemudian, dibutuhkan aplikasi kecil dengan menggunakan untuk membaca teks terjemahan tersebut dan mengirimkannya ke server utama untuk diproses, serta melakukan pengarsipan pada media penyimpanan lokal *station* tersebut. Skema ini ditunjukkan pada **Gambar 3-6**.

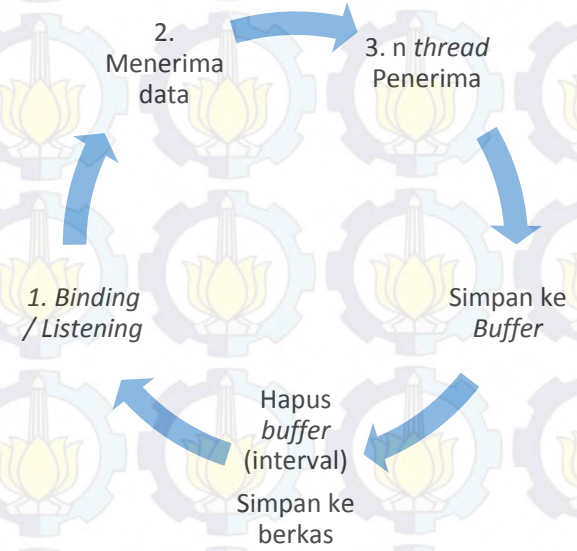
3.6.2. Skema Aliran Data pada Server

Pada server, terdapat dua aliran data, yakni receiver dan processor.

a. Skema Aliran Data pada Receiver

Pada *instance* server-receiver, dibutuhkan sebuah program yang mampu listen pada port tertentu dan menerima data dari >1 node secara simultan. Kemampuan menerima data secara simultan disebabkan oleh kondisi penerimaan data yang ada. Pada kondisi ideal, jangka waktu pengiriman data kurang dari 1 detik dari >1 receiver, meskipun pada kenyataannya, waktu tempuh data dari station ke server akan bervariasi.

Data yang diterima akan disimpan ke buffer, kemudian buffer dikosongkan setiap interval tertentu untuk dipindahkan ke sebuah *file*. Pertimbangan menggunakan sistem buffer dan pengosongannya, didasarkan atas kemungkinan pada saat bersamaan, ada >1 data dari >1 *station*



Gambar 3-7 Skema Aliran Data pada Server-Receiver

berbeda. Apabila masing-masing *thread* secara langsung memasukkan ke *file*, dimungkinkan terjadinya error karena dua *thread* mengakses sebuah berkas pada waktu yang bersamaan.

Urutan proses *listening-receiving-buffering* dan *clear buffer* ini akan berlangsung terus menerus hingga terjadi terminasi (program dihentikan dari luar). Skema secara umum dapat dilihat pada **Gambar 3-7.**

b. Skema Aliran Data pada Processor

Pada *instance server-processor*, dibutuhkan sebuah program yang memiliki kemampuan untuk membaca berkas secara parsial. Kemampuan membaca secara parsial ini dibutuhkan karena sistem logging harus tetap ada, tidak dihapus. Berkas yang dihasilkan oleh *server-receiver* akan dibaca mulai dari baris terakhir yang belum dibaca oleh sistem *processor*. Kemudian, data perlu dipisahkan, antara baris identitas dan baris *record*.

Setelah dilakukan identifikasi baris identitas, kemudian baris record diproses satu persatu. Sebuah *record* memiliki 84 variabel sesuai standar *record* yang dikirim oleh RTL1090. Pada tahap ini dilakukan seleksi validitas data. *Record* yang dianggap cukup valid adalah *record* yang memuat informasi ketinggian (*altitude*) atau *record* yang memuat setidaknya 20 variabel terisi. 20 variabel tersebut ditunjukkan pada **Tabel 3-1**.

Record yang diproses memiliki dua kemungkinan, yakni *record* merupakan lanjutan dari pesawat yang telah ada sebelumnya, ataupun *record*

pesawat baru. Apabila *record* memuat informasi pesawat yang belum ada, perlu inisialiasi data. Sebelum inisialisasi, perlu dilakukan kroscek atas identitas pesawat, sebagai seleksi kedua validitas data.

Jika *record* merupakan informasi lanjutan, maka diperlukan penyaringan tahap berikutnya yang menguji konsistensi data, antara lain:

1. Sebuah *record*, yang pada kondisi terbaik akan berisi 84 variabel, memiliki banyak variabel terisi sama atau lebih banyak dari *record* yang sebelumnya.
2. Jika sebuah *record* mengandung informasi lokasi (koordinat lintang dan bujur), maka kondisi lintang dan bujur ini akan dicocokkan dengan *heading* dan *true-track* pesawat.

BAB IV
IMPLEMENTASI

Pada bab ini dibahas implementasi dari rancangan metode yang telah dibahas pada Bab III. Bab ini meliputi implementasi node client, node server-receiver, kelas ADSBData,dan node server-processor.

4.1. Lingkungan Implementasi

Dalam implementasi algoritma digunakan perangkat-perangkat sebagai berikut:

4.1.1. Perangkat Keras

Spesifikasi perangkat keras yang digunakan saat implementasi ditunjukkan pada **Tabel 4-1**.

Tabel 4-1 Spesifikasi perangkat keras

Konfigurasi	Prosesor	Memori	Koneksi Internet
Station 1	Intel Core 2 Duo E4500 / 2,2 Ghz	2GB	1Mbps Speedy
Station 2	Intel Core 2 Duo E8400/ 3,0 Ghz	2GB	512Mbps Speedy
Station 3	Intel Atom D525 / 1,8 Ghz	2Gb	512Kbps Hypernet
Station 4	Intel Dual Core E5200 / 2,5 Ghz	2Gb	768Kbps Speedy
Server	Intel Core I5 - 2400 3,10 Ghz x 4Core	8Gb	Datacenter 5Mbps Crossnet
Node Ujicoba	Cloud Instance (Xeon E5-2630 / dibatasi 2,0 Ghz)	512Mb	Datacenter 1Gbps DigitalCloud Singapore

4.1.2. Perangkat Lunak

Berikut perangkat lunak yang digunakan dalam implementasi:

1. Windows XP / 7 (Station 1-4)
2. Ubuntu Linux (Server & Node Ujicoba)
3. RTL1090 (Station 1-4)
4. Network Time Protocol
5. Planeplotter dan FR24feed (sebagai pembanding)
6. Python 2.7.6
7. Text Editor (Notepad++ / Nano)

4.2. Implementasi *Node Client*

Implementasi node client dilakukan dalam bahasa python. Secara umum, node client melakukan koneksi ke *port* HTTP RTL1090 (31008), mengambil data yang diperlukan, kemudian mengirimkannya ke server dan mengarsipkannya. Kode sumber implementasi *Node Client* dapat dilihat pada **Kode Sumber 4-1** hingga **Kode Sumber 4-4**.

```
 1  #!/usr/bin/env python
 2  import socket
 3  import sys
 4  import os
 5  from datetime import datetime
 6  import time
 7  import cPickle as pickle
 8  import re
 9  ID = "T-WARR5"                  #Identitas Node
10
11  HOST = '127.0.0.1'              #IP Node
12  GET = '/data2'                  #folder RTL1090
13  PORT = 31008                    #Port RTL1090
14
15  HOST_SERVER = '116.68.248.195'
16  PORT_SERVER = 9876
```

Kode Sumber 4-1 Implementasi Node Client Bagian Pertama

```
17
18  content = ""
19  startdate= datetime.now()
20  try:
21    while 1:
22      startdate2= datetime.now()
23      try:
24        sock = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
25        sock_SERVER =
    socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
26      except socket.error, msg:
27        sys.stderr.write("[ERROR] %s\n" %
    msg[1])
28        sys.exit(1)
29
30      try:
31        sock.connect((HOST, PORT))
32        sock_SERVER.connect((HOST_SERVER,
    PORT_SERVER))
33      except socket.error, msg:
34        sys.stderr.write("[ERROR] %s\n" %
    msg[1])
35        sys.exit(2)
36
37      content +=
    datetime.now().strftime("##%d%m%y##%H%M%S.%f#
    #\n")
38
39      sock.send("GET %s HTTP/1.0\r\nHost:
    %s\r\n\r\n" % (GET, HOST))
40
41      data = sock.recv(102400)
42      data = sock.recv(102400)
43      sock.close()
44
45      for datanew in data.split("\n")[:-1]:
46        content+=datanew+"\n"
47
```

Kode Sumber 4-2 Implementasi Node Client Bagian Kedua

```
48      print "Getting adsb-data. press ctrl-c to
    exit"
49
50      f = open ('log_adsb_'+
    datetime.now().strftime("%y%m%d") +'.txt',
    'a')
51      f.write(content)
52      f.close()
53
54      #send length to server
55      cont = content.split("\n")
56      for c in cont:
57        c+="\r\n"
58      pickled_string = pickle.dumps(cont)
59      sock_SERVER.send(str(
60  len(pickled_string)))
        recv_SERVER=sock_SERVER.recv(1024)
61
62      while(recv_SERVER!=str(
    len(pickled_string))):
63        recv_SERVER=sock_SERVER.recv(1024)
64
65      sock_SERVER.send(ID)
66      recv_SERVER=sock_SERVER.recv(1024)
67      while(recv_SERVER!=ID):
68        recv_SERVER=sock_SERVER.recv(1024)
69
70      #divide by length
71      length_send = 102400
72      pickled_listed_string =
    [pickled_string[i:i+length_send] for i in
    range(0, len(pickled_string), length_send)]
73      for pickled_item in
    pickled_listed_string:
74          sock_SERVER.send(pickled_item)
75
76      content=""
77
```

Kode Sumber 4-3 Implementasi Node Client Bagian Ketiga

```
78      print "last logging : " +
    str(datetime.now())
79      if (sock):
80        sock.close()
81      if (sock_SERVER):
82        sock_SERVER.close()
83
84      runtime = float(str(datetime.now()-
    startdate2).split(":")[-1])
85      if (runtime < 0.5):
86        time.sleep(0.5-runtime)
87
88  except(KeyboardInterrupt,SystemExit,
    IndexError):
89    sock.close()
90    f = open ('log_adsb_'+
    datetime.now().strftime("%y%m%d") +'.txt',
    'a')
91    f.write(content)
92    f.write("Time Elapsed : " +
    str(datetime.now()-startdate) +"\n")
93    f.close()
94    print "\r\nShutting Down..."
```

Kode Sumber 4-4 Implementasi Node Client Bagian Keempat

4.3. Implementasi *Node Server-Receiver*

Implementasi node server-receiver dilakukan dalam bahasa python. Secara umum, node server-receiver menerima seluruh data yang dikirimkan lewat *socket*/*port* yang ditentukan dengan *multi-thread* ke buffer dan mengarsipkan buffer ke sebuah *file* pada interval tertentu. Implementasi node server-receiver dapat dibagi menjadi 3 bagian utama, yakni impor modul, fungsi dequeueToFile, serta fungsi utama.

4.3.1. Impor Modul dan Deklarasi Variabel

Memanggil modul pada python yang diperlukan dan melakukan deklarasi variabel, seperti ditunjukkan pada **Kode Sumber 4-5**

```
 1  # import required module
 2  import socket
 3  import select
 4  import sys
 5  import cPickle as pickle
 6  from datetime import datetime
 7
 8  mainQueue = []
 9  starttime =
    datetime.now().strftime("%y%m%d%H%M%S")
10  interval  = 3      # in seconds
```

Kode Sumber 4-5 Implementasi Pemanggilan Modul dan Inisialisasi Variabel pada Node Server-Receiver

4.3.2. Implementasi fungsi dequeueToFile

Fungsi dequeue digunakan untuk memindahkan buffer ke *file* pada interval tertentu, seperti ditunjukkan **Kode Sumber 4-6**.

```
 1  def doDeQueueToFile():
 2    global mainQueue
 3    content = ""
 4    i = len(mainQueue)
 5    for j in range(0,i-1):
 6      for k in range(0,len(mainQueue[j])-1):
 7        if type(mainQueue[j][k]) == type([]):
 8          for l in
    range(0,len(mainQueue[j][k])-1):
 9              content +=
    mainQueue[j][k][l]+"\r\n"
10        else:
11          content += mainQueue[j][k]+"\r\n"
12    f = open ('log/log_adsb_server_'+
    datetime.now().strftime("%y%m%d-%H") +'.txt',
    'a')
13    f.write(content)
14    f.close()
15    mainQueue = mainQueue[i:]
16    print str(i)+" records removed"
```

Kode Sumber 4-6 Implementasi Fungsi dequeueToFile pada Node Server-Receiver

4.3.3. Implementasi fungsi utama

Fungsi utama, menerima kiriman data, memasukkan ke buffer, dan menghapus buffer jika interval telah terpenuhi, seperti ditunjukkan pada **Kode Sumber 4-7** dan **Kode Sumber 4-8**.

```
print "ready to handle connections"
# creating socket server object, bind, and listen
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(('', 9876))
server_socket.listen(100)

# list to store accepted client
input_list = [server_socket]
try:
  while 1:
    # serving multiple client alternately; one socket in a time
    input, output, exception = select.select(input_list, [], [])
    for sock in input:
      # accept client and add it to list input
      if sock == server_socket:
        client_socket, client_address = server_socket.accept()
        input_list.append(client_socket)
      # handle sending and receiving message
      else:
        sock.settimeout(2)
        try:
          message = sock.recv(1024)
          sock.send(message)
          ID = sock.recv(1024)
          sock.send(ID)
```

Kode Sumber 4-7 Implementasi Fungsi Utama pada Node Server-Receiver Bagian Pertama

```
27            if (message):
28              msg = ''
29              msg2='1';
30              while(msg2):
31                msg2 = sock.recv(102400)
32                msg+=msg2
33
34  mainQueue.append(["\r\n**"+ID,datetime.now().
    strftime("##%d%m%y##%H%M%S.%f##")]+pickle.loa
    ds(msg))
35            else:
36              sock.close()
37              input_list.remove(sock)
38          except:
39            pass
40
41      if
    (datetime.now().strftime("%y%m%d%H%M%S") >
    str(int(starttime) + interval)):
42        print datetime.now()
43        starttime=
    datetime.now().strftime("%y%m%d%H%M%S")
44        doDeQueueToFile()
45
46  # when user press CTRL + C (in Linux), close
47  socket server and exit
48  except (KeyboardInterrupt, SystemExit):
      server_socket.close()
49    sys.exit(0)
```

Kode Sumber 4-8 Implementasi Fungsi Utama pada Node Server-Receiver Bagian Kedua

4.4. Implementasi Kelas ADSBData

Untuk mempermudah inisialisasi dan proses konsolidasi record, diperlukan kelas ADSBData. Kelas ADSBData mengimplementasikan beberapa fungsi, yakni, __init__, clearStationRecord, returnHistory, returnShortDict, returnAircraftKML, dan returnTrailKML.

