# BAB II
# TEORI PENUNJANG

Bab ini akan membahas mengenai teori dasar dari rancang bangun kendali ketinggian balon atmosfer berbasis multiwii.

## 2.1 Balon Atmosfer

Balon atmosfer adalah wahana yang digunakan untuk membawa muatan instrumen yang merekam atau mentransfer data hasil observasi di lapisan atmosfer. Tipe balon ini sering disebut sebagai radiosonde atau balon sonde. Balon ini cenderung memiliki bentuk bulat/*sphere* yang mampu mengangkat muatan dengan beban yang bervariasi (rentang 200 gram hingga 1 kg) ke ketinggian hingga 35 km dari atas permukaan tanah.



***Gambar II.1 Balon atmosfer***
Sumber: www.balloonchallenge.org

Balon atmosfer memiliki bahan yang lentur dengan bahan karet *latex* maupun *latex* sintetis menggunakan *polychloroprene*. Pada umumnya balon atmosfer diisi oleh gas yang memiliki massa jenis lebih ringan dibandingkan udara sekitar. Terdapat 2 jenis gas yang umum digunakan yakni gas helium dan hidrogen. Balon mampu mengangkat muatan lebih berat dibandingkan berat balon itu sendiri

dengan cara mengisi lebih banyak gas yang akan memperbesar volume dari balon atmosfer.

## 2.2 Teori Pengontrolan Ketinggian

Pada hukum Archimedes menjelaskan bahwa suatu benda yang dicelupkan sebagian/seluruhnya ke dalam suatu fluida (udara atau air), maka benda tersebut akan mengalami gaya ke atas (*bouyancy*) yang besarnya sama dengan massa fluida yang dipindahkan oleh benda tersebut. Gaya ke atas pada balon atmosfer diistilahkan sebagai daya angkat total (T). Nilai T dapat dihitung berdasarkan Persamaan (II.1). Nilai dari masing-masing parameter akan berubah seiring dengan ketinggian dari balon atmosfer.

$$T = V(\rho - \rho_g) \qquad \textbf{(II.1)}$$

Di mana:

T = daya angkat total

V = volume balon

$\rho$ = massa jenis udara

$\rho_g$ = massa jenis gas di dalam balon atmosfer (contoh: hidrogen, helium)

daya angkat bebas *(free lift*) adalah daya angkat yang diperlukan untuk membawa muatan dengan kecepatan tertentu. Nilai daya angkat bebas (L) adalah seperti pada Persamaan (II.2).

$$L = T - W \qquad \textbf{(II.2)}$$

Di mana W adalah berat total muatan yakni dari berat balon dan berat muatan. Jika L mendekati nol atau daya angkat total sama dengan berat total muatan maka balon atmosfer tidak dapat bergerak ke atas maupun ke bawah sehingga balon melayang di udara.

## 2.3 PID (*Proportional-Integral-Derivative*)

### 2.3.1 Terminologi

***Controlled Variable*** **dan** ***Control Signal*** **atau** ***Manipulated Variable***. *Controlled variable* adalah kuantitas atau kondisi yang terukur dan terkontrol. *Control signal* atau *manipulated variable* adalah kuantitas atau kondisi yang

divariasi oleh pengontrol sehingga mempengaruhi nilai dari *controlled variable*. Biasanya, *controlled variable* adalah *output* dari sistem. *Control* berarti mengukur sebuah nilai *controlled variable* dari suatu sistem dan menerapkan *control signal* pada sistem untuk mengoreksi atau membatasi simpangan nilai ukur dari nilai yang diinginkan.

***Plants*.** Sebuah *plant* dapat berupa sebuah peralatan, mungkin juga berupa seperangkat bagian mesin yang berfungsi bersamaan, dengan tujuan untuk melakukan sebuah operasi tertentu. Kita dapat menyebut sebuah benda fisik yang dikontrol sebagai *plant*.

***Processes*.** *Processes* didefinisikan sebagai proses bertahap yang berkelanjutan atau pengembangan yang ditandai dengan serangkaian perubahan bertahap yang berhasil satu sama lain dengan cara yang relatif tetap dan mengarah pada hasil tertentu atau hasil akhir. Kita dapat menyebut sebuah operasi yang dikontrol sebagai p*rocesses*.

***Systems.*** Sebuah sistem adalah kombinasi dari berbagai komponen yang berperan bersama dan melakukan tujuan tertentu. Sebuah sistem tidak harus benda fisik. Konsep dari sistem dapat diterapkan untuk sesuatu yang abstrak, fenomena dinamis seperti yang ditemui pada bidang ekonomi.

***Disturbances.*** *Disturbances* adalah sebuah sinyal yang cenderung mempengaruhi nilai *output* pada sebuah sistem. Jika *disturbances* dihasilkan dalam sistem, maka itu disebut *internal disturbance*, sedangkan *external disturbance* dihasilkan di luar suatu sistem dan *input*.

***Feedback Control.*** *Feedback control* merujuk pada sebuah operasi yang hadir bersama gangguan, cenderung mengurangi perbedaan antar *output* pada sistem dan beberapa referensi *input* dan itu terjadi atas dasar perbedaan tersebut. Di sini hanya untuk spesifikasi gangguan yang tak terprediksi, karena gangguan yang diketahui atau terprediksi selalu dapat dikompensasikan dalam sistem.

