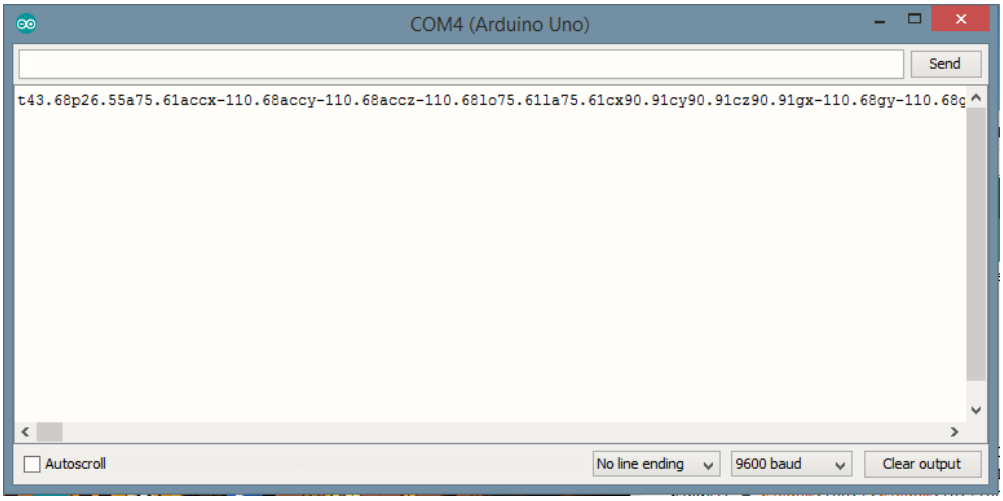
BAB IV
HASIL DAN PEMBAHASAN

4.1 HASIL

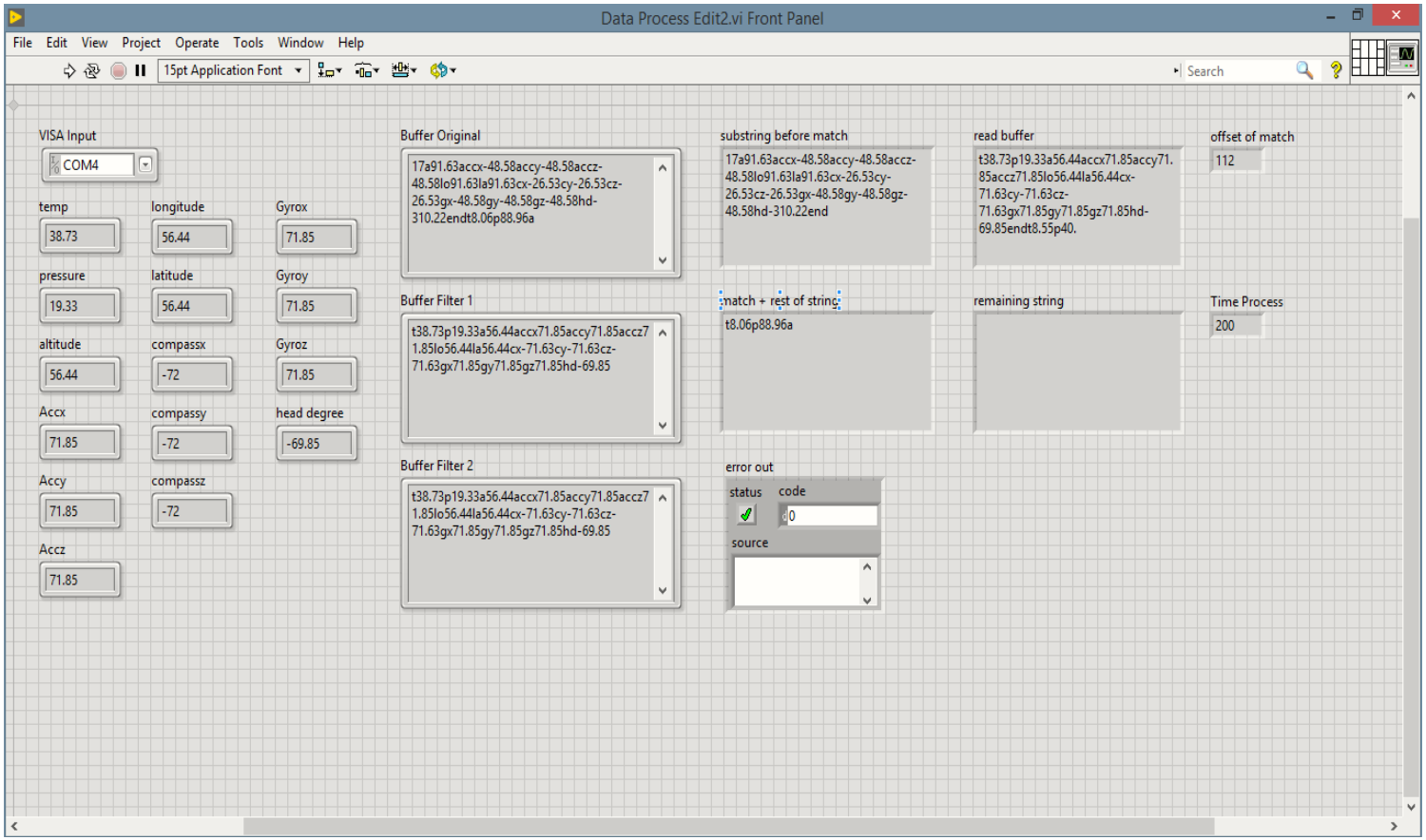
Pada sub bagian ini akan disajikan hasil dari metode penelitian yang telah dilakukan. Hasil yang disampaikan berupa gambar saja, agar mempermudah dalam penjelasan di sub bagian pembahasan nantinya. Selain itu, gambar akan lebih mudah untuk disimak.