4.4.1. Impor Modul dan Deklarasi Variabel

Memanggil modul pada python yang diperlukan dan melakukan deklarasi variabel, seperti ditunjukkan pada **Kode Sumber 4-9**.

```
1  from datetime import datetime, timedelta
2  import math
3  import os
4
5  class ADSBData:
```

Kode Sumber 4-9 Implementasi Impor Modul dan Deklarasi Variabel pada Kelas ADSBData

4.4.2. Implementasi Fungsi __init__

Fungsi init digunakan untuk proses inisialisasi data, seperti terlihat pada **Kode Sumber 4-10**.

```
1    def __init__(self, planeHex,
     flightNumber="", dataRecord=[],
     station="INIT" ):
2
3        self.planeHex     = planeHex.upper()
4        self.flightNumber = flightNumber.upper()
5        self.dataRecord   = []
6        self.stationRecord       = []
7        self.lastUpdate   = datetime.now()
8        self.lastUsedDataRecord = -1
9        self.stationRecord.append(station)
10       self.dataRecord.append(dataRecord)
```

Kode Sumber 4-10 Implementasi fungsi __init__ pada kelas ADSBData

4.4.3. Implementasi Fungsi clearStationRecord

Fungsi clearStationRecord digunakan untuk proses hapus record station pada sebuah record, apabila record dihilangkan, seperti terlihat pada **Kode Sumber 4-11**.

```
1    def clearStationRecord(self):
2      while (len(self.stationRecord)>1):
3        self.stationRecord.pop(0)
4      pass
```

Kode Sumber 4-11 Implementasi Fungsi clearStationRecord pada Kelas ADSBData

4.4.4. Implementasi Fungsi returnHistory

Fungsi returnHistory digunakan untuk menghasilkan komponen histori *record* pesawat dengan standar JSON, seperti terlihat pada **Kode Sumber 4-12**.

```
1    def returnHistory(self):
2      if len(self.dataRecord)>1 :
3        return ', "history" :' +
   str(self.dataRecord).replace("'",'"')
4      else:
5        return ""
```

Kode Sumber 4-12 Implementasi Fungsi returnHistory pada Kelas ADSBData

4.4.5. Implementasi Fungsi returnShortDict

Fungsi returnShortDict digunakan untuk menghasilkan komponen informasi dasar pesawat dengan standar JSON, seperti ditunjukkan pada **Kode Sumber 4-13** hingga **Kode Sumber 4-15**.

```
1  def returnShortDict(self):
2      if len(self.dataRecord)>1 :
3        curValidPos = 1
4
5        if(self.dataRecord[-1][4] != ""):
6          curSquawk = self.dataRecord[-1][4]
7        else:
8          curSquawk = "----"
9
10       if(str(self.flightNumber) != ""):
11         curFlightNum=self.flightNumber
```

Kode Sumber 4-13 Implementasi Fungsi returnShortDict pada kelas ADSBData Bagian Pertama

```
12          else:
13            curFlightNum = 'N/A'
14          if(self.dataRecord[-1][5] != ""):
15            curLat = self.dataRecord[-1][5]
16          else:
17            curLat = 0.0
18            curValidPos = 0
19
20          if(self.dataRecord[-1][6] != ""):
21            curLon = self.dataRecord[-1][6]
22          else:
23            curLon = 0.0
24            curValidPos = 0
25
26          if(self.dataRecord[-1][10] != ""):
27            curAlt = self.dataRecord[-1][10]
28          else:
29            curAlt = '"N/A"'
30
31          if(self.dataRecord[-1][16] != ""):
32            curVRate = self.dataRecord[-1][16]
33          else:
34            curVRate = '"N/A"'
35
36          if(self.dataRecord[-1][19] != ""):
37            curTrack = self.dataRecord[-1][19]
38          else:
39            curTrack = '"N/A"'
40
41          if(self.dataRecord[-1][24] != ""):
42            curSpeed = self.dataRecord[-1][24]
43          else:
44            curSpeed = '"N/A"'
45
```

Kode Sumber 4-14 Implementasi Fungsi returnShortDict pada Kelas ADSBData Bagian Kedua

```
46          return '"hex":
"'+str(self.planeHex)+'", "squawk": "'+
str(curSquawk) + '", "flight": "'+
str(curFlightNum) +'", "lat": '+ str(curLat)
+ ', "lon": '+ str(curLon) +',
"validposition": '+ str(curValidPos) +',
"altitude": '+ str(curAlt) +', "vert_rate":
'+ str(curVRate) +', "track": '+
str(curTrack) +', "validtrack": 1, "speed":
'+ str(curSpeed) +', "messages":
'+str(len(self.dataRecord))+', "seen":
'+str(math.ceil((datetime.now()-
self.lastUpdate).total_seconds()))+',
"station": "'+str(self.stationRecord[-1])+'"'
47      else:
48        return ""
```

Kode Sumber 4-15 Implementasi Fungsi returnShortDict pada Kelas ADSBData Bagian Ketiga

4.4.6. Implementasi Fungsi returnAircraftKML

Fungsi returnAircraftKML digunakan untuk menghasilkan komponen informasi dasar pesawat dengan standar KML. Kode sumber fungsi returnAircraftKML ditunjukkan pada **Kode Sumber 4-16** dan **Kode Sumber 4-17**.

```
1   def returnAircraftKML(self):
      if len(self.dataRecord)>1 :
        curValidPos=1
2       if(str(self.flightNumber) != ""):
3         curFlightNum=self.flightNumber
4       else:
5         curFlightNum = 'N/A'
6
7       if(self.dataRecord[-1][5] != ""):
8         curLat = self.dataRecord[-1][5]
9       else:
10        curLat = 0.0
11        curValidPos = 0
12      if(self.dataRecord[-1][6] != ""):
```

Kode Sumber 4-16 Implementasi Fungsi returnAircraftKML pada Kelas ADSBData Bagian Pertama

```
13          curLon = self.dataRecord[-1][6]
14        else:
15          curLon = 0.0
16          curValidPos = 0
17        curReg = ""
18        curType = ""
19        if(self.dataRecord[-1][10] != ""):
20          curAlt = self.dataRecord[-1][10]
21        else:
22          curAlt = '"N/A"'
23          curValidPos = 0
24        if(self.dataRecord[-1][9] != ""):
25          curFLevel = self.dataRecord[-1][9]
26        else:
27          curFLevel = '"N/A"'
28        if(self.dataRecord[-1][19] != ""):
29          curTrack = self.dataRecord[-1][19]
30        else:
31          curTrack = '"N/A"'
32          curValidPos = 0
33        if (curValidPos ==1):
34          return
    "<Placemark><description>Flight : " +
    str(curFlightNum) + "\nReg : " + str(curReg)
    + "\nHex : " + str(self.planeHex) + "\nType :
    " + str(curType) + "\nFlt Level : " +
    str(curFLevel) + "</description><name>" +
    curFlightNum + " " + str(curReg) + " " +
    self.planeHex + " " + str(curType) + " " +
    str(curFLevel) + "</name><styleUrl>#mystyle"
    + str(int(float(curTrack)/5)).zfill(2) +
    "</styleUrl><visibility>1</visibility><Point>
    <altitudeMode>absolute</altitudeMode><coordin
    ates>" + str(curLon) + "," + str(curLat) +
    "," + str(int(float(curAlt)*0.3048)) +
    "</coordinates></Point></Placemark>"
35        else:
36              return ""
```

Kode Sumber 4-17 Implementasi Fungsi returnAircraftKML pada Kelas ADSBData Bagian Kedua

4.4.7. Implementasi Fungsi returnTrailKML

Fungsi returnAircraftKML digunakan untuk menghasilkan komponen posisi historis pesawat dengan standar KML, seperti ditunjukkan pada **Kode Sumber 4-18**.

```
 1  def returnTrailKML(self):
 2     if len(self.dataRecord)>1 :
 3       listOfCoordinate = []
 4       if(str(self.flightNumber) != ""):
 5        curFlightNum=self.flightNumber
 6       else:
 7        curFlightNum =self.planeHex
 8
 9         for x in self.dataRecord:
10          if((x[6] != "") and (x[5] != "") and
    (x[10] != "")):
11            coordInfo =
    str(x[6]+","+x[5]+","+str(int(float(x[10])*0.
    3048)))
12            if (coordInfo not in
    listOfCoordinate):
13             listOfCoordinate.append(coordInfo)
14        if len(listOfCoordinate)>1:
15          strReturn = "<Placemark> <name>" +
    curFlightNum + "-trail</name>
    <styleUrl>#mystyle72</styleUrl>
    <visibility>1</visibility> <LineString>
    <extrude>0</extrude>
    <tessellate>1</tessellate>
    <altitudeMode>absolute</altitudeMode>
    <coordinates> "
16          for i in listOfCoordinate:
17            strReturn+=i+" "
18          strReturn+= "</coordinates>
    </LineString> </Placemark>"
19          return strReturn
20        else:
21                return ""
```

Kode Sumber 4-18 Implementasi Fungsi returnTrailKML pada Kelas ADSBData

4.5. Implementasi *Node Server-Processor*

Implementasi *node server-processor* dilakukan dalam bahasa python. Secara umum, *node server-processor* membaca file yang dihasilkan oleh *Node Server-Receiver*. Baris yang dibaca merupakan baris yang belum pernah dibaca sebelumnya. Proses pencarian baris yang belum pernah dibaca ini berada pada fungsi returnLinesAfter Hasil data tersebut menjadi masukan untuk fungsi processData

Proses validasi data terdapat pada fungsi checkHexValidity, checkByArea dan checkByHeading. Terdapat fungsi createJSON dan createJSONComplete untuk menghasilkan *file* JSON, serta createAircraftKML dan createAircraftTrailKML untuk menghasilkan *file* KML. Adapun fungsi removeDataFromMainList juga dibutuhkan agar data yang sudah tidak diperbarui oleh seluruh *station* tidak memenuhi memori dan hasil baik JSON maupun KML. Terakhir, keseluruhan fungsi akan dijalankan oleh fungsi utama.