### 2.3.2 Sistem Kontrol *Open-loop* dan *Closed-loo*p

**Sistem kontrol *closed-loop*.** *Feedback control systems* yang sering disebut sistem kontrol *closed-loop*. Dalam prakteknya, istilah *feedback control* dan kontrol *closed-loop* digunakan secara bergantian. Pada sistem kontrol *closed-loop*, sinyal *error* aktuator yang merupakan perbedaan antara sinyal *input* dan sinyal *feedback* (yang mungkin sinyal *output* itu sendiri atau fungsi dari sinyal *output* dan *derivative* dan/atau *integral*-nya), diumpankan ke kontroler sehingga dapat mengurangi *error* dan membawa *output* dari sistem pada nilai yang diinginkan. Istilah kontrol *closed-loop* selalu diimplikasikan sebagai penggunaan tindakan *feedback control* guna mengurangi kesalahan sistem.

**Sistem Kontrol *open-loop*.** Sistem yang *output*-nya tidak memiliki efek pada aksi kontrolnya disebut sistem kontrol *open-loop*. Dengan kata lain, pada sistem kontrol *open-loop output* tidak diukur juga tidak memberikan *feedback* untuk dibandingkan dengan input. Salah satu contoh praktis adalah mesin cuci. Perendaman, pencucian, dan pembilasan di mesin cuci beroperasi atas dasar waktu. Mesin tidak mengukur sinyal *output*, yaitu kebersihan pakaiannya.

**Perbandingan sistem kontrol *closed-loop* dan *open-loop*.** Keuntungan dari sistem kontrol *closed-loop* adalah fakta bahwa penggunaan *feedback* membuat respon sistem relatif tidak sensitif terhadap gangguan eksternal dan variasi internal sistem parameter. Dengan demikian hal ini memungkinkan untuk menggunakan komponen yang relatif akurat dan murah untuk mendapatkan kontrol akurat dari *plant* tertentu, sedangkan ini tidak mungkin dilakukan pada kasus *open-loop*. Dari sudut pandang stabilitas, sistem kontrol *open-loop* lebih mudah untuk dibuat karena stabilitas sistem bukan masalah besar. Di sisi lain, stabilitas adalah masalah utama pada sistem kontrol *closed-loop*, yang cenderung untuk melakukan koreksi berlebihan pada *error* dan dengan demikian dapat menyebabkan osilasi dari amplitudo yang konstan atau berubah-ubah.

### 2.3.3 Respon *Transient* dan Respon *Steady-State*

Waktu respon dari sistem kontrol terbagi ke dalam dua bagian, yaitu respon *transient* dan respon *steady-state*. Dengan respon *transient*, yang berarti respon yang bergerak dari keadaan awal ke keadaan akhir. Dan dengan respon *steady-state*,

yang berarti cara di mana *output* sistem berperilaku sebagai t mendekati tak terhingga. Dengan demikian respon sistem C(t) dapat ditulis seperti pada Persamaan (II.3).

$$C(t) = C_{tr}(t) + C_{ss}(t) \qquad \textbf{(II.3)}$$

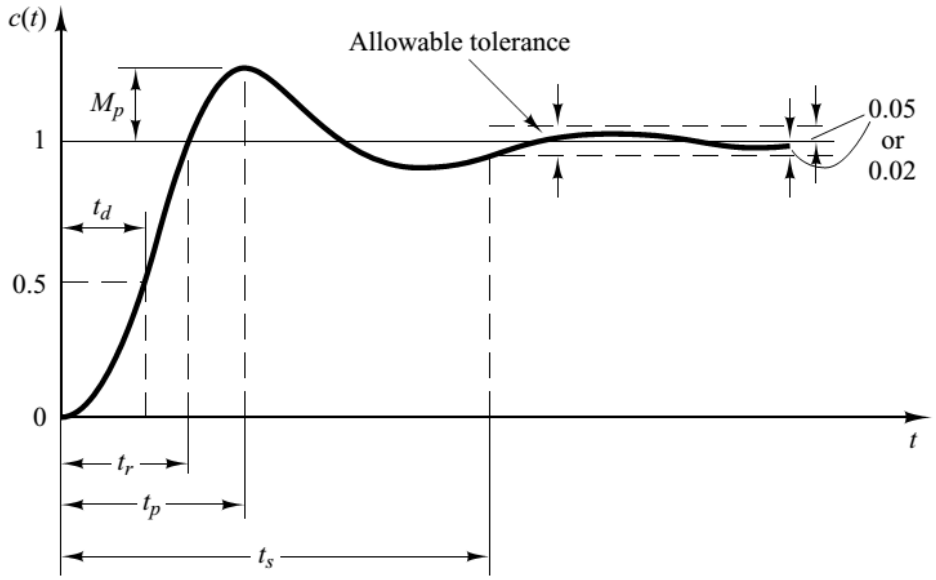
### 2.3.4 Spesifikasi Respon *Transient*

Respon *transient* dari sistem kontrol praktis sering menunjukkan osilasi teredam sebelum mencapai *steady-state*. Dalam menentukan karakteristik respon *transient* dari sistem kontrol untuk *input*, umumnya adalah menentukan hal-hal berikut ini:

1. ***Delay time, td***
2. ***Rise time, tr***
3. ***Peak time, tp***
4. ***Maximum overshoot, Mp***
5. ***Settling time, ts***

Spesifikasi tersebut didefinisikan seperti pada berikut ini dan ditunjukkan pada Gambar II.2 (Ogata, 2002:170).