Setelah dilakukan simulasi *dummy data* menggunakan Arduino UNO, diperoleh hasil berupa data dinamis yang terus berjalan hingga tak terhingga (selama Arduino UNO terhubung serial). Data tersebut diambil dari komunikasi COM4 antara Arduino UNO dengan laptop (Arduino IDE). Berikut ini (Gambar 4.1) merupakan hasil dari dibukanya *Serial Monitor*.



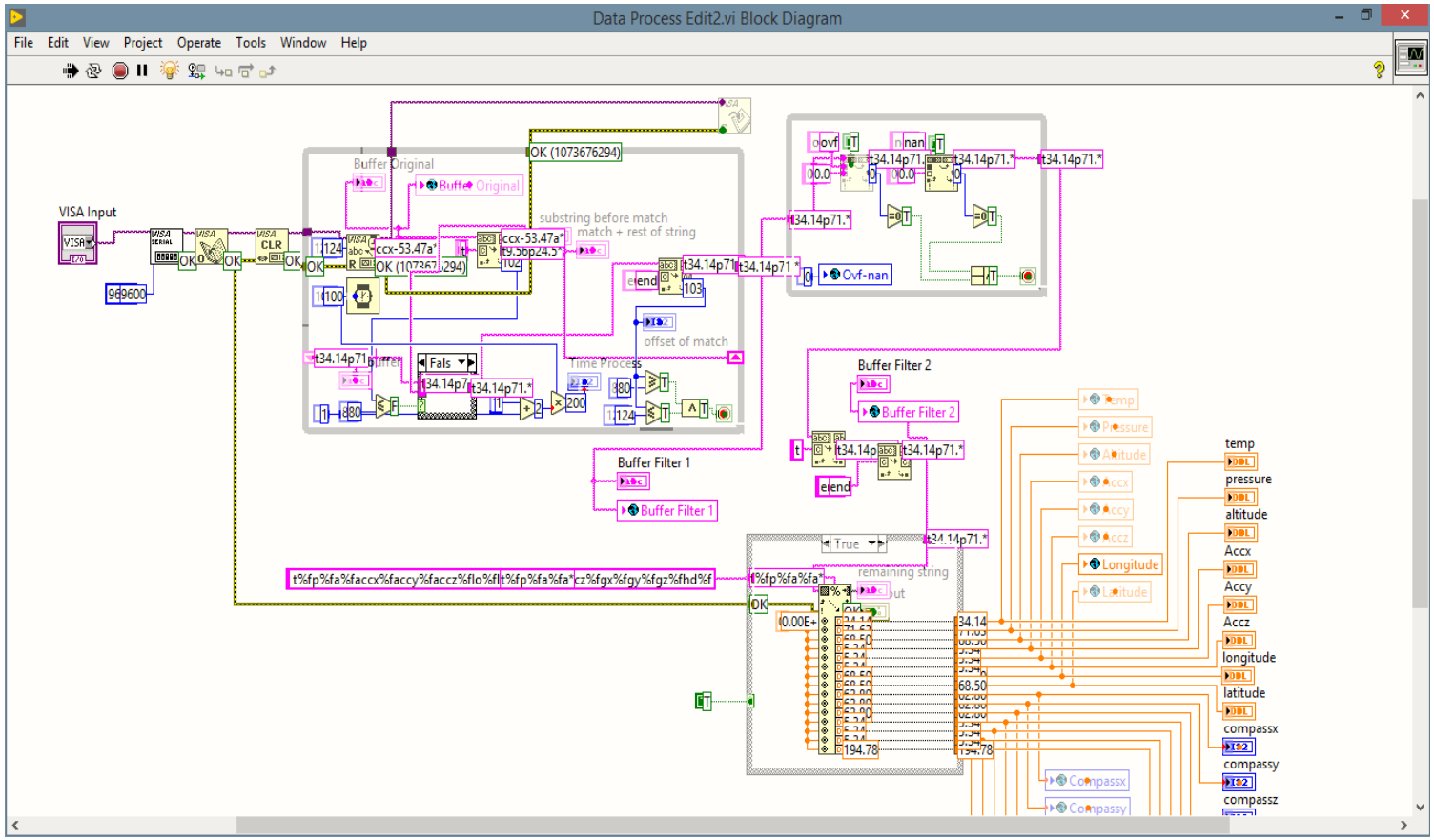
Gambar 4.1 Tampilan hasil dummy data *pada komunikasi serial*
(Diambil pada 4 Maret 2019 pukul 22.45)

Kemudian, setelah simulasi pada Arduino berhasil, simulasi dilanjutkan pada proses pengujian antarmuka yang telah dibuat menggunakan LabVIEW 2017. Setelah dilakukan beberapa kali pengembangan antarmuka, akhirnya diperoleh tampilan antarmuka yang sesuai dengan tujuan tugas akhir. Hasil tersebut dapat dilihat pada Gambar 4.2 berikut ini.