4.5.1. Impor Modul dan Deklarasi Variabel

Memanggil modul pada python yang diperlukan dan melakukan deklarasi variabel. Kode sumber implementasi fungsi ini ditunjukkan pada **Kode Sumber 4-19** dan **Kode Sumber 4-20**.

```
1  from ADSBData import ADSBData
2  from datetime import datetime
3  from traceback import format_exc
4  import math
5  import time as ttime
6  import sys
7  lastLine= 1
8  time = datetime.now().strftime("%H")
9  curfile = 'log/log_adsb_server_'+
   datetime.now().strftime("%y%m%d-%H") +'.txt'
```

Kode Sumber 4-19 Implementasi Impor Modul dan Deklarasi Variabel pada Node Server-Processor Bagian Pertama

```
10  interval= 3    ##Interval to sleep
11  removeInterval= 600   ##Interval to remove data
12  from main list
13  list_of_hex= []
14  list_of_invalid_hex= []
15  list_of_data= []
```

Kode Sumber 4-20 Implementasi Impor Modul dan Deklarasi Variabel pada Node Server-Processor Bagian Kedua

4.5.2. Implementasi Fungsi returnLinesAfter

Membaca *file* yang dihasilkan oleh node server-receiver, dan mengembalikan baris yang belum pernah diproses sebelumnya, seperti ditunjukkan pada **Kode Sumber 4-21**.

```
 1  def returnLinesAfter(fname):
 2    global lastLine
 3    linenum = lastLine-1
 4    data = []
 5    i=0
 6    with open(fname) as f:
 7      for i, l in enumerate(f):
 8        if (i>=linenum):
 9          if not len(l.strip()) == 0 :
10            data.append(l)
11  if (lastLine == i+1):
12     return []
13    else:
14      lastLine = i+1
15            return data
```

Kode Sumber 4-21 Implementasi Fungsi returnLinesAfter pada Node Server-Processor

4.5.3. Implementasi Fungsi checkHexValidity

Melakukan validasi identitas pesawat berdasarkan alokasi ICAO. Kode yang merupakan angka heksadesimal, memiliki batas atas dan bawah. Untuk itu digunakan metode *binary search* agar pencarian menjadi efektif. Khusus untuk kelompok identitas Indonesia dan Malaysia, dikeluarkan dari proses *binary search*

untuk mempercepat komputasi. Kode sumber fungsi checkHexValidity dapat dilihat pada **Kode Sumber 4-22** hingga **Kode Sumber 4-24**

```
1  def checkHexValidity(planeHex):
2    validHexList = [[16384, 17407], [24576,
   28671], [32768, 197631], [204800, 208895],
   [212992, 218111], [221184, 225279],
   [229376, 233471], [253952, 258047],
   [262144, 266239], [270336, 274431],
   [278528, 282623], [286720, 290815],
   [294912, 295935], [303104, 304127],
   [311296, 315391], [327680, 331775],
   [344064, 348159], [360448, 364543],
   [368640, 369663], [376832, 380927],
   [385024, 386047], [393216, 394239],
   [401408, 405503], [409600, 413695],
   [425984, 430079], [434176, 435199],
   [442368, 446463], [450560, 454655],
   [458752, 462847], [475136, 476159],
   [483328, 484351], [491520, 495615],
   [499712, 500735], [507904, 511999],
   [524288, 528383], [540672, 544767],
   [557056, 561151], [565248, 569343],
   [573440, 577535], [589824, 593919],
   [606208, 607231], [614400, 615423],
   [622592, 623615], [630784, 634879],
   [638976, 643071], [647168, 648191],
   [655360, 692223], [696320, 697343],
   [700416, 701439], [704512, 708607],
   [712704, 716799], [720896, 724991],
   [729088, 733183], [737280, 741375],
   [745472, 749567], [753664, 757759],
   [761856, 765951], [770048, 771071],
   [778240, 782335], [786432, 790527],
   [794624, 798719], [802816, 806911],
   [811008, 815103], [819200, 823295],
   [827392, 828415], [835584, 836607],
   [851968, 917503], [1048576, 3407871],
```

Kode Sumber 4-22 Implentasi Fungsi checkHexValidity pada Node Server-Processor Bagian Pertama

```
..  [3407872, 4456447], [4456448, 4587519],
    [4587520, 4718591], [4718592, 4849663],
    [4849664, 4980735], [4980736, 5014527],
    [5021696, 5025791], [5029888, 5033983],
    [5046272, 5047295], [5054464, 5055487],
    [5062656, 5063679], [5242880, 7340031],
    [7536640, 7667711], [7700479, 8982527],
    [8994816, 8999935], [9003008, 9008127],
    [9011200, 9016319], [9437184, 12845055],
    [13107200, 13144063], [13148160, 13149183],
    [13156352, 13157375], [13160448, 13161471],
    [13164544, 13165567], [13172736, 13173759],
    [13631488, 15208447], [15220736, 15224831],
    [15237120, 15241215], [15253504, 15257599],
    [15269888, 15273983], [15286272, 15290367],
    [15466496, 16777215]]
 3    flag = -1
 4    intHex = int(planeHex.upper(),16)
 5    if (9043968 <= intHex <= 9076735):
    #Indonesia

 6      return True         #8A0000-8A7FFF
 7    elif (7667712 <= intHex <= 7700479):
    #Malaysia
 8      return True         #750000-757FFF
 9    else:
10      minNode = 0
11      maxNode = len(validHexList)
12      while(flag == -1):
13        Node = (maxNode+minNode)/2
14        tempvHLN0 = validHexList[Node][0]
15        tempvHLN1 = validHexList[Node][1]
16        if(intHex in range(tempvHLN0-
    1,tempvHLN1+1)):
17          flag = 1
18        elif((tempvHLN1 < intHex) and
    (validHexList[Node+1][0] > intHex)):
19          flag = 0
20        elif(intHex<validHexList[0][0]):
21          flag = 0
```

Kode Sumber 4-23 Implentasi Fungsi checkHexValidity pada Node Server-Processor Bagian Kedua

```
22        elif(intHex>validHexList[-1][1]):
23          flag = 0
24        elif(tempvHLN0 > intHex):
25          maxNode=Node
26        elif(tempvHLN1 < intHex):
27          minNode=Node
28      if (flag == 1):
29        return True
30      else:
31        return False
```

Kode Sumber 4-24 Implentasi Fungsi checkHexValidity pada Node Server-Processor Bagian Ketiga

4.5.4. Implementasi Fungsi checkByArea

Memeriksa apakah koordinat yang dikirimkan oleh station ADS-B masih dalam jarak 300 mil (1.5x standar paten ADS-B [19], 200 mil) terhadap station ADS-B itu sendiri, seperti terlihat pada **Kode Sumber 4-25** dan **Kode Sumber 4-26**.

```
1  def checkByArea(lat, lon, station):
2    try:
3      lat = float(lat)
4      lon = float(lon)
5    except:
6      return True
7
8    stationLat = [0, 0, 0, -7.279815, 0, -
   7.291593, -7.329213, -7.366824]
9    stationLon = [0, 0, 0, 112.797507, 0,
   112.758128, 112.741370, 112.775430]
10
11   try:
12     stationID = int(str(station)[-1])
13   except:
14     return True
15
```

Kode Sumber 4-25 Implementasi Fungsi checkByArea pada Node Server-Processor Bagian Pertama

```
16    if (stationID == 3) or (stationID ==
      5) or (stationID == 6) or (stationID ==
      7):
17      if(lat) and (lon):
18
19        StatLocationRad =
      [math.radians(stationLat[stationID]),
      math.radians(stationLon[stationID])]
20
21        DataLocationRad =
      [math.radians(lat), math.radians(lon)]
22
23        GCD  = math.degrees(math.acos((
      math.sin(StatLocationRad[0]) *
      math.sin(DataLocationRad[0])) +
      (math.cos(StatLocationRad[0]) *
      math.cos(DataLocationRad[0])
      *math.cos(math.fabs(DataLocationRad[1]-
      StatLocationRad[1])))))) * 69.11511768616
24
25        print GCD
26        if GCD <= 300 :
27          return True
28        else:
29          return False
30      else:
31        return True
32    else:
33      return False
34
```

Kode Sumber 4-26 Implementasi Fungsi checkByArea pada Node Server-Processor Bagian Kedua

4.5.5. Implementasi Fungsi checkByHeading

Memeriksa apakah koordinat yang dikirimkan oleh sebuah pesawat, merupakan koordinat yang memang mungkin dilalui pesawat tersebut dari posisi sebelumnya. Dilakukan validasi dengan menghitung sudut antara koordinat awal dan koordinat akhir, kemudian dicocokkan dengan variabel

HDG (*heading*) dan TT (*true-track*) dari data yang dikirim oleh pesawat tersebut, seperti terlihat pada **Kode Sumber 4-27**.

```
1  def checkByHeading(planeHexIndex, newLat,
   newLon):
2    if (newLat) and (newLon):
3      global list_of_data
4      lastDataRecord =
   list_of_data[planeHexIndex].dataRecord[-
   1][:21]
5      if (lastDataRecord[5]==newLat and
   lastDataRecord[6]==newLon):
6        return True
7      elif((lastDataRecord[19]) and
   (lastDataRecord[20]) and (lastDataRecord[6])
   and (lastDataRecord[5])):
8        estTrack =
   (math.degrees(math.atan2(float(newLon)-
   float(lastDataRecord[6]),float(newLat)-
   float(lastDataRecord[5])))+360)%360
9        return ( (math.fabs(estTrack-
   float(lastDataRecord[19]))<=60) or
   (math.fabs(estTrack-float(lastDataRecord[19])-
   360)<=60) or (math.fabs(estTrack-
   float(lastDataRecord[19])+360)<=60) or
   (math.fabs(estTrack-
   float(lastDataRecord[20]))<=60) or
   (math.fabs(estTrack-
   float(lastDataRecord[20])+360)<=60) or
   (math.fabs(estTrack-
   float(lastDataRecord[20])+360)<=60) or
   (math.fabs(estTrack-float(lastDataRecord[20])-
   360)<=60))
       else :
10       return True
11   else :
12     return True
13
```

Kode Sumber 4-27 Implementasi Fungsi checkByHeading pada Node Server-Processor

4.5.6. Implementasi Fungsi createJSON

Melakukan pembuatan *file* JSON yang memuat informasi dasar pesawat saja. Kode sumber fungsi createJSON dapat dilihat pada **Kode Sumber 4-28**.

```
1  def createJSON():
2    global list_of_data
3    counter = 0
4    b =
   open("dump1090/public_html/data.json","w")
5
6    b.write("[")
7    for f in range(0,len(list_of_data)):
       jsonOutput =
8  str(list_of_data[f].returnShortDict())
9      if (jsonOutput!= ""):
         if math.ceil((datetime.now()-
   list_of_data[f].lastUpdate).total_seconds() <
   60) :
10           if(counter==0):
11             b.write("\n{" + jsonOutput + "}")
12           else :
13             b.write(",\n{" + jsonOutput + "}")
14           counter+=1
15
16   b.write("\n]")
17   b.close()
```

Kode Sumber 4-28 Implementasi Fungsi createJSON pada Node Server-Processor

4.5.7. Implementasi Fungsi createJSONComplete

Melakukan pembuatan *file* JSON yang memuat informasi dasar pesawat berikut *record* historis pesawat. Kode sumber fungsi createJSONComplete dapat dilihat pada **Kode Sumber 4-29**.

```
1  def createJSONComplete():
2    global list_of_data
3    counter = 0
4    b =
   open("dump1090/public_html/datacomplete.json"
   ,"w")
5    b.write("[")
6    for f in range(0,len(list_of_data)):
7      jsonOutput =
   str(list_of_data[f].returnShortDict())
8      if(jsonOutput!=""):
9        if(counter==0):
           b.write("\n{" + jsonOutput +
10 str(list_of_data[f].returnHistory()) + "}")
11       else :
12         b.write(",\n{" + jsonOutput +
   str(list_of_data[f].returnHistory()) + "}")
13       counter+=1
14   b.write("\n]")
15   b.close()
```

Kode Sumber 4-29 Implementasi Fungsi createJSONComplete pada Node Server-Processor

4.5.8. Implementasi Fungsi createAircraftKML

Melakukan pembuatan *file* KML yang memuat informasi dasar pesawat. Implementasi fungsi ini dapat dilihat pada **Kode Sumber 4-30** dan **Kode Sumber 4-31**

```
1  def createAircraftKML():
2    global list_of_data
3    counter = 0
4    b = open("dump1090/public_html/aircraft.kml"
5  ,"w")
6    c = open("kmlheader.txt")
7    for i in c.readlines():
8      b.write(i)
9    c.close()
```

Kode Sumber 4-30 Implementasi Fungsi createAircraftKML pada Node Server-Processor Bagian Pertama

```
10    for f in range(0,len(list_of_data)):
        KMLOutput =
11  str(list_of_data[f].returnAircraftKML())
        if(math.ceil((datetime.now()-
    list_of_data[f].lastUpdate).total_seconds()
    <60) and KMLOutput!=""):
12          b.write(KMLOutput+"\n")
13    c = open("kmlfooter.txt")
14    for i in c.readlines():
15      b.write(i)
16    c.close()
17          b.close()
```

Kode Sumber 4-31 Implementasi Fungsi createAircraftKML pada Node Server-Processor Bagian Kedua

4.5.9. Implementasi Fungsi createAircraftTrailKML

Melakukan pembuatan *file* KML yang memuat informasi dasar pesawat berikut *record* historis lokasi dan ketinggian pesawat. Implementasi fungsi ini dapat dilihat pada **Kode Sumber 4-32** dan **Kode Sumber 4-33**.

```
1   def createAircraftTrailKML():
2     global list_of_data
3     counter = 0
4     b = open("dump1090/public_html/aircraft-
    trail.kml","w")
5     c = open("kmlheader.txt")
6     for i in c.readlines():
7       b.write(i)
8     c.close()
9     c = open("kmlheaderstyle.txt")
10    for i in c.readlines():
11      b.write(i)
12
```