1. *Delay time, td*: waktu pertama yang dibutuhkan oleh respon untuk mencapai setengah nilai akhir.
2. *Rise time, tr*: waktu yang dibutuhkan oleh respon untuk naik dari 10% ke 90%, 5% ke 95%, atau 0% ke 100% dari nilai akhirnya. Untuk sistem yang memiliki redaman cukup (*underdamped*) nilai yang digunakan adalah 0% ke 100%. Untuk sistem yang memiliki redaman berlebih (*overdamped*) biasanya nilai yang digunakan adalah 10% ke 90%.
3. *Peak time, tp*: waktu pertama yang dibutuhkan respon untuk mencapai puncak maksimum saat *overshoot*.
4. *Maximum overshoot, Mp*: nilai puncak maksimum dari kurva respon yang terukur seluruhnya.
5. *Settling time, ts*: waktu yang dibutuhkan kurva respon untuk sampai dan diam pada nilai akhirnya yang memiliki persentase *steadystate error* sekitar 0,02 atau 0,05.




***Gambar II.2 Kurva kontrol***
(Sumber: Ogata ,2002:170)

### 2.3.5 Kontrol *Proportional Integral Derivatif* (PID)

Sebuah kontroler *proporsional-integral-derivatif* (PID kontroler) adalah mekanisme kontrol umpan balik (*feedback controller*) yang biasa digunakan dalam sistem kontrol industri. Sebuah kontroler PID terus menghitung nilai *error* sebagai perbedaan antara variabel proses yang terukur dan *set point* yang diinginkan. *Kontroler* mencoba untuk meminimalkan kesalahan dari waktu ke waktu dengan menyesuaikan variabel kontrol, seperti posisi katup kontrol. Sehingga dihasilkan sebuah persamaan untuk menghasilkan nilai *output* baru yang ditentukan oleh jumlah tertimbang seperti pada Persamaan (II.4).

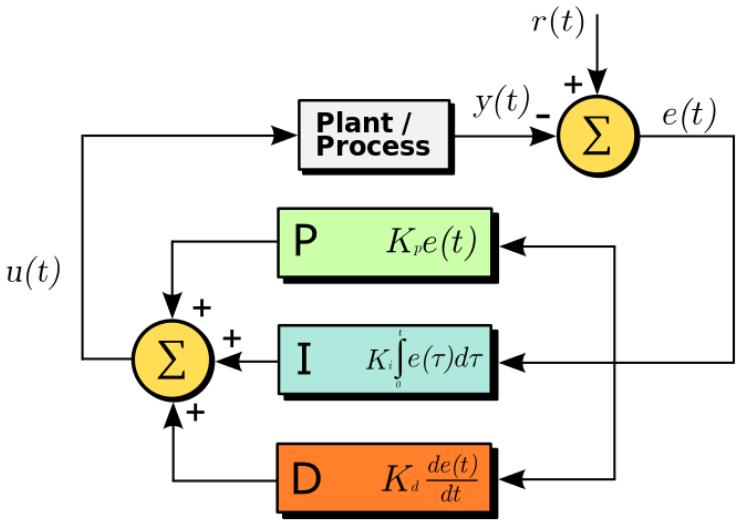
$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \qquad \textbf{(II.4)}$$

Dimana, $K_p$, $K_i$ dan $K_d$, semuanya adalah nilai non-negatif, menunjukkan koefisien untuk istilah *proportional*, *integral*, dan *derivative*, kadang-kadang dilambangkan P, I, dan D. Dalam model ini,

- Nilai P, untuk nilai sekarang dari sebuah *error* (misalnya jika kesalahan besar dan positif, variabel kontrol akan besar dan negatif),
- Nilai I, untuk nilai sebelumnya dari sebuah *error* (misalnya jika *output* tidak cukup untuk mengurangi ukuran kesalahan, variabel kontrol akan

terakumulasi dari waktu ke waktu, menyebabkan kontroler untuk menerapkan tindakan yang lebih kuat), dan

- Nilai D, untuk nilai selanjutnya yang mungkin muncul dari sebuah *error*, berdasarkan dari tingkat perubahan saat ini.