Gambar 4.2 Hasil pengolahan data melalui LabVIEW 2017
(Diambil pada 4 Maret 2019 pukul 23.03)

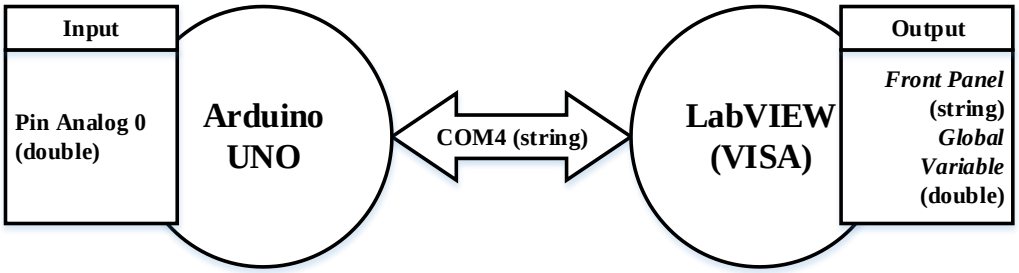
Dari *Front Panel* LabVIEW di atas, tersembunyi proses pengolahan data yang berada di bagian blok diagram. Proses ini dapat dilihat menggunakan ikon  ketika LabVIEW dijalankan. Berikut ini hasil dari proses tersebut.



Gambar 4.3 Proses pengolahan data pada blok diagram LabVIEW 2017
(Diambil pada 4 Maret 2019 pukul 23.05)

4.2 PEMBAHASAN

Pembahasan meliputi proses aliran data dari Arduino ke VISA dan aliran data dari VISA ke tampilan LabVIEW. Proses aliran data dari Arduino ke VISA diawali proses pemrograman dan menghasilkan *dummy data* yang kemudian dikirimkan ke serial sehingga dapat diterima oleh VISA. Hal ini akan dibahas lebih lanjut pada bagian 4.2.1. Selanjutnya, proses aliran data dari VISA ke tampilan LabVIEW melalui proses pengolahan data pada *Diagram Block* sehingga hasilnya dapat ditampilkan pada *Front Panel*. Hal ini akan dijelaskan pada bagian 4.2.2 dan 4.2.3. Berikut ini (Gambar 4.4) adalah blok diagram dari aliran data dari Arduino UNO ke LabVIEW, beserta tipe datanya (di dalam kurung).



Gambar 4.4 Blok diagram aliran data dari Arduino UNO ke LabVIEW
(Dibuat pada 14 Maret 2019 pukul 08.53)

4.2.1 PEMROGRAMAN DAN *DUMMY DATA*

Pertama, pemrograman mikrokontroler (Arduino) dan pengiriman *dummy data* ke komunikasi serial sebelum dibaca oleh LabVIEW melalui VISA. Hal ini dimulai dari perintah "*double randTemp, randPress, randAlt, randG, randC, randHD;*" yang merupakan sebuah inisialisasi/deklarasi variabel, yaitu menjelaskan jenis data dari variabel yang nantinya digunakan pada bagian program utama (*void loop()*). Format dari variabel-variabel di atas adalah ***double***, yaitu jenis data berjenis bilangan bulat desimal. Ada enam variabel yang dideklarasikan dengan cara dipisahkan dengan koma (,) untuk setiap variabel. Perintah deklarasi tersebut ditutup dengan titik koma (;) untuk menunjukkan bahwa program terbatas pada hal tersebut.

Kemudian, pengaturan Arduino dilakukan melalui fungsi "*void setup()*", dimana fungsi ini telah tersedia secara *default* pada Arduino IDE pada saat akan

diprogram. Kode program "*Serial.begin(9600);*" digunakan untuk mengatur besar *baud rate* pada komunikasi serial Arduino dengan laptop. *Baud rate* yang digunakan cukuplah 9600 karena besaran tersebut sudah terlalu besar untuk digunakan pada aplikasi yang kecil seperti ini. Hal ini karena pada tugas akhir ini, data yang dikirimkan per detik tidak sampai 9600 bit. Selanjutnya akan dibahas pada proses pengolahan data untuk lebih detailnya mengenai pengaruh *baud rate* terhadap kecepatan data. Terakhir, digunakan kode program "*randomSeed(analogRead(0));*" untuk menginisialisasi pin 0 Analog pada Arduino UNO untuk melakukan pembuatan angka acak.

Selanjutnya pembahasan mengenai kode program di dalam fungsi utama "*void loop()*". Kode program baris pertama pada fungsi utama atau baris ke sembilan dari seluruh kode program di Arduino IDE yaitu "*randTemp = random(50)+random(10)/100.0+random(10)/100.0;*" dan sejenisnya digunakan untuk mendefinisikan variabel tersebut dengan nilai acak yang dibuat menggunakan fungsi "*random()*". Di dalam fungsi "*random()*", dimasukkan angka untuk merepresentasikan angka acak yang berada di dalam interval 0 hingga angka tersebut (misal dimasukkan angka 100, maka angka acak yang dihasilkan adalah antara 0 hingga 100). Hal tersebut hanya memproduksi angka acak yang bersifat bilangan bulat. Untuk mendapatkan angka desimal, digunakan logika "nilai acak yang dihasilkan oleh fungsi *random()* dibagi dengan 10^x, dimana x bilangan bulat positif". Penggunaan kode "*random(10)/10.0*" menghasilkan bilangan desimal satu angka di belakang koma (0,1 – 0,9), sedangkan penggunaan kode "*random(10)/100.0*" menghasilkan bilangan desimal dua angka di belakang koma (0,01 – 0,09). Akhirnya, nilai-nilai acak itu dijumlahkan dalam satu rumus tersebut sehingga menghasilkan angka acak dengan interval 0.00 – 50.99. Untuk yang menggunakan fungsi "*random(-120, 120)*", berarti interval angka acak yang dihasilkan adalah antara -120 hingga 120.