Kode Sumber 4-32 Implementasi Fungsi createAircraftTrailKML pada Node Server-Processor Bagian Pertama

```
13    c.close()
14    for f in range(0,len(list_of_data)):
15      KMLOutput =
    str(list_of_data[f].returnAircraftKML())
16      if(math.ceil((datetime.now()-
    list_of_data[f].lastUpdate).total_seconds()
    <60) and KMLOutput!=""):
17        b.write(KMLOutput+"\n")
18        trailOutput =
    str(list_of_data[f].returnTrailKML())
19        if(trailOutput!=""):
20          b.write(trailOutput+"\n")
21    c = open("kmlfooter.txt")
22    for i in c.readlines():
23      b.write(i)
24    c.close()
25    b.close()
```

Kode Sumber 4-33 Implementasi Fungsi createAircraftTrailKML pada Node Server-Processor Bagian Kedua

4.5.10. Implementasi Fungsi processData

Melakukan gabungan proses inisialisasi dan pemrosesan record. Implementasi fungsi ini dapat dilihat pada **Kode Sumber 4-34** hingga **Kode Sumber 4-37**.

```
1  def processdata(theData):
2    global lastLine
3    curLine = 0
4    flag  = False
5    dataLen = len(theData)-1
6    curStation = ""
7
8  ##BAGIAN 1: Cari baris yang memuat identitas
   station
```

Kode Sumber 4-34 Implementasi Fungsi processData pada Node Server-Processor Bagian Pertama

```
  while(curLine<dataLen):
    if ((flag is False)or(theData[curLine][:2] == "**")) :
      while not(theData[curLine][:2] == "**"):
        curLine+=1
      curStation = theData[curLine][2:].split()[0]
      curLine+=3
      if curLine<dataLen:
        if (theData[curLine][:2] != "**"):
          flag = True
      else:
        curLine+=1
##BAGIAN 2: Inisialisasi pesawat yang belum ada pada list
    else:
      curHex = theData[curLine].split(":")[0]
      curFlightNum = theData[curLine].split(":")[2]
      curDataRecord = theData[curLine].split(":")[3:-1] + [curStation]
      #Cek 1: Apakah variabel terisi >=20 atau variabel altitude (ketinggian) terisi?
      if ((len(filter(None,curDataRecord))>=20) or (curDataRecord[10]!="")):
        #Cek 2: Apakah pesawat tidak ada dalam list valid maupun invalid?
        if (curHex not in list_of_hex) and (curHex not in list_of_invalid_hex):
          if(checkHexValidity(curHex)):
            if(curStation != ""):
              if(curFlightNum != ""):

#INISIALISASI w/o Station w/ Flight Number
                newADSBData = ADSBData(planeHex=curHex, dataRecord=curDataRecord, flightNumber=curFlightNum)
```

Kode Sumber 4-35 Implementasi Fungsi processData pada Node Server-Processor Bagian Kedua

```
36                  else:
37  #INISIALISASI w/o Station w/o Flight Number
38                      newADSBData = ADSBData(
    planeHex=curHex, dataRecord=curDataRecord)
39
40                  else:
41                    if(curFlightNum != ""):
42  #INISIALISASI w/ Station w/ Flight Number
43                      newADSBData = ADSBData(
    planeHex=curHex, flightNumber=curFlightNum,
    dataRecord=curDataRecord, station=curStation
    )
44                  else:
45  #INISIALISASI w/ Station w/o Flight Number
46                      newADSBData = ADSBData(
    planeHex=curHex, dataRecord=curDataRecord,
    station=curStation)
47
48  #Jika sudah diinisialisasi, masukkan ke LIST
49                  list_of_hex.append(curHex)
50                  list_of_data.append(newADSBData)
51
52  #Jika HEX tidak valid, masukkan ke daftar hex
    tidak valid
53              else:
54                list_of_invalid_hex.append(
    curHex)
55  #Jika HEX ada dalam daftar hex tidak
    valid,lewati
56            elif (curHex in list_of_invalid_hex):
57              pass
58
59  ##BAGIAN 3: Penambahan record pada pesawat
    yang sudah ada dalam list
60            else:
61              selectedData =
    list_of_data[list_of_hex.index(curHex)]
62          try:
63  #Cek apakah variabel terisi sama atau lebih
    banyak dari sebelumnya
```

Kode Sumber 4-36 Implementasi Fungsi processData pada Node Server-Processor Bagian Ketiga

```
64          if (len(filter(None,
    selectedData.dataRecord[-1])) <= len(
    filter(None,curDataRecord))):
65
66  #Jika variabel terisi sama atau lebih banyak,
    cek Heading dan area
67          if (checkByHeading(
    list_of_hex.index(curHex), curDataRecord[5],
    curDataRecord[6]) and
    checkByArea(curDataRecord[5],
    curDataRecord[6], curStation)):
68
69  #Jika lolos cek heading dan area, masukkan
    data
70              if (curStation in
    selectedData.stationRecord):
71          selectedData.stationRecord.remove(
    curStation)
                selectedData.stationRecord.append(
72  curStation)
                selectedData.lastUsedDataRecord =
    len(selectedData.dataRecord)
73              if curFlightNum != "":
                    selectedData.flightNumber=
    curFlightNum
74                  if(curDataRecord !=
75  selectedData.dataRecord[-1]):
76
77  selectedData.dataRecord.append(curDataRecord)
                    selectedData.lastUpdate =
78  datetime.now()
                     except:
79                       pass
80          curLine+=1
81          if (theData[curLine][:2] == "**"):
82            flag=False
83
```

Kode Sumber 4-37 Implementasi Fungsi processData pada Node Server-Processor Bagian Keempat

4.5.11. Implementasi Fungsi removeDataFromMainList

Melakukan proses hapus data dari list. Fungsi ini akan dipanggil setiap interval waktu tertentu. Implementasi fungsi ini dapat dilihat pada **.**

```
1  def removeDataFromMainList():
2    global list_of_data
3    global list_of_hex
4    global removeInterval
5    items_to_be_removed = []
6
7    for f in range(0,len(list_of_data)):
8        if(math.ceil((datetime.now()-
   list_of_data[f].lastUpdate).total_seconds() >
   removeInterval)):
9        items_to_be_removed.append(f)
10     else:
11       list_of_data[f].clearStationRecord()
12
13   if len(items_to_be_removed) > 0:
14     for x in reversed(items_to_be_removed):
15       list_of_data.pop(x)
16
```

Kode Sumber 4-38 Implementasi Fungsi removeDataFromMainList pada Node Server-Processor

4.5.12. Implementasi Fungsi Utama

Mengatur perilaku node server-processor dan mengulanginya sampai ada permintaan berhenti dari luar program, seperti terlihat pada **Kode Sumber 4-39** dan **Kode Sumber 4-40**

```
1  counter=-1
2  while 1:
3    try:
```

Kode Sumber 4-39 Implementasi Fungsi Utama pada Node Server-Processor Bagian Pertama

```
    counter+=1
    processdata(returnLinesAfter(curfile))
    createJSON()
    createJSONComplete()
    if(counter%10 ==0):
      createAircraftKML()
      createAircraftTrailKML()
    removeDataFromMainList()
    if  not  (datetime.now().strftime("%H")  ==
time):
      processdata(returnLinesAfter(curfile))
      curfile = 'log/log_adsb_server_'+
datetime.now().strftime("%y%m%d-%H") +'.txt'
      ltime = datetime.now().strftime("%H")
      lastline=1
  # when user press CTRL + C (in Linux), close
socket server and exit
  except (KeyboardInterrupt, SystemExit):
    sys.exit(0)

  except:
    print  '[ERR]  '+format_exc().split('\n')[-
2]
    pass
       ttime.sleep(0.5*interval)
```

Kode Sumber 4-40 Implementasi Fungsi Utama pada Node Server-Processor Bagian Kedua

BAB V
UJI COBA DAN EVALUASI

Pada bab ini, dibahas uji coba dan evaluasi implementasi algoritma.

5.1. Lingkungan Uji Coba

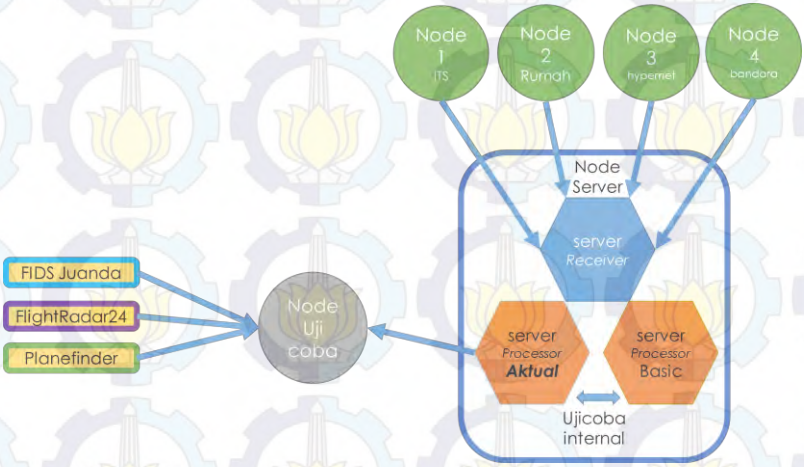
Studi kasus yang digunakan adalah wilayah Surabaya, atau dapat dianggap sebagai *aerodrome* bandara internasional Juanda (kode ICAO: WARR, kode IATA: SUB, koordinat lintang: S7,379999º, bujur: E112,786666º). Berdasarkan analisa diperlukan 4 *station* untuk mendapatkan data yang cukup lengkap, baik untuk fase *approach*, *landing*, *take-off*, dan *ground movement*.

Lokasi *receiver* pada studi kasus Bandara Juanda, ketinggian tempat (elevasi) beserta jarak masing-masing tempat ke Bandara Juanda dapat dilihat pada **Tabel 5-1**. Adapun posisi *receiver* terhadap Bandara Juanda secara relatif dapat dilihat pada **Gambar 5-1** dan skema uji coba dapat dilihat pada **Gambar 5-2**.

Tabel 5-1 Lokasi *Station* Uji Coba

No / Lokasi		*Lintang*	Bujur	Keting-gian	Jarak ke Juanda
		(º)	(º)	(m dpl)	(km)
1	Jl. Ngagel Jaya Tengah	S7,2915	E112,7581	+7	10,27
2	Informatika ITS	S7,2798	E112,7975	+15	11,14
3	HyperNet, Jemursari	S7,3292	E112,7413	+18	7,52
4	Jl. Bandara	S7,3668	E112,7754	+7	1,91

Gambar 5-1 Plot Lokasi *Station* pada Google Earth



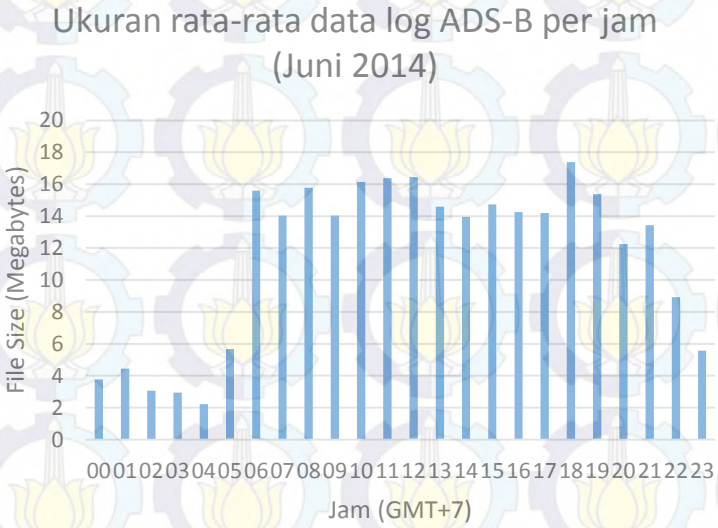
Gambar 5-2 Skema Uji Coba

Keempat station menggunakan perangkat keras dan perangkat lunak yang serupa, termasuk menggunakan versi RTL1090, Python, Planeplotter, dan FR24feed yang sama. Planeplotter dan

FR24feed digunakan untuk mengunggah data yang dihasilkan RTL1090 ke portal planefinder.com dan portal flightradar24.com agar bisa dijadikan pembanding validitas data.

5.2. Data Uji Coba

Pada kondisi ideal setiap *station* akan mengirimkan data setiap 0.5 detik. Pada bulan Juni 2014, rata-rata ukuran *file* per jam adalah 12Mb. Rata-rata ukuran *file* per hari adalah 261,4Mb. Grafik rata-rata ukuran file per jam pada bulan Juni 2014 dapat dilihat pada **Gambar 5-3**.



Gambar 5-3 Ukuran rata-rata data log ADS-B per jam (Juni 2014)

Data uji coba pembanding diperoleh dengan teknik *scrapping* pada FIDS (*Flight Information Display System*) Bandara Juanda, portal planefinder.com, dan portal flightradar24.com. Seperti disebutkan pada bagian 5.1, data yang dihasilkan RTL1090 juga diunggah ke

kedua portal tersebut, menjamin setidaknya ada data masukan yang sama untuk diukur validitas keluarannya.

Selain pengujian dengan hasil sistem lain, dilakukan pengujian mandiri, terhadap sebuah *node-processor* lain yang hanya mengolah data tanpa adanya metode pohon keputusan sebagai sarana penyaringan dan uji validitas data.