***Gambar II.3 Diagram blok kontrol PID***
Sumber: www.en.wikipedia.org

**Kontrol *Proportional***

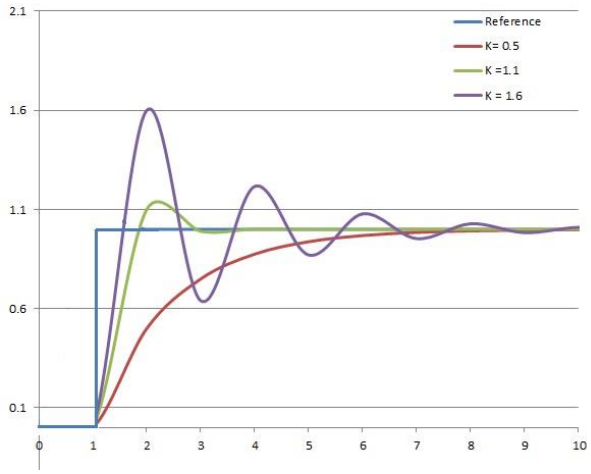
Kontrol *proportional* menghasilkan nilai *output* yang sebanding dengan nilai *error* saat ini. Respon *proportional* dapat disesuaikan dengan mengalikan kesalahan oleh $K_p$ konstan, disebut *gain* konstan *proportional*. Kontrol *proportional* direpresentasikan pada Persamaan (II.5) .

$$P_{out} = K_p e(t) \qquad \textbf{(II.5)}$$

Sebuah *gain proportional* yang tinggi mengakibatkan perubahan besar pada *output* untuk perubahan yang diberikan dalam *error*, sehingga:

- Jika *gain proportional* terlalu tinggi, sistem dapat menjadi tidak stabil. Sebaliknya, *gain* kecil menghasilkan respon *output* yang kecil untuk *error input* yang besar, dan kontroler kurang responsif atau kurang sensitif.
- Jika *gain proportional* terlalu rendah, tindakan kontrol mungkin terlalu kecil ketika merespon gangguan sistem.

Dalam sistem nyata, menggunakan kontrol *proportional* saja akan meninggalkan *offset error* pada kondisi steady-state akhir. Tindakan *Integral* diperlukan untuk menghilangkan kesalahan ini. Gambar II.4 menunjukkan efek kontrol *proportional* dengan nilai gain yang berbeda-beda.



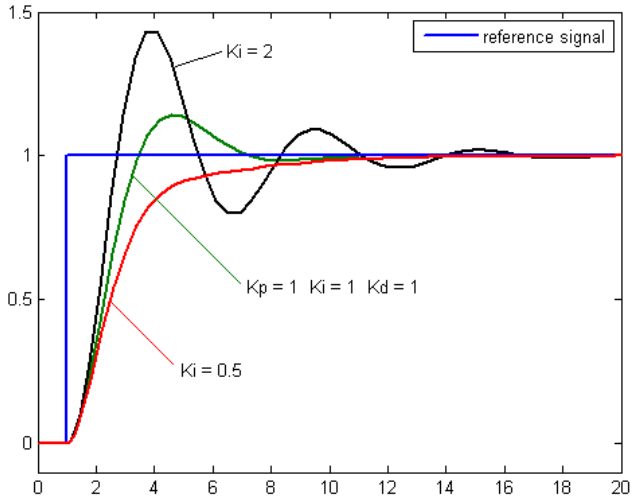

***Gambar II.4 Efek kontrol proportional***
Sumber: www.en.wikipedia.org

**Kontrol *Integral***

Kontribusi dari kontrol *integral* sebanding dengan besarnya kesalahan dan durasi kesalahan. *Integral* dalam kontroler PID adalah jumlah *error* sesaat dari waktu ke waktu dan memberikan akumulasi *offset* yang seharusnya diperbaiki sebelumnya. Akumulasi *error* kemudian dikalikan dengan gain integral ($K_i$) dan ditambahkan ke *output* kontroler. Kontrol *integral* direpresentasikan dengan Persamaan (II.6).

$$I_{out} = K_i \int_0^t e(\tau) d\tau \qquad \textbf{(II.6)}$$

Kontrol *integral* mempercepat pergerakan proses menuju *set point* dan menghilangkan sisa *error* steady state yang terjadi dengan kontroler *proportional* murni. Namun, karena kontrol *integral* merespon akumulasi *error* dari sebelumnya, hal ini dapat menyebabkan *overshoot* nilai sekarang dari nilai *set point*. Gambar II.5 menunjukkan efek kontrol *integral* dengan nilai gain yang berbeda-beda.



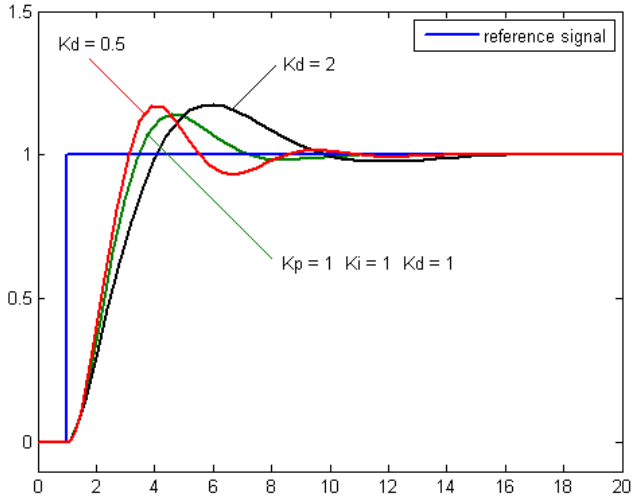

***Gambar II.5 Efek kontrol integral***
Sumber: www.en.wikipedia.org

**Kontrol *Derivative***

Turunan dari *error* proses dihitung dengan menentukan kemiringan *error* dari waktu ke waktu dan mengalikan perubahan nilai ini dengan gain *derivative* $K_d$. Besarnya kontribusi kontrol *derivative* untuk tindakan pengendalian secara keseluruhan disebut gain *derivative*, $K_d$. Kontrol *derivative* direpresentasikan dengan Persamaan (II.7).