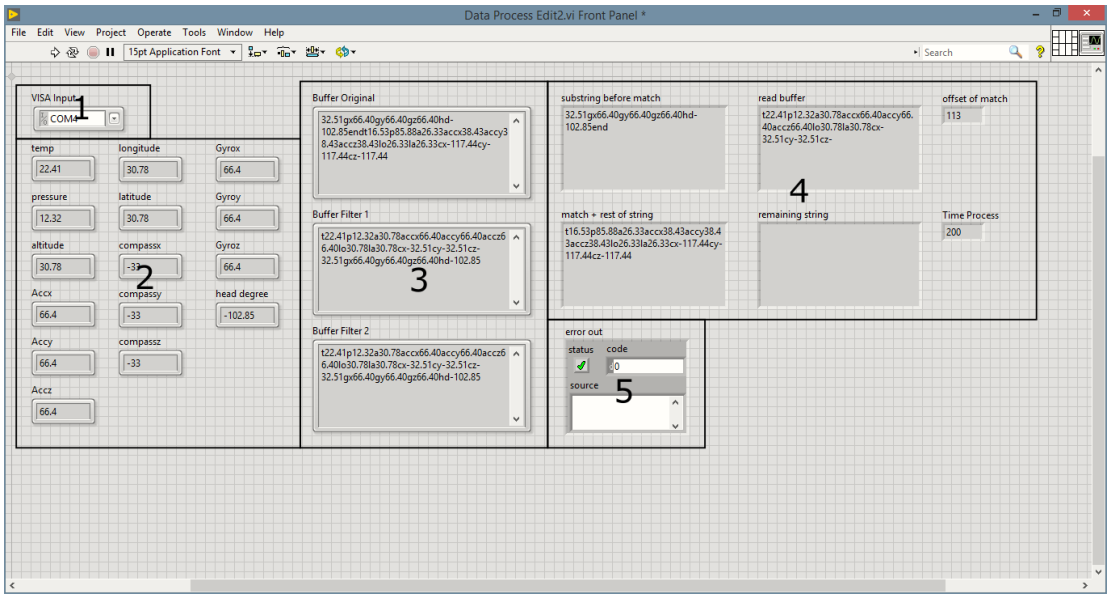
Proses yang tidak dilewatkan adalah penulisan data yang sudah dihasilkan dari pin 0 Analog tadi ke komunikasi serial. Penggunaan fungsi "*Serial.print(t)*" dan seterusnya digunakan untuk melakukan tugas tersebut. Urutan pengiriman ini disesuaikan dengan format yang data yang telah disampaikan pada bab metode

penelitian tentang perancangan piranti lunak atau program. Selain itu, untuk mengatur pola pengiriman data secara baik, maka dibuat jeda setiap 100 milisekon dengan kode program "*delay(100)*".

Seperti yang terlihat pada Gambar 4.1 (bagian hasil), hasil keluaran yang dihasilkan oleh pin 0 Analog dikirimkan ke komunikasi serial. Hasilnya adalah angka yang sangat panjang. Jenis dari data tersebut adalah ***string***. Ini disebabkan dari penggunaan kode program "*Serial.print()*", bukan "*Serial.println()*". Maksud dari "*println*" adalah *print line* atau cetak per baris.

Data memang dibuat tidak per baris, berdasarkan penelitian sebelumnya. Sebenarnya, data dapat dikirim secara per baris untuk memudahkan penerimaan data pada LabVIEW. Namun, ada kemungkinan data yang diterima tidak lengkap, akibat faktor internal dari perangkat keras, umumnya akibat dari Arduino UNO. Ini pun Arduino UNO yang bukan asli, artinya produksi selain dari *Arduino CC* (Italia). Hal ini sama sekali tidak mempengaruhi kinerja VISA, karena VISA hanya berfungsi untuk mengambil nilai dari komunikasi serial (COM4), baik datanya lengkap maupun tidak.

4.2.2 TAMPILAN LABVIEW



Gambar 4.5 Bagian-bagian dari hasil Front Panel
(Diambil pada 14 Maret 2019 pukul 16.59)

Kedua, tampilan panel antarmuka melalui LabVIEW. Seperti yang telah dijelaskan mengenai perencanaan dan pengembangan tampilan LabVIEW bahwa tampilan tersebut sesuai dengan kebutuhan KOMURINDO 2019. Hasilnya dapat dilihat dari Gambar 4.2 (bagian hasil). Bagian-bagian pada tampilan panel antarmuka memiliki penjelasan masing-masing. Hal ini dapat dilihat pada Gambar 4.5 di atas.

Pertama yang dibahas adalah bagian 1, yaitu port yang digunakan dalam komunikasi ini adalah COM4. Hal ini karena COM4 merupakan satu-satunya port yang tersedia antara laptop dengan Arduino. Port dibuat menggunakan *VISA Resource Name (Silver)* yang tersedia pada *Controls Pallette* di bagian *I/O*. Fungsi ini memungkinkan memilih jenis port menggunakan *combo box*.

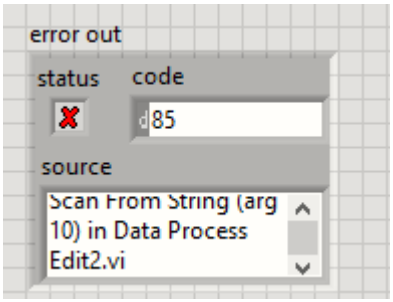
Kedua, bagian 2, yaitu tampilan hasil proses pengolahan data melalui blok diagram. Data yang ditampilkan berbentuk ***string*** karena penampil yang digunakan merupakan penampil data *string*. Penampil yang digunakan adalah *String Indicator (Silver)* yang tersedia pada *Controls Pallette* di bagian *String & Paths*.