5.3. Skenario Uji Coba

Sebuah skenario diperlukan untuk melihat kebenaran data masukan dan keluaran, diperlukan pengujian terhadap data tersebut. Skenario pertama adalah pengujian data terhadap *ground-truth,* dalam hal ini adalah FIDS (*Flight Information Display System*) Bandara Juanda, Surabaya. Kemudian, untuk menguji kualitas data, diperlukan uji coba secara intern terhadap data mentah. Uji coba intern ini dilakukan dengan data yang telah diarsipkan. Sementara untuk menguji validitas data, digunakan uji coba terhadap keluaran sistem lain yang memiliki masukan yang sama. Uji coba dengan sistem lain ini, dilakukan secara *real-time*.

5.3.1. Skenario Uji Coba dengan Ground-truth

Pada skenario ini, dilakukan scrapping pada *web* FIDS Bandara Juanda Surabaya. Hasil *web scrapping* disimpan selama setidaknya 60 menit untuk mengakomodasi kemungkinan keberangkatan penerbangan yang terlebih dahulu hilang dari FIDS sebelum muncul di sistem yang dibuat. Hasil yang disimpan ini kemudian dibandingkan dengan keluaran sistem yang dibuat. Pengujian dilakukan mulai tanggal 10 Juni 2014 hingga 25 Juni 2014 (16 hari), dengan interval pengecekan setiap satu menit.

Secara umum, fungsi uji coba ini adalah memastikan bahwa data yang ditangkap oleh ADS-B *receiver* adalah perjalanan pesawat yang *real.*

5.3.2. Skenario Uji Coba Intern

Pada skenario ini, digunakan *file* terarsip mulai tanggal 05 Mei 2014 hingga 25 Juni 2014 (52 hari). Ujicoba yang akan dilakukan meliputi:

1. Perbandingan banyak pesawat valid-tidak valid beserta penyebabnya.
2. Perbandingan ukuran *file* JSON, mewakili penyaringan data yang tidak perlu.

5.3.3. Skenario Uji Coba Ekstern

Pada skenario ini, digunakan pembandingan hasil sistem yang dibuat, secara *realtime* dengan sistem serupa, yakni planefinder.com dan flightradar.com. Hasil dari planefinder.com dan flightradar.com merupakan hasil *web-scrapping*. Pengujian terhadap planefinder.com dan flightradar24.com dimulai tanggal 24 Juni 2014 hingga 26 Juni 2014 (3 hari) dengan interval pengecekan setiap satu menit.

5.4. Uji Coba

5.4.1. Uji Coba terhadap Ground Truth

Uji coba terhadap *ground truth* dapat dikatakan tidak maksimal. Hal ini dikarenakan keterbatasan sistem web FIDS Juanda yang hanya memuat keberangkatan dan kedatangan di Terminal 1 (domestik). Selain itu, web FIDS Juanda juga kurang stabil (sering *down*).

Sistem pengujian ini juga memiliki limitasi, bahwa data yang diproses dari FIDS merupakan data apa-adanya. Termasuk adanya penerbangan yang baru akan terbang dalam belasan jam ataupun keesokan harinya.

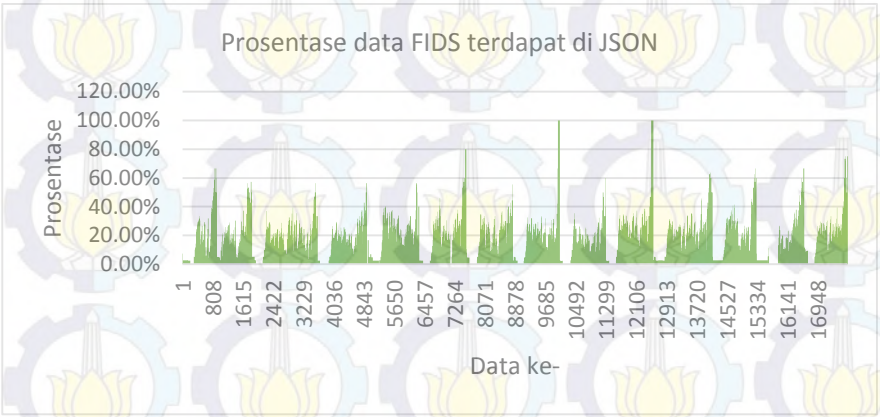
Uji coba dilakukan terhadap dua sisi, yakni, keseluruhan data di FIDS terhadap data di JSON, dan keseluruhan data di JSON terhadap data di FIDS. Kedua ujicoba ini mungkin menghasilkan angka pembilang (data yang valid) berbeda karena data disimpan dalam interval 1 jam. Banyak data yang berhasil didapatkan sebanyak 17.744 data selama tanggal 10 Juni 2014 hingga 25 Juni 2014 (16 hari), dengan interval pengecekan setiap satu menit. Statistik uji coba dapat dilihat pada **Tabel 5-2**.

Tabel 5-2 Statistik Kesamaan Data FIDS terhadap JSON dan sebaliknya

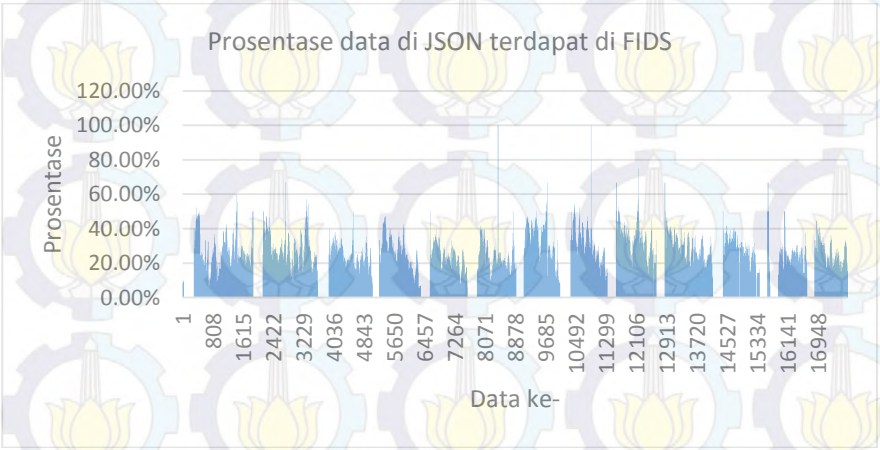
	JSON terhadap FIDS	**FIDS terhadap JSON**
Rata-rata (Termasuk #div/0)	17,77%	20,95%
Rata-rata (Tidak termasuk #div/0)	18,01%	20,98%
Modus	0,00 % (1.873 data)	0,00% (3.478 data)
Terendah	N/A % (243 data) 0,00 % (1.873 data)	N/A % (22 data) 0,00 % (3.478 data)
Tertinggi	100% (94 data)	100% (3 data)

Indikator kesamaan dilihat dari adanya nilai-nilai tidak nol, baik pada rata-rata maupun kondisi tertinggi. Apabila tidak ada kesamaan, maka nilai tertinggi dan rata-rata adalah 0.

Grafik kemunculan data FIDS pada data JSON dapat dilihat pada **Gambar 5-4**, sementara grafik kemunculan data JSON pada data FIDS dapat dilihat pada **Gambar 5-5**.



Gambar 5-4 Grafik Prosentase Kemunculan Data FIDS pada data JSON



Gambar 5-5 Grafik Prosentase Kemunculan Data JSON pada Data FIDS

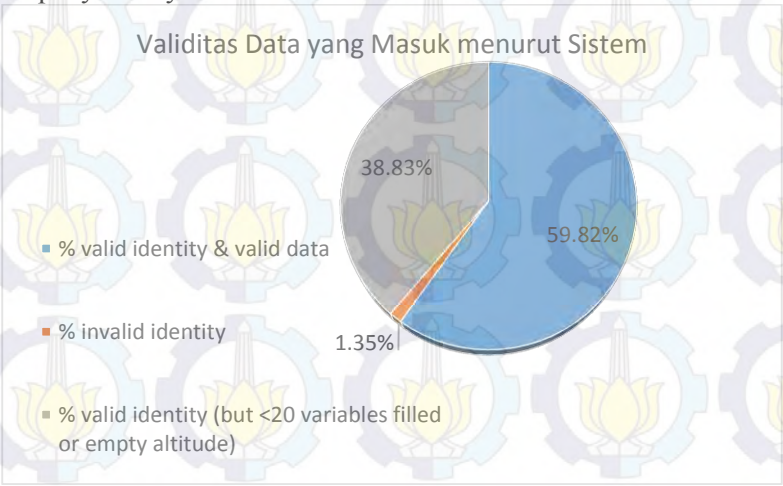
Terlihat pada **Gambar 5-4** dan **Gambar 5-5**, terdapat pola yang kurang lebih berulang setiap harinya. Data yang nilainya 0% atau N/A, umumnya terbentuk pada saat Bandara Juanda tidak beroperasi (Pk. 00.00-05.00 waktu lokal. atau Pk. 17.00-22.00 UTC). Puncak data berada pada jam paling ramai bandara, yakni Pk. 05.00-07.00 waktu lokal dan Pk. 17.00-20.00 waktu lokal.

Data dan grafik ini mampu mempresentasikan bahwa data yang diterima oleh *ADS-B receiver* memang merupakan bagian dari pergerakan *real* di sekitar Bandara Juanda, Surabaya.

5.4.2. Uji Coba Intern

Pada uji coba intern ini, digunakan 1.208 data yang tersedia (dari 1.248 data pada kondisi ideal). Dua uji coba intern yang dilakukan antara lain:

1. Perbandingan banyak pesawat valid-tidak valid beserta penyebabnya.

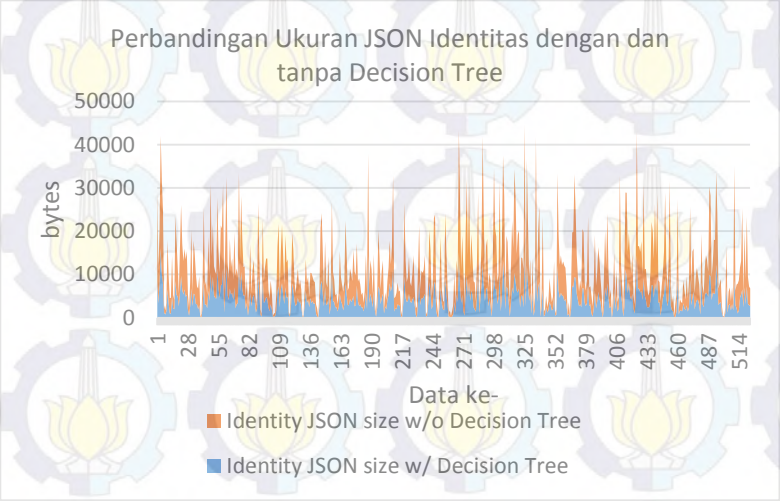


Gambar 5-6 Grafik Validitas Data yang Masuk menurut Sistem

Validitas pesawat dilihat dari dua hal, pertama adalah validitas identitas (ICAO Hex). Hal kedua adalah banyaknya variabel terisi minimal 20 atau nilai *altitude* tidak kosong. Apabila kedua syarat ini dipenuhi (relasi AND), maka data dianggap valid.

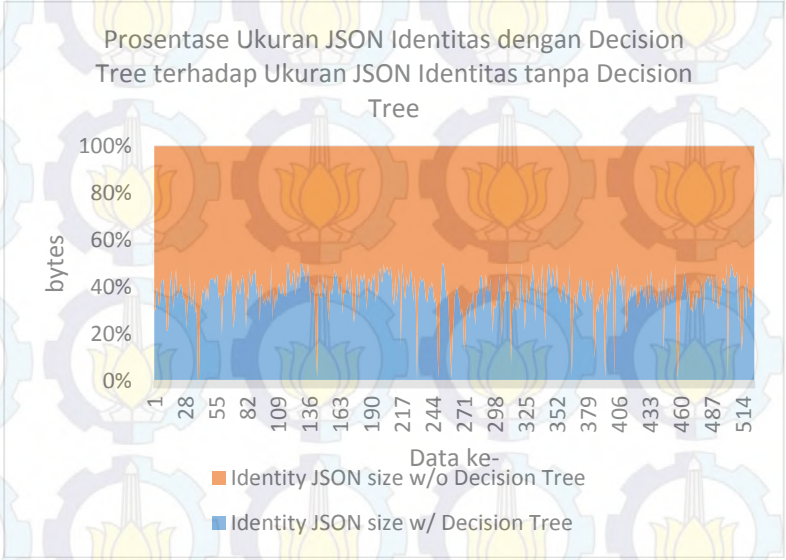
Data yang valid untuk uji coba validitas ini adalah sebanyak 524 data. Mayoritas data datang dengan identitas yang valid, namun memiliki variabel yang tidak terisi. Hal ini terlihat pada **Gambar 5-6**. Pada **Gambar 5-6** tersebut, keseluruhan data didapatkan dari sistem "*basic*" yang menerima data tanpa melakukan proses apapun terhadap data selain mengarsipkannya. Sedangkan sistem "aktual" mampu membedakan data-data yang valid dan tidak valid beserta alasannya (apakah tidak valid menurut identitas ataupun menurut konten)

2. Perbandingan ukuran *file* JSON, mewakili penyaringan data yang tidak perlu.



Gambar 5-7 Perbandingan Ukuran JSON Identitas dengan dan tanpa Decision Tree

Pada ujicoba ini, dilakukan pembedaan antara *node-receiver* yang menggunakan metode *decision tree* dan *node-receiver* yang hanya menerima saja tanpa menggunakan metode *decision tree*. Terdapat 524 data yang valid untuk uji coba ini.Hasil pembandingan dapat dilihat pada gambar **Gambar 5-7**. Sementara, rasio ukuran JSON identitas dengan menggunakan metode Decision Tree dan ukuran JSON identitas tanpa menggunakan metode Decision tree dapat dilihat pada **Gambar 5-8**.