$$D_{out} = K_d \frac{d}{dt} e(t) \qquad \textbf{(II.7)}$$

Aksi *derivative* memprediksi perilaku sistem dan dengan demikian meningkatkan waktu penyelesaian dan stabilitas sistem. Sebuah derivatif yang ideal tidak kausal, sehingga implementasi kontroler PID termasuk sebuah tambahan *low pass filter* untuk kontrol *derivative*, untuk membatasi gain frekuensi tinggi dan *noise*. Tindakan *derivative* jarang digunakan dalam prakteknya karena variabelnya berdampak pada stabilitas sistem dalam aplikasi dunia nyata. Gambar II.6 menunjukkan efek kontrol *derivative* dengan nilai gain yang berbeda-beda.




***Gambar II.6 Efek kontrol derivative***
Sumber: www.en.wikipedia.org

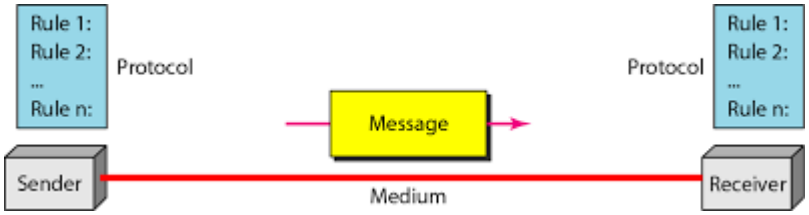
Dengan demikian dapat kita simpulkan hasil pembahasan kontrol terhadap sistem yang ada dalam Tabel II.1 berikut.

***Tabel II.1 Efek kontrol (Proporsional, Integral, Derivatif)***

| Kontrol | *Rise time* | *Overshoot* | *Settling time* | *Steady-state error* |
|---|---|---|---|---|
| ***Proportional*** | Menurunkan | Meningkatkan | Perubahan kecil | Mengurangi |
| ***Integral*** | Menurunkan | Meningkatkan | Meningkatkan | Mengeliminasi |
| ***Derivative*** | Perubahan kecil | Menurunkan | Menurunkan | Perubahan Kecil |

## 2.4 Teori Komunikasi Data Radio

Komunikasi data adalah pertukaran data antara dua perangkat melalui beberapa bentuk media transmisi seperti kabel kawat. Agar komunikasi data terjadi, perangkat komunikasi harus menjadi bagian dari sistem komunikasi yang terdiri dari kombinasi *hardware* (peralatan fisik) dan *software* (program).



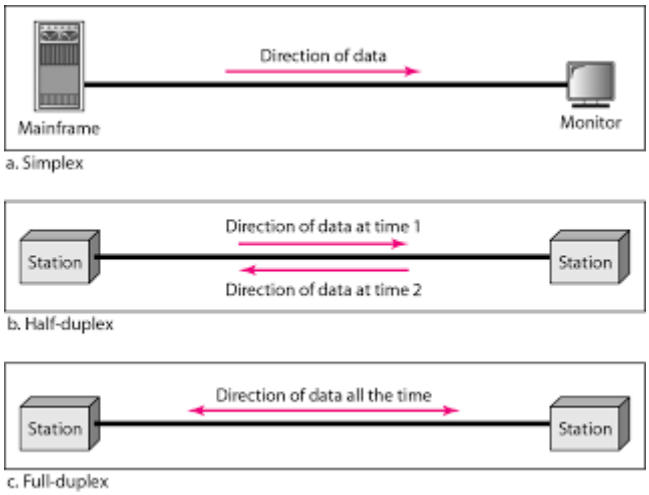

***Gambar II.7 5 Komponen komunikasi data***
Sumber: Forouzan (2007:4)

Berikut adalah 5 komponen komunikasi data (Forouzan, 2007:4):

1. ***Message***. *Message* atau pesan merupakan informasi yang ditransmisikan.
2. ***Sender***. *Sender* atau pengirim merupakan suatu perangkat yang memiliki tugas untuk mengirimkan *Message.*
3. ***Receiver***. *Receiver* atau penerima merupakan suatu perangkat yang memiliki tugas untuk menerima *Message.*
4. ***Transmission medium***. *Transmission Medium* atau media transmisi merupakan media fisik yang menjadi tempat *Message* untuk berpindah dari *Sender* ke *Receiver*.
5. ***Protocol***. *Protocol* atau protokol merupakan suatu set aturan yang mengatur jalannya komunikasi data.

### 2.4.1 *Data Flow* atau Aliran Data

Komunikasi antara 2 perangkat dapat berlangsung secara *simplex*, *half duplex*, dan *full duplex* seperti yang ditunjukkan pada Gambar II.8 (Forouzan, 2007:6).