Ketiga, bagian 3, yaitu tampilan data *buffer* yang diolah oleh blok diagram. Ada tiga indikator yang ditampilkan, yaitu Buffer Original, Buffer Filter 1, dan Buffer Filter 2. Buffer Original menampilkan data *buffer* dari serial. Kemudian, Buffer Filter 1 menampilkan data *buffer* hasil pengolahan *loop* pertama. Terakhir, Buffer Filter 2 menampilkan data *buffer* hasil pengolahan *loop* kedua (detailnya dijelaskan pada blok diagram). Bagian ini juga menggunakan *String Indicator (Silver)*, hanya saja pada bagian *visible items* – menu yang muncul ketika objek diklik kanan – dicentang pilihan *vertical scrollbar* agar terlihat fungsi *scroll* secara vertikal.

Keempat, bagian 4, yaitu tampilan data *buffer* di dalam *loop* pertama dan waktu proses dari *loop* tersebut. Tampilan data *buffer* dibagi menjadi beberapa bagian, yaitu *substring before match*, *match + rest of string*, *read buffer*, *remaining string*, dan *offset of match*. Intinya, semua objek tersebut menggunakan *String Indicator (Silver)*. Proses kerja objek-objek tersebut akan dijelaskan pada bagian proses pengolahan data.

Bagian terakhir adalah indikator error. Indikator ini menunjukkan apakah data yang diolah pada *Diagram Block* dalam kondisi utuh atau ada yang kurang dari format

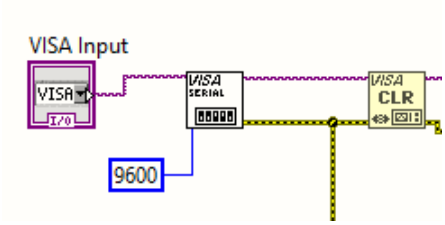
yang sudah ditentukan. Kehandalan didapatkan dari melakukan *stacking error* melalui menampilkan error pada *Front Panel* menggunakan objek *Error Out* yang tersedia pada *Pallette Controls* bagian *Array, Matrix, & Cluster*. Jika ada data yang kurang (*string* tidak lengkap), maka akan muncul indikator seperti pada Gambar 4.6. Hal ini tidak akan menghentikan proses dari LabVIEW.



Gambar 4.6 Kondisi error pada data yang diolah oleh LabVIEW
(Diambil pada 14 Maret 2019 pukul 12.25)

4.2.3 PROSES PENGOLAHAN DATA

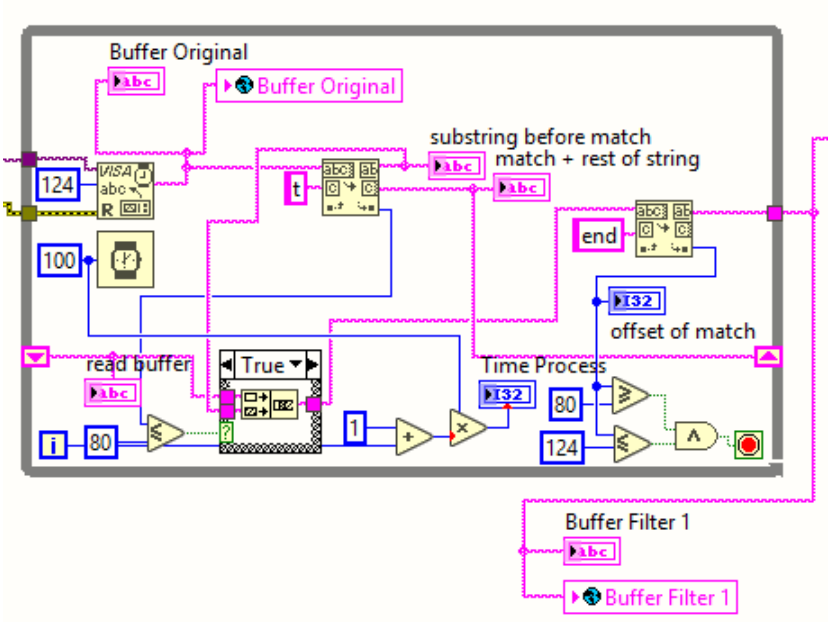
Ketiga, proses pengolahan data, yaitu mencari data (*tracing*) maupun akuisisi data melalui blok diagram di LabVIEW. Hal ini terjadi melalui enam proses. Setiap proses sangat berperan dalam pengolahan data.



Gambar 4.7 Blok diagram VISA untuk menerima data dari COM4
(Diambil pada 14 Maret 2019 pukul 21.48)

Proses yang pertama kali dilalui oleh data dari serial (COM4) adalah masuk ke LabVIEW. *VISA Serial* terhubung ke alamat serial yang sesuai dengan yang tersedia (COM4) melalui pilihan pada *Front Panel*. Besar *baud rate* ditentukan sebesar 9600 untuk membatasi besar data yang masuk ke LabVIEW. Dalam ukuran tersebut, data yang masuk harus berukuran maksimal 96.000 bit per detik dan/atau 12.000 byte. Itu merupakan ruang transfer data yang cukup besar untuk data yang dikirim melalui Arduino UNO yang hanya mengirimkan sebuah data dari satu pin saja.

Sebelum masuk ke *Loop()*, *buffer data* yang tersisa di VISA dibersihkan dulu. Hal ini didukung *VISA CLR* yang berfungsi membersihkan data *cache* yang tersisa ketika melakukan pembacaan data sebelumnya. Tanpa adanya *VISA CLR*, akan timbul error ketika pembacaan string di bagian blok diagram *tracing* data karena ada noise berupa data *cache* tersebut.