Gambar 5-8 Prosentase Ukuran JSON Identitas dengan Decision Tree terhadap Ukuran JSON Identitas tanpa Decision Tree

Pada **Gambar 5-8** tersebut ditunjukkan bahwa rasio ukuran dengan menggunakan metode decision tree rata-rata dibawah 50 %. Sehingga data yang perlu diproses sistem lain menjadi lebih sedikit dengan validitas data yang lebih terjamin.

5.4.3. Uji Coba Ekstern

Pengujian terhadap dua portal yang memiliki layanan serupa, yakni flightradar24.com dan planefinder.com, dilakukan dengan cara mencocokkan data identitas pesawat di data JSON dan data dari portal-portal tersebut. Seluruh *node* yang mengirim data ke sistem yang dibuat, juga mengirim data ke FlightRadar24 (FR24) serta planefinder (PF).

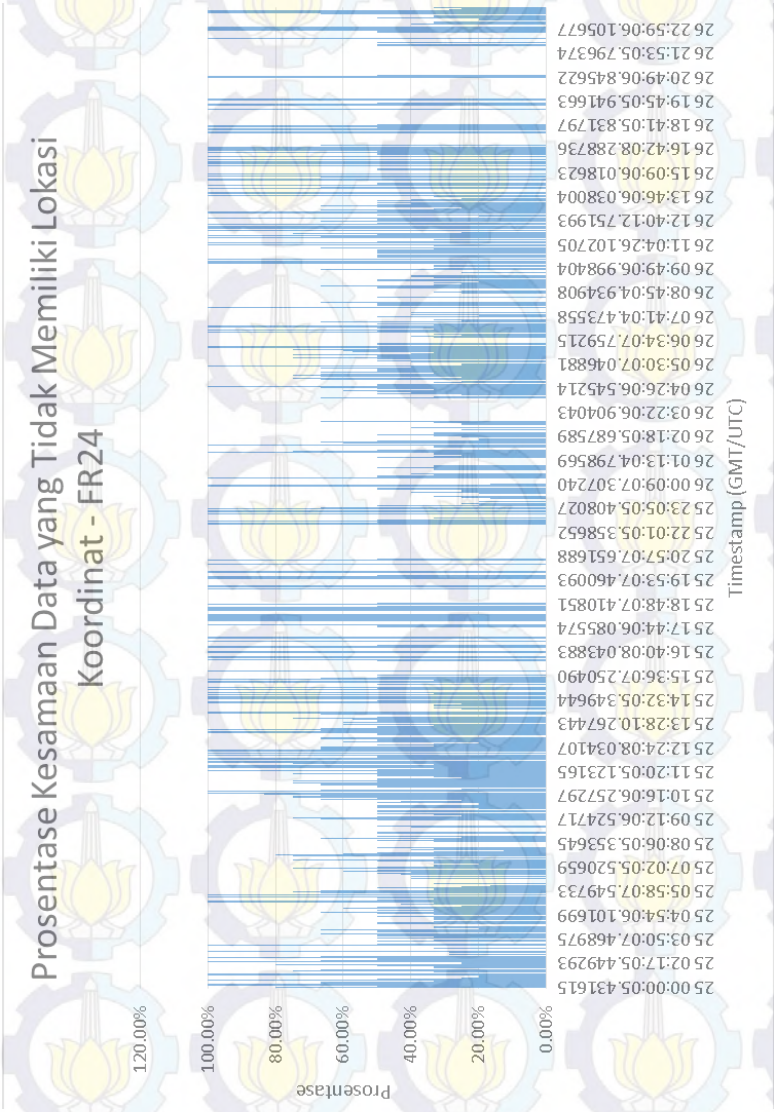
Data yang ada, dibagi dua, yakni data yang memiliki koordinat posisi lintang-bujur dan data yang tidak memiliki koordinat posisi. Data yang memiliki koordinat posisi seharusnya ada pada FR24 dan PF. Sementara data yang tidak memiliki posisi, seharusnya tidak ada pada FR24 dan PF.

1. Uji Coba terhadap data flightradar24.com
   Prosentase data yang memiliki koordinat terhadap data FR24 ditunjukkan pada **Gambar 5-9**. Sementara prosentase data yang tidak memiliki koordinat ditunjukkan pada **Gambar 5-10**. Banyak data: 2.621.
2. Uji Coba terhadap data planefinder.com
   Prosentase data yang memiliki koordinat terhadap data FR24 ditunjukkan pada **Gambar 5-11**. Sementara prosentase data yang tidak memiliki koordinat ditunjukkan pada **Gambar 5-12**. Banyak data: 2.670.

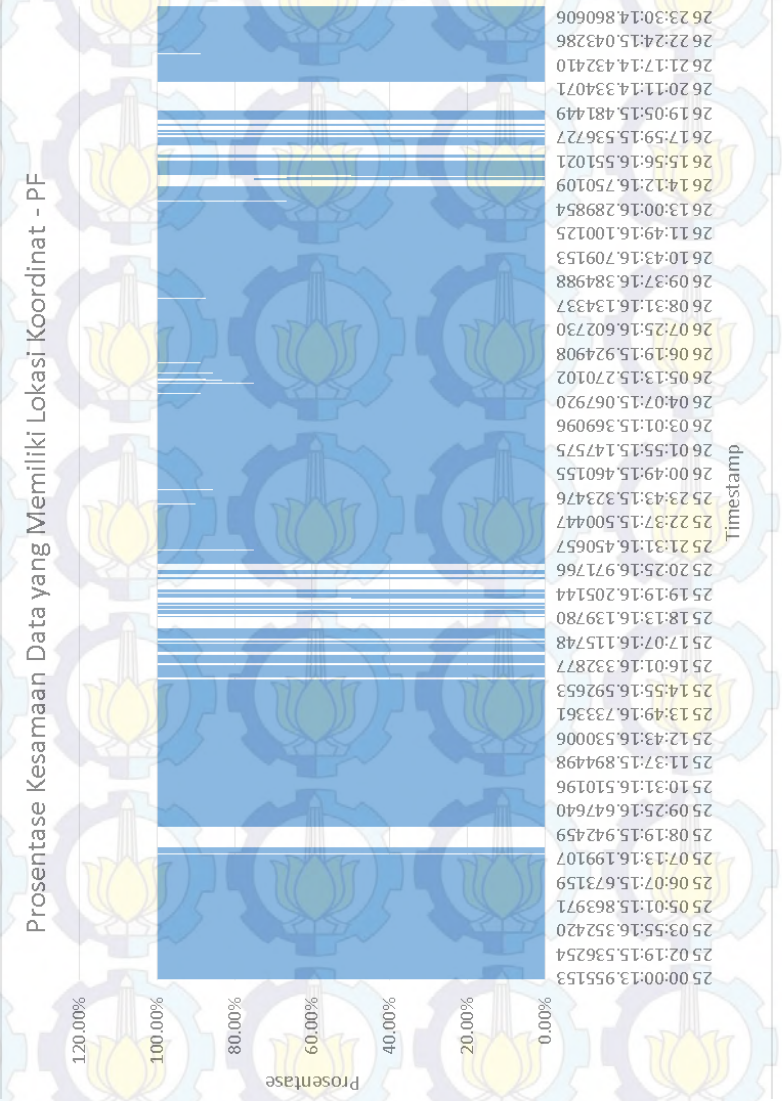
Adapun perhitungan kesamaan secara rata-rata, dapat dilihat pada **Tabel 5-3**.

Secara umum, grafik dan tabel menunjukkan persamaan yang sangat baik untuk data yang memiliki posisi koordinat, baik terhadap FR24 maupun PF. Namun untuk data yang tidak memiliki posisi koordinat, hasilnya masih cukup. Hal ini dikarenakan sistem FR24 maupun PF memiliki aturan tersendiri untuk menyeleksi validitas data yang tidak

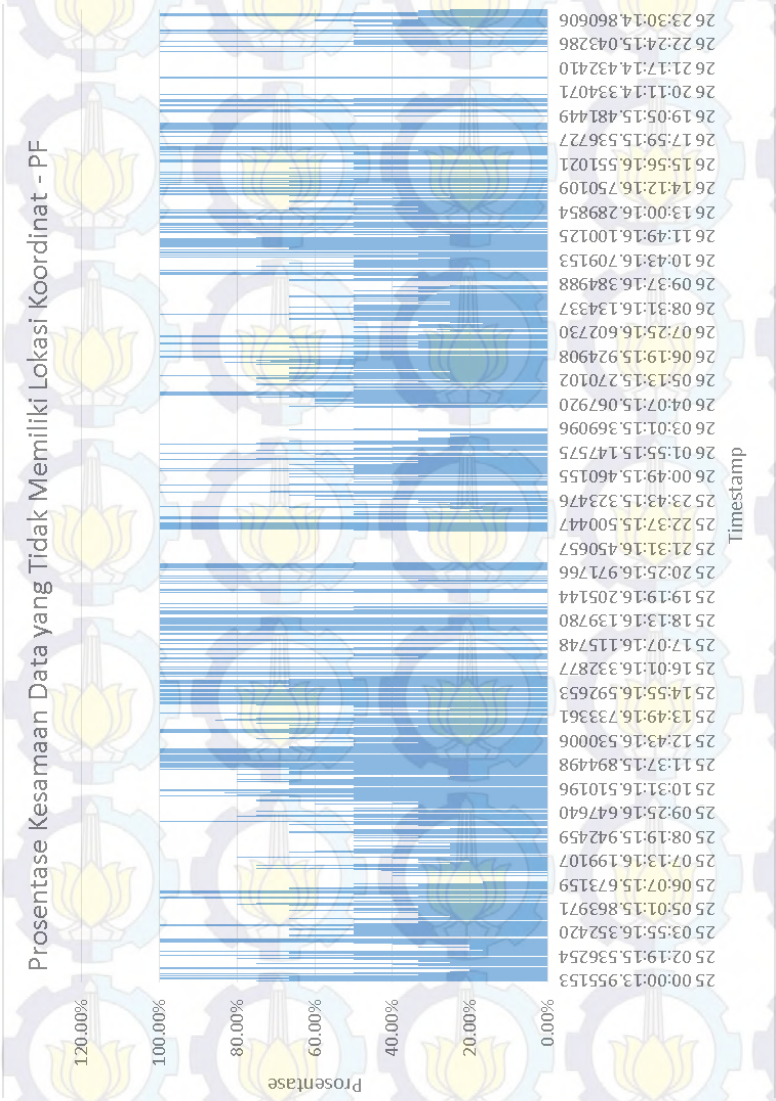
Gambar 5-9 Prosentase Kesamaan Data yang Memiliki Lokasi Koordinat - FR24



Gambar 5-10 Prosentase Kesamaan Data yang Tidak Memiliki Lokasi Koordinat - FR24



Gambar 5-11 Prosentase Kesamaan Data yang Memiliki Lokasi Koordinat - PF



Gambar 5-12 Prosentase Kesamaan Data yang Tidak Memiliki Lokasi Koordinat – PF

diketahui. Selain itu, dimungkinkan adanya receiver lain (diluar sistem) yang mengirim data ke FR24/PF.

Tabel 5-3 Rata-rata Kesamaan Data Flightradar24

Kesamaan (Akurasi)	**Data yang memiliki lokasi koordinat** (tinggi lebih baik)	**Data yang tidak memiliki lokasi koordinat** (rendah lebih baik)
Flightradar24 (average[1])	98.00%	54.56%
Flightradar24 (averagea[2])	81.66%	22.27%
Planefinder (average)	98.41%	60.27%
Planefinder (averagea)	81.93%	30.70%

5.5. Evaluasi validitas JSON dengan Viewer1090

Data JSON yang dibuat oleh sistem telah diuji dengan aplikasi viewer1090, bagian dari aplikasi dump1090 yang dikembangkan oleh Malcolm Robb [20]. Dilakukan modifikasi terhadap viewer1090 untuk mencocokkan nama field JSON yang dihasilkan pada *file* JSON oleh sistem. Aplikasi viewer1090 yang berbasis web ini, digunakan untuk menampilkan data keluaran sistem dump1090 yang mirip dengan keluaran sistem yang dibuat pada tugas akhir ini.