***Gambar II.8 Data flow (simplex, half-duplex, dan full-duplex)***
Sumber: Forouzan (2007:6)

Berikut adalah penjelasan dari setiap jenis *data flow*:

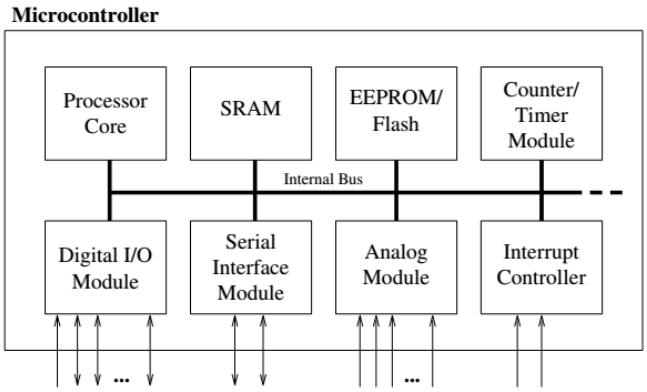
1. *Simplex* merupakan metode pengiriman aliran data di mana setiap pihak hanya memiliki 1 peran, di mana pihak satu akan menjadi pengirim dan pihak lainnya adalah penerima.
2. *Half-Duplex* merupakan metode pengiriman aliran data di mana setiap pihak mampu memiliki peran ganda sebagai pengirim atau penerima tapi tidak disaat bersamaan.
3. *Full-Duplex* merupakan metode pengiriman aliran data di mana setiap pihak mampu menjadi pengirim dan penerima secara bersamaan.

## 2.5 Mikrokontroler

Mikrokontroler adalah sebuah prosesor dengan memori dan komponen lainnya terpasang atau terintegrasi pada sebuah *chip* (*integrated chip*) (Gridling, 2007:2). Sebuah mikrokontroler berisi semua komponen yang memungkinkan untuk beroperasi secara mandiri, dan telah dirancang secara khusus untuk memantau dan/atau mengontrol suatu tugas. Sebagai tambahannya prosesor itu mencakup memori, berbagai kontroler antarmuka, satu atau lebih *timer*, sebuah *interrupt* kontroler, dan pin *general purpose* I/O yang memungkinkan untuk langsung berhadapan dengan lingkungannya. Mikrokontroler juga mencakup operasi bit yang memungkinkan untuk mengubah satu bit dalam byte tanpa menyentuh bit lainnya.

Arsitektur pendesainan mikrokontroler yang populer ada 2 jenis yakni *Complex Instruction Set Computing* (CISC) dan *Reduce Instruction Set Computing* (RISC). Pada CISC terdapat efisiensi dalam mengurangi ukuran program yang ditanamkan dengan cara memasukan sejumlah instruksi kompleks. Instruksi kompleks ekivalen dari tiga atau empat instruksi sederhana. Namun diperlukan *compiler* efisiensi tinggi untuk menggunakan instruksi kompleks. Sedangkan pada RISC, instruksi yang digunakan adalah instruksi sederhana di mana tiap panjang instruksinya sama untuk semua instruksi. Penggunaan instruksi sederhana pada RISC mempermudah proses pendeteksian suatu kegagalan pada sebuah pekerjaan yang dilakukan oleh sistem.

Gambar II.9 (Gridling & Weiss, 2007:5) menunjukkan diagram blok pada mikrokontroler secara umum. Seluruh komponen terhubung via *internal bus* dan seluruhnya terintegrasi pada sebuah chip (*intergrated chip*).




***Gambar II.9 Diagram blok mikrokontroler secara umum***
Sumber: Gridling & Weiss (2007:5)

### 2.5.1 Inti Prosesor dan Memory

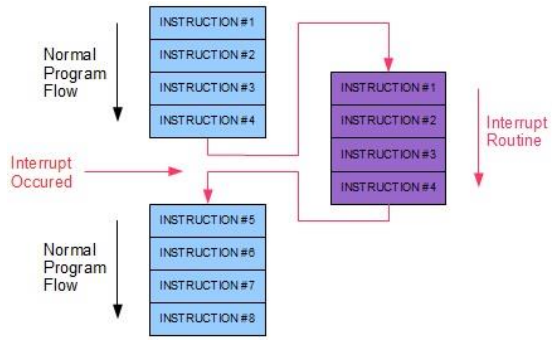
Sebuah CPU (*Central Processing Unit*) pada kontroler terdiri dari 3 unit antara lain unit *arithmetic logic*, unit kontrol dan register. Tiap-tiap unit memiliki peran penting pada pemrosesan suatu tugas yang dilakukan oleh mikrokontroler.

Mikrokontroler memiliki memori yang tertanam di dalamnya dan ukuran memorinya terbatas. Bahkan, hampir tidak mungkin untuk memperluas memori eksternal (hal ini untuk menjaga desain tetap sederhana). Namun, karena mikrokontroler sering digunakan untuk tugas-tugas yang relatif sederhana, maka jumlah memorinya pun tidak perlu berlebihan. Sehingga memasukkan sejumlah kecil data dan instruksi pada memori harus dilakukan secara bijaksana. Dengan begitu, total biaya sistemnya lebih murah, dan bahkan jika memori tidak diperluas, anggota yang berbeda dalam keluarga mikrokontroler biasanya menyediakan berbagai jumlah memori yang berbeda (ada yang lebih besar dan lebih kecil), sehingga dapat memilih mikrokontroler tertentu yang menawarkan ruang memori yang sesuai.