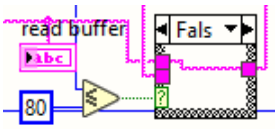
Gambar 4.8 Blok diagram pengolah buffer data yang pertama
(Diambil pada 14 Maret 2019 pukul 22.16)

Selanjutnya, data yang telah diperoleh VISA dimasukkan ke dalam *while loop()* di atas. *Loop()* tersebut hanya berulang ketika logika yang diterima bernilai ***FALSE***. Oleh karena itu, terdapat gerbang logika *AND* yang berperan menghasilkan nilai ***TRUE*** ketika besar data sudah sesuai dengan ukuran yang ditentukan yaitu antara 80 sampai 124 byte.

Proses diawali dengan mengambil data sebesar 124 byte dari VISA menggunakan *VISA Read* yang diatur *byte count*-nya. Data keluar melalui *read buffer*, dan selanjutnya ditampilkan pada *Front Panel* melalui *Buffer Original*. Selain ditampilkan, juga data dikirim ke *Global Variable*.

Search/Split String pertama digunakan untuk mencari nilai sebelum "t" dan setelah "t". Data sebelum "t" diambil melalui *substring before match*, sedangkan data

setelah “t” diambil melalui *match + rest of string*. Keduanya kemudian ditampilkan pada *Front Panel* melalui *Indicator String (Silver)*. Jika ada data sebelum “t”, maka data tersebut akan dikirim ke *Concatenate String* (berada di dalam *Case Structure*) untuk digabungkan dengan data yang tersimpan pada *shift register*. *Shift register* berperan menyimpan sementara data setelah “t” yang diproses pada iterasi sebelumnya. Hal ini bertujuan untuk mencegah adanya data yang hilang pada setiap proses. Tentu saja, proses didukung dengan objek *Wait* yang membuat iterasi bekerja teratur setiap 100 milisekon. Dengan adanya jeda, data akan menunggu untuk diproses sehingga data saling terhubung pada setiap jeda.

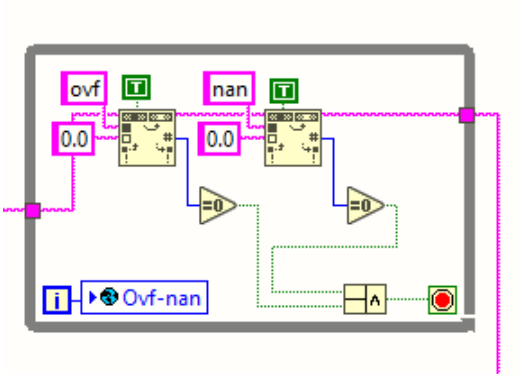


Gambar 4.9 Blok diagram Case Structure ketika posisi data lebih dari 80
(Diambil pada 15 Maret 2019 pukul 10.07)

Fungsi dari *Case Structure* adalah memastikan posisi “t” (*offset of match*) melebihi dari angka 80 atau tidak. Pada Gambar 4.8, ditampilkan *Case Structure* ketika posisi “t” masih kurang dari 80, sehingga perlu untuk digabungkan dengan data dari *shift register*. Akan tetapi, jika posisi “t” sudah kurang dari 80, maka tidak perlu digabungkan data tersebut dengan data dari *shift register* karena kemungkinan data tersebut sudah lengkap secara format. Cukup data dari *shift register* langsung diproses. Gambar 4.9 di atas menjelaskan bahwa posisi “t” sudah lebih dari 80, maka tidak perlu dilakukan penggabungan data menggunakan *Concatenate String*.

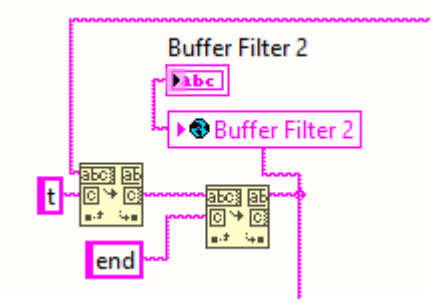
Data kemudian masuk ke *Search/Split String* kedua untuk memastikan data diakhiri dengan “end”. Data yang diambil adalah data sebelum “end” dengan fitur *substring before match* pada objek tersebut. Dalam hal ini, ada kemungkinan error karena data tidak selalu cukup lengkap karena proses dari *shift register* memiliki resiko hilangnya data. Oleh karena itu, digunakan *offset of match* untuk memastikan bahwa data telah mencapai ukuran panjang yang ditentukan yaitu di atas 80 dan di bawah 124. Perbandingan digunakan sebelum gerbang *AND* untuk memastikan hal itu. Jika data masih belum sesuai dengan format tersebut, maka iterasi akan diulang kembali.

Namun, jika data sudah sesuai format, maka iterasi selesai dan data ditampilkan ke *Buffer Filter 1* dan dikirim ke *Global Variable*.