---

[1] Perhitungan rata-rata yang hanya menggunakan nilai numerik saja. Nilai seperti #N/A atau #DIV/0 tidak diperhitungkan

[2] Perhitungan rata-rata yang menggunakan nilai numerik, nilai logika, atau teks (dinilai 0), nilai #N/A atau #DIV/0 dinilai 0.

Gambar 5-13 Tampilan Viewer1090 pada http://viewer.adsb.yoza.my.id

Data yang dikeluarkan oleh sistem dapat diakses pada viewer1090 secara *Online Real-time* pada tautan: http://viewer.adsb.yoza.my.id. Tampilan viewer1090 yang mengakomodasi data keluaran sistem dapat dilihat pada **Gambar 5-13**.

5.6. Evaluasi validitas KML dengan Google Earth

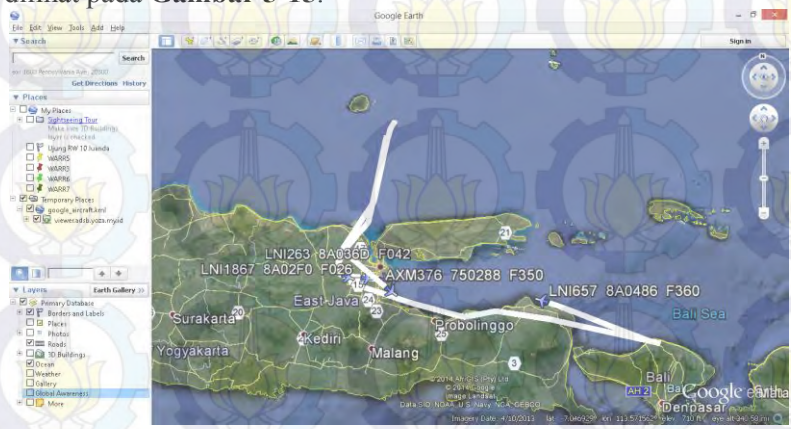
Data KML yang dibuat oleh sistem telah diuji dengan aplikasi google earth. Dibutuhkan sebuah KML lain dengan network-link untuk dapat membaca data KML secara *Online Real-time*, seperti ditunjukkan **Gambar 5-14**.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Document>
<visibility>1</visibility>
<NetworkLink>
<name>ADSB Viewer</name>
<flyToView>0</flyToView>
<Url><href> PATH KE ONLINE KML </href>
<refreshMode>onInterval</refreshMode>
<refreshInterval>15</refreshInterval>
```