### 2.5.2 Kontroler Interupsi

Interupsi berguna untuk menghentikan aliran program yang berjalan normal karena kasus penting yang terjadi secara eksternal atau internal (Gridling, 2007:5).

Bila berbicara pada kasus *sleep mode*, mereka membantu menghemat daya. Mikrokontroler itu sendiri menawarkan cara yang mudah dalam bentuk interupsi. Seperti yang ditunjukkan pada Gambar II.10, mikrokontroler melakukan *polling* pada sinyal dan menginterupsi program utama hanya jika perubahan *event* terdeteksi. Selama perubahan tidak terdeteksi, program utama hanya mengeksekusi tanpa memperhatikan *event* apa pun. Begitu peristiwa itu terjadi, mikrokontroler memanggil *Interrupt Service Routine* (ISR) yang menangani *event* tersebut. ISR harus disediakan oleh *programmer*.

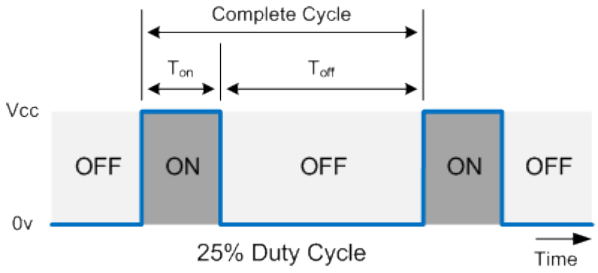



***Gambar II.10 Interrupt Service Routine (ISR)***
Sumber: www.microautomate.com

Bit *interrupt enable* (IE) diatur oleh *programmer* untuk menunjukkan bahwa kontroler harus memanggil ISR sebagai reaksi terhadap *event* tersebut. Bit *interupsi flag* (IF) diatur oleh mikrokontroler setiap kali *event* itu terjadi, dan itu dikerjakan dengan baik secara otomatis saat memasuki ISR atau secara manual oleh *programmer*. Pada dasarnya, bit IF menunjukkan bahwa kondisi *interrupt* telah terjadi, sedangkan bit IE memungkinkan interupsi itu sendiri terjadi.

### 2.5.3 *Timer/Counter*

*Timer* pada dasarnya hanya menghitung pulsa *clock*. Frekuensi pulsa *clock* akan bergantung pada penggunaan kristal pada sistem mikroprosesor. Pada mikroprosesor terdapat beberapa hal yang berhubungan dengan *Timer* yakni frekuensi, perioda dan *duty cycle*. Frekuensi adalah jumlah pulsa yang terjadi dalam satuan waktu. Perioda adalah kebalikan dari frekuensi. *Duty cycle* adalah perbandingan antara waktu pulsa *High* dan perioda yang dinyatakan dalam persen seperti yang diilustrasikan pada Gambar II.11.



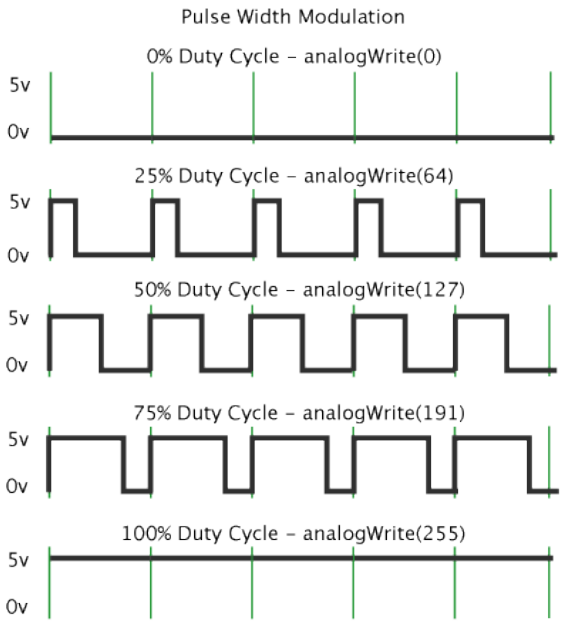

***Gambar II.11 Duty cycle***
Sumber: www.bristolwatch.com

Dengan persamaan sebagai berikut

$$Duty\ Cycle = \frac{T_{ON}}{T}\ x\ 100\% = \frac{T_{ON}}{T_{ON} + T_{OFF}} x100\% \qquad \textbf{(II.8)}$$

### 2.5.4 *Pulse Width Modulation* (PWM)

*Pulse Width Modulation* (PWM) adalah suatu fitur yang dimiliki oleh mikrokontroler. PWM bekerja dengan cara melakukan perubahan level tegangan pada frekuensi tertentu. Konsep PWM dapat digunakan dalam beberapa hal seperti pengatur kecepatan motor, *digital to analog converter* dan sebagainya.