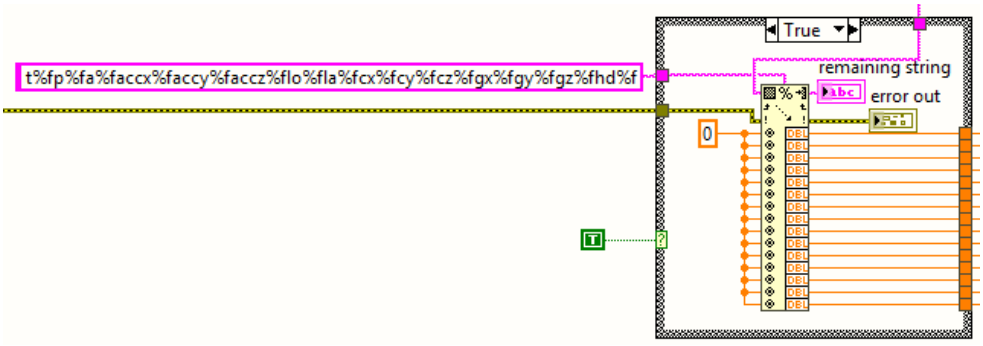
Gambar 4.10 Blok diagram pengolah buffer data yang kedua
(Diambil pada 15 Maret 2019 pukul 07.33)

Data yang sudah diolah dari *loop()* pertama kemudian masuk ke *loop()* kedua. *Loop()* ini berfungsi untuk mengantisipasi error yang terjadi ketika data berubah menjadi "ovf" atau "nan" dalam deretan string yang sudah diolah. Menurut penelitian sebelumnya, error ini terjadi karena kesalahan sensor GY-05. Logika *loop()* kedua ini masih sama dengan *loop()* pertama, yaitu akan terus berulang ketika bertemu kondisi ***FALSE***. Objek *Search and Replace String* yang digunakan di atas berfungsi mengganti komponen variabel yang mengandung unsur "ovf" dan/atau "nan" menjadi nol agar tidak membuat proses *tracing* data menjadi error. Dalam proses *tracing* data tidak dikenal format "ovf" dan "nan". Keluaran dari objek tersebut menghasilkan logika ***TRUE*** yang masuk ke gerbang logika *AND*, sehingga menghasilkan ***TRUE*** yang menyebabkan *loop()* berhenti mengulang proses dan proses berlanjut.



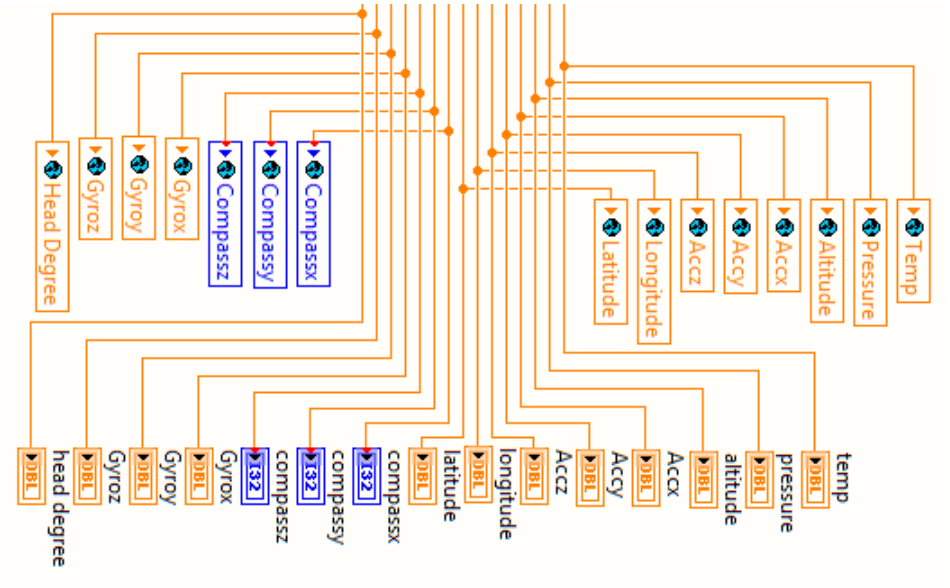
Gambar 4.11 Blok diagram pengolahan buffer data yang ketiga
(Diambil pada 15 Maret 2019 pukul 07.55)

Terdapat dua objek *Search/Split String* yang digunakan untuk melakukan *splitting* atau pencarian komponen *string* yang berawalan "t" dan diakhiri "end". Proses ini dilakukan untuk benar-benar memastikan data yang akan masuk ke proses *parsing* data per variabel di blok diagram berikutnya. Objek pertama akan meloloskan komponen *string* setelah "t" melalui fitur *match + rest of string*, sedangkan objek kedua akan meloloskan komponen *string* sebelum "end" melalui fitur *substring before match*. Setelah proses itu selesai, maka data akan dikirim ke *Global Variable* dan ditampilkan pada *Front Panel*. Selain dari itu, data lanjut ke proses *parsing*.



Gambar 4.12 Blok diagram parsing data
(Diambil pada 15 Maret 2019 pukul 08.30)

Kemudian, data diolah menggunakan objek *Scan From String* untuk mendapatkan nilai satuan dari setiap variabel. Data masuk ke objek tersebut melalui *input string*. Data tersebut dibandingkan dengan format variabel yang ditentukan pada bagian *format string*. Saat data sesuai dengan format, maka akan dibagi menjadi 15 variabel bertipe data ***double***. Jika ada variabel yang kurang, maka otomatis data pada variabel tersebut dan setelahnya di-nol-kan. Saat terjadi error, error langsung ditampilkan melalui *error out*. Jika tidak ada penampil tersebut, proses LabVIEW akan terhenti karena error harus ditampilkan secara langsung sehingga proses harus dihentikan. Hal ini sangat mengganggu kehandalan dari proses pengolahan data yang diharapkan kontinyu tanpa halangan. Selain itu, untuk mengetahui letak error yang ada pada variabel, sisa variabel yang tidak sesuai format data langsung ditampilkan pada *Front Panel* melalui *remaining string*.