```
<viewRefreshMode>never</viewRefreshMode>
</Url>
<refreshVisibility>1</refreshVisibility>
</NetworkLink>
</Document>
</kml>
```

Gambar 5-14 Isi berkas KML dengan Network Link untuk Mengakses Data KML ADS-B

Tampilan google earth yang mengakomodasi data keluaran sistem dapat dilihat pada **Gambar 5-15**.



Gambar 5-15 Tampilan Aplikasi Google Earth yang mengakomodasi Data Keluaran Sistem

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini dibahas kesimpulan dari perancangan, implementasi, uji coba, dan evaluasi sistem. Selain itu, dibahas pula saran untuk mendapatkan hasil yang lebih baik.

6.1. Kesimpulan

Kesimpulan yang diperoleh dari uji coba dan evaluasi adalah sebagai berikut:

1. Data masukan sistem (data ADS-B) adalah data yang merepresentasikan pergerakan pesawat yang berada di sekitar bandara Juanda, Surabaya, sesuai lingkup penerimaan *receiver* ADS-B seperti dibuktikan oleh pengujian pada bagian 5.4.1 Uji Coba terhadap Ground Truth.
2. Sistem mampu melakukan peningkatan kualitas data, dibuktikan dengan kemampuan sistem membedakan validitas identitas dan konten data ADS-B pada pengujian 5.4.2 Uji Coba Intern bagian pertama.
3. Sistem mampu melakukan penyaringan terhadap data yang tidak valid, sehingga mengurangi ukuran data yang dihasilkan oleh sistem, seperti diperlihatkan pada pengujian 5.4.2 Uji Coba Intern bagian kedua.
4. Hasil keluaran sistem telah diuji dengan data dari sistem serupa, dan menghasilkan performa yang sangat baik untuk data dengan koordinat lokasi, serta cukup baik untuk data yang tidak memiliki koordinat lokasi, seperti ditunjukkan pada pengujian 5.4.3 Uji Coba Ekstern.
5. Hasil keluaran sistem yang berupa JSON dan KML dapat dimanfaatkan oleh aplikasi lain, diantaranya viewer1090 dan google earth, seperti diperlihatkan pada pengujian 5.5 Evaluasi validitas JSON dengan Viewer1090 dan 5.6 Evaluasi validitas KML dengan Google Earth.

6.2. Saran

Saran yang dapat diberikan dalam pengujian sistem ini adalah sebagai berikut:

1. Penggunaan RTL2832U/R820T adalah salah satu jalan mendapatkan data ADS-B paling mudah dan murah, akan tetapi kualitas data dan luas jangkauan menjadi permasalahan yang harus dipecahkan. Pada penulisan tugas akhir ini, diperlukan antena c*oaxial-collinear* untuk meningkatkan sensitivitas alat, sehingga data yang diterima menjadi lebih luas cakupannya dan lebih baik kualitasnya.
2. Pada aplikasi *decoder* RTL1090 ditemukan kutu/*bug* sehingga lokasi pada data ground menjadi kacau (sangat jauh dari nilai aslinya). *Decoder* sangat dimungkinkan untuk diganti dengan dump1090 atau aplikasi *decoder* lain yang serupa.
3. Sistem perlu membandingkan *timestamp* yang ada dalam data. Pada tahap pembandingan timestamp ini, terjadi perbedaan waktu pada komputer yang digunakan, meskipun seluruh komputer pada sistem, telah tersinkronisasi dengan pool Network Time Protocol (NTP). Ada kalanya timestamp ketika data dikirim, lebih besar daripada timestamp ketika data diterima, yang pada kondisi umum, tidak mungkin terjadi.
4. Pertimbangan kualitas data dapat dikembangkan dengan metode-metode lain atau ukuran-ukuran lain. Pada tugas akhir ini, digunakan metode pohon keputusan / decision tree karena sejauh ini masalah yang ditemui cukup diatasi dengan metode pohon keputusan. Tidak tertutup kemungkinan terdapat masalah-masalah lain yang tidak/belum ditemukan.

DAFTAR PUSTAKA

[1] FAA - Federal Aviation Administration, "Surveillance and Broadcast Services," [Online]. Available: http://www.faa.gov/nextgen/implementation/programs/adsb/broadcastservices/. [Accessed 10 Maret 2014].

[2] FAA - Federal Aviation Administration, "ADS-B Final Rule," 28 Mei 2010. [Online]. Available: http://edocket.access.gpo.gov/2010/pdf/2010-12645.pdf. [Accessed 1 Maret 2014].

[3] A. B. Pradana, Automatic Dependent Surveillance (ADS) Controller-Pilot Data Link Communication (CPDLC), Curug-Tangerang: Civil Aviation Training Institute - Aviation Safety Training Department, 2012.

[4] jetvision.de, "RTL1090 Installation and Operating Manual," [Online]. Available: http://rtl1090.web99.de/homepage/index.php?way=1&site=READOUT&DERNAME=Manual&dm=rtl1090&USER=rtl1090&goto=1&XURL=rtl1090&WB=1&EXTRAX=X&PIDX=102385. [Accessed 01 Maret 2014].

[5] S. Tsang, B. Kao, K. Y. Yip, W. S. Ho and S. D. Lee, "Decision Trees for Uncertain Data," *IEEE Transactions on Knowledge and Data Engineering,* vol. 23, no. 1, pp. 64-78, 01 2011.

[6] Department of Computer and Information Science and Engineering, University of Florida, "ID3," 1997. [Online]. Available: http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm. [Accessed 24 06 2014].

[7] C. Batini, C. Cappiello, C. Francalanci and A. Maurino, "Methodologies for data quality assessment and improvement," *ACM Computing Surveys (CSUR),* vol. 41, no. 3, pp. 16:1 - 16:52, July 2009.

[8] Y. Wand and R. Y. Wang, "Anchoring Data Quality Dimensions in Ontological Foundations," *Communications of the ACM,* pp. 86-95, Nov 1996.

[9] R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," *Journal of Management Information Systems,* vol. 12, no. 4, pp. 5-33, 1996.

[10] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Madison, Wisconsin, 2002.

[11] ADS-B Technologies, "ADS-B Technologies Website," ADS-B Technologies, LLC, [Online]. Available: http://www.ads-b.com/. [Accessed 24 06 2014].

[12] F. K. Jondral, "Software-Defined Radio—Basics and Evolution to Cognitive Radio," *EURASIP Journal on Wireless Communications and Networking,* vol. 2005, no. 3, pp. 275-283, 2005.

[13] FAA - Federal Aviation Administration, "The Mode S Team," FAA - Federal Aviation Administration, 06 Maret 2012. [Online]. Available: http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/safety_ops_support/nas_engineering/modes_digitizers_sbsm/modes/. [Accessed 10 Maret 2014].

[14] JSON, "Pengenalan JSON," JSON, [Online]. Available: http://json.org/json-id.html. [Accessed 30 03 2014].

[15] Google Developers, "Keyhole Markup Language -- Google Developers," Google, 17 06 2014. [Online]. [Accessed 19 06 2014].

[16] National Center for Geographic Information and Analysis - University of California, Santa Barbara, "Table 3 -

Calculating the Great Circle Distance Between Two Cities," [Online]. Available: http://www.ncgia.ucsb.edu/education/curricula/giscc/units/u014/tables/table03.html. [Accessed 14 04 2014].

[17] Python Software Foundation, "About," Python Software Foundation, 2014. [Online]. Available: http://legacy.python.org/about/. [Accessed 15 04 2014].

[18] Puslit KIM LIPI, "Network Time Protocol," Puslit KIM LIPI, 27 03 2006. [Online]. Available: http://ntp.kim.lipi.go.id/ntp-ind.htm. [Accessed 15 05 2014].

[19] E. G. Rolfe, P. R. Drake and P. L. Hoover, "Methods and Apparatus For Coordinating ADS-B with Mode-S SSR and/or Having Single Link Communication". United States of America Patent US008004452B2, 23 08 2011.

[20] M. Robb, "MalcolmRobb/Dump1090 - Github," 27 6 2014. [Online]. Available: https://github.com/MalcolmRobb/dump1090. [Accessed 29 6 2014].

BIODATA PENULIS

Adrianus Yoza Aprilio, merupakan anak pertama dari dua bersaudara. Penulis dilahirkan di Surabaya pada tanggal 14 April 1992. Penulis telah menempuh pendidikan formal di TK Katolik Santa Clara Surabaya (1996-1998), SD Katolik Santa Clara Surabaya (1998-2004), SMP Katolik Kolese Santo Yusup 2 Malang (2004-2007), SMAK Kolese Santo Yusup Malang (2007-2010).

Pada tahun 2010 penulis diterima di S1 Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Di Jurusan Teknik Informatika, penulis mengambil bidang minat Komputasi Cerdas dan Visualisasi (KCV). Selama kuliah penulis pernah menjadi asisten beberapa mata kuliah, diantaranya Pemrograman Terstruktur dan Pemrograman Perangkat Mobile. Saat ini penulis juga sedang menempuh Program Studi S2 (Pascasarjana) di Jurusan Teknik Informatika ITS melalui jalur Fast Track yang diharapkan akan diselesaikan pada bulan September 2015.

Selain itu, penulis merupakan Lead Nokia / Microsoft Mobility Innovation Lab, Institut Teknologi Sepuluh Nopember Surabaya. Penulis juga merupakan anggota aktif beberapa forum, diantaranya DVLUP Indonesia, Diskusi-SDR, Proyek BlankOn, Indoflyer.net dan Ilmuterbang.com. Penulis dapat dihubungi melalui alamat email yoza10@mhs.if.its.ac.id.

LAMPIRAN

Lampiran 1. Daftar Alokasi ICAO Hex

Tabel 8-1 Daftar Alokasi ICAO Hex

No	Awal	Akhir	Negara
1	000000	003FFF	(unallocated)
2	004000	0043FF	Zimbabwe
3	006000	006FFF	Mozambique
4	008000	00FFFF	South Africa
5	010000	017FFF	Egypt
6	018000	01FFFF	Libya
7	020000	027FFF	Morocco
8	028000	02FFFF	Tunisia
9	030000	0303FF	Botswana
10	032000	032FFF	Burundi
11	034000	034FFF	Cameroon
12	035000	0353FF	Comoros
13	036000	036FFF	Congo
14	038000	038FFF	Côte d Ivoire
15	03E000	03EFFF	Gabon
16	040000	040FFF	Ethiopia
17	042000	042FFF	Equatorial Guinea
18	044000	044FFF	Ghana
19	046000	046FFF	Guinea
20	048000	0483FF	Guinea-Bissau
21	04A000	04A3FF	Lesotho
22	04C000	04CFFF	Kenya
23	050000	050FFF	Liberia
24	054000	054FFF	Madagascar
25	058000	058FFF	Malawi
26	05A000	05A3FF	Maldives
27	05C000	05CFFF	Mali
28	05E000	05E3FF	Mauritania

No	Awal	Akhir	Negara
29	060000	0603FF	Mauritius
30	062000	062FFF	Niger
31	064000	064FFF	Nigeria
32	068000	068FFF	Uganda
33	06A000	06A3FF	Qatar
34	06C000	06CFFF	Central African Republic
35	06E000	06EFFF	Rwanda
36	070000	070FFF	Senegal
37	074000	0743FF	Seychelles
38	076000	0763FF	Sierra Leone
39	078000	078FFF	Somalia
40	07A000	07A3FF	Swaziland
41	07C000	07CFFF	Sudan
42	080000	080FFF	Tanzania
43	084000	084FFF	Chad
44	088000	088FFF	Togo
45	08A000	08AFFF	Zambia
46	08C000	08CFFF	D R Congo
47	090000	090FFF	Angola
48	094000	0943FF	Benin
49	096000	0963FF	Cape Verde
50	098000	0983FF	Djibouti
51	098000	0983FF	Djibouti
52	09A000	09AFFF	Gambia
53	09C000	09CFFF	Burkina Faso
54	09E000	09E3FF	Sao Tome
55	0A0000	0A7FFF	Algeria
56	0A8000	0A8FFF	Bahamas
57	0AA000	0AA3FF	Barbados
58	0AB000	0AB3FF	Belize
59	0AC000	0ACFFF	Colombia
60	0AE000	0AEFFF	Costa Rica
61	0B0000	0B0FFF	Cuba
62	0B2000	0B2FFF	El Salvador
63	0B4000	0B4FFF	Guatemala
64	0B6000	0B6FFF	Guyana

No	Awal	Akhir	Negara
65	0B8000	0B8FFF	Haiti
66	0BA000	0BAFFF	Honduras
67	0BC000	0BC3FF	St.Vincent + Grenadines
68	0BE000	0BEFFF	Jamaica
69	0C0000	0C0FFF	Nicaragua
70	0C2000	0C2FFF	Panama
71	0C4000	0C4FFF	Dominican Republic
72	0C6000	0C6FFF	Trinidad and Tobago
73	0C8000	0C8FFF	Suriname
74	0CA000	0CA3FF	Antigua & Barbuda
75	0CC000	0CC3FF	Grenada
76	0D0000	0D7FFF	Mexico
77	0D8000	0DFFFF	Venezuela
78	100000	1FFFFF	Russia
79	200000	27FFFF	(reserved, AFI)
80	201000	2013FF	Namibia
81	202000	2023FF	Eritrea
82	280000	2FFFFF	(reserved, SAM)
83	300000	33FFFF	Italy
84	340000	37FFFF	Spain
85	380000	3BFFFF	France
86	3C0000	3FFFFF	Germany
87	400000	43FFFF	United Kingdom
88	440000	447FFF	Austria
89	448000	44FFFF	Belgium
90	450000	457FFF	Bulgaria
91	458000	45FFFF	Denmark
92	460000	467FFF	Finland
93	468000	46FFFF	Greece
94	470000	477FFF	Hungary
95	478000	47FFFF	Norway
96	480000	487FFF	Netherlands
97	488000	48FFFF	Poland
98	490000	497FFF	Portugal
99	498000	49FFFF	Czech Republic
100	4A0000	4A7FFF	Romania

No	Awal	Akhir	Negara
101	4A8000	4AFFFF	Sweden
102	4B0000	4B7FFF	Switzerland
103	4B8000	4BFFFF	Turkey
104	4C0000	4C7FFF	Yugoslavia
105	4C8000	4C83FF	Cyprus
106	4CA000	4CAFFF	Ireland
107	4CC000	4CCFFF	Iceland
108	4D0000	4D03FF	Luxembourg
109	4D2000	4D23FF	Malta
110	4D4000	4D43FF	Monaco
111	500000	5003FF	San Marino
112	500000	5FFFFF	(reserved, EUR/NAT)
113	501000	5013FF	Albania
114	501C00	501FFF	Croatia
115	502C00	502FFF	Latvia
116	503C00	503FFF	Lithuania
117	504C00	504FFF	Moldova
118	505C00	505FFF	Slovakia
119	506C00	506FFF	Slovenia
120	507C00	507FFF	Uzbekistan
121	508000	50FFFF	Ukraine
122	510000	5103FF	Belarus
123	511000	5113FF	Estonia
124	512000	5123FF	Macedonia
125	513000	5133FF	Bosnia & Herzegovina
126	514000	5143FF	Georgia
127	515000	5153FF	Tajikistan
128	600000	6003FF	Armenia
129	600000	67FFFF	(reserved, MID)
130	600800	600BFF	Azerbaijan
131	601000	6013FF	Kyrgyzstan
132	601800	601BFF	Turkmenistan
133	680000	6FFFFF	(reserved, ASIA)
134	680000	6803FF	Bhutan
135	681000	6813FF	Micronesia
136	682000	6823FF	Mongolia

No	Awal	Akhir	Negara
137	683000	6833FF	Kazakhstan
138	684000	6843FF	Palau
139	700000	700FFF	Afghanistan
140	702000	702FFF	Bangladesh
141	704000	704FFF	Myanmar
142	706000	706FFF	Kuwait
143	708000	708FFF	Laos
144	70A000	70AFFF	Nepal
145	70C000	70C3FF	Oman
146	70E000	70EFFF	Cambodia
147	710000	717FFF	Saudi Arabia
148	718000	71FFFF	Korea (South)
149	720000	727FFF	Korea (North)
150	728000	72FFFF	Iraq
151	730000	737FFF	Iran
152	738000	73FFFF	Israel
153	740000	747FFF	Jordan
154	748000	74FFFF	Lebanon
155	750000	757FFF	Malaysia
156	758000	75FFFF	Philippines
157	760000	767FFF	Pakistan
158	768000	76FFFF	Singapore
159	770000	777FFF	Sri Lanka
160	778000	77FFFF	Syria
161	780000	7BFFFF	China
162	7C0000	7FFFFF	Australia
163	800000	83FFFF	India
164	840000	87FFFF	Japan
165	880000	887FFF	Thailand
166	888000	88FFFF	Viet Nam
167	890000	890FFF	Yemen
168	894000	894FFF	Bahrain
169	895000	8953FF	Brunei
170	896000	896FFF	United Arab Emirates
171	897000	8973FF	Solomon Islands
172	898000	898FFF	Papua New Guinea

No	Awal	Akhir	Negara
173	899000	8993FF	Taiwan (unofficial)
174	8A0000	8A7FFF	Indonesia
175	900000	9FFFFF	(reserved, NAM/PAC)
176	900000	9003FF	Marshall Islands
177	901000	9013FF	Cook Islands
178	902000	9023FF	Samoa
179	A00000	AFFFFF	United States
180	B00000	BFFFFF	(reserved)
181	C00000	C3FFFF	Canada
182	C80000	C87FFF	New Zealand
183	C88000	C88FFF	Fiji
184	C8A000	C8A3FF	Nauru
185	C8C000	C8C3FF	Saint Lucia
186	C8D000	C8D3FF	Tonga
187	C8E000	C8E3FF	Kiribati
188	C90000	C903FF	Vanuatu
189	D00000	DFFFFF	(reserved)
190	E00000	E3FFFF	Argentina
191	E40000	E7FFFF	Brazil
192	E80000	E80FFF	Chile
193	E84000	E84FFF	Ecuador
194	E88000	E88FFF	Paraguay
195	E8C000	E8CFFF	Peru
196	E90000	E90FFF	Uruguay
197	E94000	E94FFF	Bolivia
198	EC0000	EFFFFF	(reserved, CAR)
199	F00000	F07FFF	ICAO (1)
200	F00000	FFFFFF	(reserved)
201	F09000	F093FF	ICAO (2)

Lampiran 2. Daftar 84 Variabel Keluaran HTTP RTL1090

Tabel 8-2 Daftar 84 Variabel Keluaran HTTP RTL1090

Item	ICAO code	Field name	Data type
0	AA	Address announced	Hex integer (String [6])
1	CAT	Aircraft category	Char [A,B,C,D] + Integer(0..7)
2	CS	Callsign	String [0..8]
3	FS	Flight status	Integer (0..7)
4	CA	Capability	Integer (0..7)
5	DR	Downlink request	Integer (0..31)
6	UM	Utility message	Integer (0..15)
7	ID	Identity (Mode A code)	Octal (0000..7777)
8	LAT	Latitude	Float
9	LON	Longitude	Float
10	NIC	Navigation Integrity Category	Integer (0..11)
11	NUCR/ NACV	Navigation Uncertainty Category - Velocity	Integer (0..4)
12	FL	Flight level	String[4]
13	AC	Altitude code	Integer (-1000...99999)
14	M	Metric altitude	Integer (0,1)
15	Q	Altitude resolution	Integer (0,1)

Item	ICAO code	Field name	Data type
16	GNSS	GPS altitude difference	Integer (-99999..+99999)
17	HAE	GNSS Height (HAE)	Integer (-99999..+99999)
18	VSRC	Vertical rate source	Integer (0,1)
19	VR	Vertical rate	Integer (-99999..+99999)
20	VRF	Vertical rate formatted	String[0..3]
21	HAB	Heading available bit	Integer (0,1)
22	TT	True track	Integer (0..359)
23	HDG	Heading	Integer (0..359)
24	AST	ADS-B airspeed type	Integer (0,1)
25	IAS	Indicated Air Speed (IAS)	Integer (0..999)
26	TAS	True Air Speed (TAS)	Integer (0..999)
27	GS	Ground Speed (GS)	Integer (0..999)
28	MACH	Mach number	Integer (0..999)
29	MCP	MCP altitude source	Integer (0..3)
30	RA	Bank angle	Integer (-90..90)
31	TR	Turn rate	Integer (-90..90)
32	MCP	Selected altitude	Integer (0..655)
33	FMS	Selected altitude	Integer (0..655)
34	QNH	Altimeter setting	Integer (800..1209)

Item	ICAO code	Field name	Data type
35	FCU	MCP/FCU mode	Integer (0..7)
36	PA	Permanent alert condition (Emergency)	Integer (0,1)
37	PS	Priority status	Integer (0,1)
38	ACAS	ACAS alert	Integer (0,1)
39	ALRT	Alert	Integer (0,1)
40	SPI	SPI	Integer (0,1)
41	GR	Ground report	Integer (0,1)
42	IC	Intent change	Integer (0,1)
43	SSC	Supersonic	Integer (0,1)
44	IFR	IFR capability	Integer (0,1)
45	TS	Time sync	Integer (0,1)
46	RI	Reply information (air-air)	Integer (0..15)
47	CC	Cross-link capability	Integer (0,1)
48	SL	Sensitivity level report	Integer (0..7)
49	ADSB	Last ADS-B packet format	Integer (17,18,19)
50	BDS40	BDS4,0	Integer (0,1)
51	BDS50	BDS5,0	Integer (0,1)
52	BDS60	BDS6,0	Integer (0,1)
53	BDS65	BDS6,5	Integer (0,1)
54	FMT30	FMT30	Integer (0,1)
55	BDS61	BDS6,1	Integer (0,1)
56	WDIR	Wind direction	Integer (0..359)
57	WSPD	Wind speed	Integer (0..999)
58	TEMP	OAT	Integer (-99..+99)
59	AGE_0	Data age DF0	Integer (0...9999)

Item	ICAO code	Field name	Data type
60	AGE_4	Data age DF4	Integer (0...9999)
61	AGE_5	Data age DF5	Integer (0...9999)
62	AGE_11	Data age DF11	Integer (0...9999)
63	AGE_16	Data age DF16	Integer (0...9999)
64	AGE_17_0e	Data age DF17-19	Integer (0...9999)
65	AGE_17_0o	Data age DF17-19	Integer (0...9999)
66	AGE_17_1	Data age DF17-19	Integer (0...9999)
67	AGE_17_5e	Data age DF17-19	Integer (0...9999)
68	AGE_17_5o	Data age DF17-19	Integer (0...9999)
69	AGE_17_19	Data age DF17-19	Integer (0...9999)
70	AGE_17_28	Data age DF17-19	Integer (0...9999)
71	AGE_17_30	Data age DF17-19	Integer (0...9999)
72	AGE_17_31	Data age DF17-19	Integer (0...9999)
73	AGE_20_20	Data age DF20 BDS2,0	Integer (0...9999)
74	AGE_20_40	Data age DF20 BDS4,0	Integer (0...9999)
75	AGE_20_50	Data age DF20 BDS5,0	Integer (0...9999)
76	AGE_20_60	Data age DF20 BDS6,0	Integer (0...9999)
77	AGE_21_20	Data age DF21 BDS2,0	Integer (0...9999)
78	AGE_21_40	Data age DF21 BDS4,0	Integer (0...9999)

Item	ICAO code	Field name	Data type
79	AGE_21_50	Data age DF21 BDS5,0	Integer (0...9999)
80	AGE_21_60	Data age DF21 BDS6,0	Integer (0...9999)
81	TIMESTAMP		Integer
82	TIMEOUT		Integer
83	STATION ID		String