Gambar 4.13 Blok diagram penampil akhir dan Global Variable
(Diambil pada 15 Maret 2019 pukul 09.39)

Akhirnya, variabel-variabel yang diperoleh dari hasil *parsing* ditampilkan ke *Front Panel* dan dikirim ke *Global Variabel*. Tipe data dari semua variabel adalah ***double***, kecuali *compassx*, *compassy*, dan *compassz* (***int32,*** sejenis tipe data *int*). Tampilan data di *Global Variable* dapat dilihat di Lampiran.

4.2.4 ANALISIS PENGUJIAN KUANTITATIF DAN KUALITATIF

Pengujian secara kuantitatif memberikan hasil berupa empat parameter, yaitu jumlah percobaan, waktu proses, terjadi error atau tidak, dan posisi variabel yang terjadi error ketika proses *parsing data*. Berikut ini adalah Tabel 5 yang menampilkan hasil pengujian kuantitatif. Tabel di bawah ini merupakan penyederhanaan dari tabel hasil yang terlampir pada lampiran tugas akhir ini. Warna abu-abu pada sebuah baris menunjukkan adanya error pada percobaan tersebut.

Tabel 5. Hasil pengujian kuantitatif LabVIEW

Percobaan ke-	Waktu Proses (milisekon)	Error (Ya/Tidak)	Posisi Error (Variabel ke-)
1	400	Tidak	-
2	300	Tidak	-
3	100	Ya	10
4	200	Tidak	-

5	300	Tidak	-
6	300	Tidak	-
7	200	Tidak	-
8	200	Tidak	-
9	800	Tidak	-
10	300	Tidak	-
11	200	Tidak	-
12	200	Tidak	-
13	500	Tidak	-
14	200	Tidak	-
15	100	Tidak	-
16	100	Ya	4
17	700	Tidak	-
18	200	Tidak	-
19	200	Tidak	-
20	200	Tidak	-
21	200	Tidak	-
22	800	Tidak	-
23	200	Tidak	-
24	600	Tidak	-
25	200	Tidak	-
26	300	Tidak	-
27	200	Tidak	-
28	200	Tidak	-
29	100	Ya	12
30	200	Tidak	-
31	100	Tidak	-
32	200	Tidak	-
33	500	Tidak	-
34	600	Tidak	-
35	200	Tidak	-
36	100	Ya	3
37	300	Tidak	-
38	200	Tidak	-
39	500	Tidak	-
40	200	Tidak	-
41	100	Ya	10
42	200	Tidak	-
43	200	Tidak	-

44	200	Tidak	-
45	100	Tidak	-
46	200	Tidak	-
47	500	Tidak	-
48	200	Tidak	-
49	200	Tidak	-
50	200	Tidak	-

Keterangan dari Tabel 5 adalah sebagai berikut.

1. Kolom pertama meliputi jumlah pengambilan data.
2. Kolom kedua meliputi waktu proses dari *while loop* pertama (proses pencarian deret data). Waktu proses dalam satuan milisekon (ms).
3. Kolom ketiga meliputi terjadi atau tidaknya error setelah proses *parsing data* atau pembagian data menjadi lima belas variable yang akan ditampilkan ke *Front Panel*.
4. Kolom keempat meliputi posisi error dari deret data ketika dilakukan proses *parsing data*. Posisi ini terdeteksi ketika data yang masuk ke *Scan From String* ada yang tidak sesuai dengan format yang diberikan ke *Scan From String*. Angka yang muncul itu merupakan variabel ke-X yang mulai tidak sesuai dengan format yang diberikan.

Kemudian, dari tabel di atas dapat dihitung error yang terjadi dan waktu rata-rata proses yang terjadi pada *while loop* pertama. Berikut ini adalah proses perhitungannya.

$$t_{rata-rata} = \frac{total\ waktu\ proses}{jumlah\ datum} = \frac{13500\ ms}{50} = 270\ ms$$

$$error = \frac{jumlah\ error}{jumlah\ datum} = \frac{5}{50} = \frac{1}{10} = 10\%$$

Ketika diujikan, proses data berjalan cukup cepat sesuai kebutuhan lomba (data terbaca per detik sudah cukup baik). Seperti pada tabel di atas, dapat diperoleh rata-rata data terbaca dari 50 kali pengambilan data adalah 270 milisekon. Proses data berjalan dengan waktu minimal 100 milisekon dan waktu maksimal 800 milisekon. Dari 50 kali pengambilan data tersebut, hanya terjadi lima kali kesalahan (error sebesar

10% dari total pengujian) berupa kehilangan data. Ada **kemungkinan** ini terjadi akibat kesalahan dari Arduino UNO, karena LabVIEW hanya berperan dalam menerima data saja. Hal ini masih sekedar hipotesis saja karena belum dilakukan pengujian antara Arduino UNO yang asli (buatan Italia) dan yang palsu (contoh buatan Cina).

Terakhir, pengujian kualitatif dilakukan dengan cara menjalankan LabVIEW selama 10 menit. Dari pengujian ini, diperoleh hasil tiga kali error yang ditampilkan pada *Error Out*. Tidak ada error yang bersifat menyebabkan LabVIEW berhenti dalam melakukan proses data. Namun, ketika pengujian dilakukan di atas 10 menit, yaitu 15 menit, laptop justru menjadi *Blue Screen of Death* atau *BSoD* (sebuah tampilan error pada Windows) ketika memasuki lima menit akhir. Hal ini terjadi karena program LabVIEW dijalankan secara *run continously* (ikon ). Windows 8.1 tidak mampu menerima proses simulasi selama itu, sehingga akhirnya Windows menampilkan *blue screen*. Oleh karena itu, pengujian hanya dilakukan dalam interval waktu 10 menit saja.