***Gambar II.12 Pulse Width Modulation (PWM)***
Sumber: www.arduino.cc

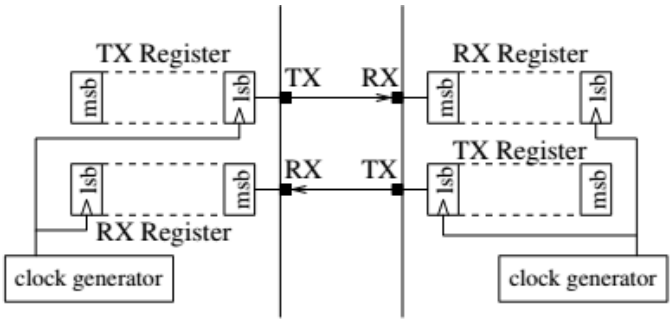
Berdasarkan Gambar II.12, bila kita katakan sumber tegangan bernilai 5V. Maka pada *duty cycle* 0%, keluaran PWM bernilai 0V. Demikian juga, pada

*duty cycle* 25% keluaran PWM pernilai 1,25V, *duty cycle* 50% keluaran PWM bernilai 2,5V, dan pada *duty cycle* 100% keluaran PWM bernilai 5V. Oleh karena itu, dapat disimpulkan dengan adanya manipulasi terhadap perubahan level tegangan maka terjadi perubahan tegangan. Hal tersebut dimanfaatkan untuk mengatur kecepatan motor.

### 2.5.5 *Universal Asynchronous Receiver Transmitter* (UART)

*Serial Communication Interface* (SCI) menyediakan *interface* komunikasi *asynchronous* (*Universal Asynchronous Receiver Transmitter*, UART). Modul UART menggunakan dua jalur, sebuah jalur *trasnmiter* (TXD) dan *receiver* (RXD), untuk komunikasi *full* atau *half duplex*.

Gambar II.13 (Gridling & Weiss, 2007:75) menunjukkan struktur internal UART. Pada dasarnya, modul terdiri dari sebuah register pengirim dan penerima untuk menyimpan data. Sesuai dengan sifatnya yaitu *asynchronous*, pengiriman dan penerimaan pada sebuah *node* dipicu oleh generator *clock* lokal.




***Gambar II.13 Struktur internal UART***
Sumber: Gridling & Weiss (2007:75)

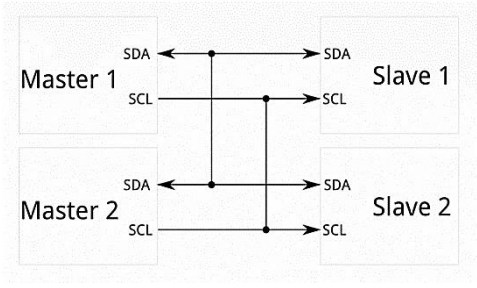
UART tidak hanya sebagai protokol komunikasi semata, tetapi modul UART dapat digunakan sebagai serial komunikasi *asynchronus*. Oleh karena itu, modul UART pada mikrokontroler memungkinkan aplikasi untuk mengontrol perilakunya. Parameter konfigurasinya meliputi:

1. **Jumlah Bit Data:** Tergantung pada UART, jumlah bit data yang dapat dipilih agar memiliki rentang yang besar atau kecil. Seri ATmega misalnya, memiliki rentang antara 5 dan 9 bit data. Modul UART lain mungkin memiliki rentang yang lebih besar atau lebih kecil.

2. **Bit Paritas:** Pengguna dapat memilih apakah harus memiliki sedikit paritas atau tidak, dan jika ya, apakah paritas harus ganjil atau genap. Jika paritas diatur genap, bit paritas adalah 0 jika jumlah nilai bit 1 genap dan sebaliknya jika diatur ganjil.
3. ***Stop* Bit:** Pengguna umumnya dapat memilih apakah harus ada satu atau dua *stop* bit.
4. ***Baud Rate*:** Modul UART memungkinkan pengguna untuk memilih *baud rate* tertentu (yaitu, kecepatan transmisi, diberikan dalam bit per detik (bps)) dari satu set yang mungkin. Nilai *baud rate* umumnya berkisar 9600 hingga 115200 baud. Namun, karena *baud rate* tergantung pada frekuensi *clock*, kecepatan *clock* yang berbeda akan memberikan satu set nilai *baud rate* yang berbeda.

### 2.5.6 *Inter Integrated Circuit* (I2C)

I2C merupakan salah satu fitur baru dalam teknologi komunikasi. I2C memiliki kelebihan pada penggunaan port dibandingkan komunikasi UART. Pada UART tiap perangkat memerlukan satu jalur untuk menerima data dan satu jalur mengirim data sedangkan fitur I2C hanya menggunakan 2 buah *port* untuk semua perangkat sebagai media komunikasi yakni jalur data (menerima/mengirim) dan sinkronisasi *clock*. Berdasarkan protokol I2C, jumlah maksimum perangkat yang terhubung pada sepasang *port* SDA dan SCL adalah sebanyak 128 perangkat.




***Gambar II.14 Wiring diagram I2C***
Sumber: www.learn.sparkfun.com

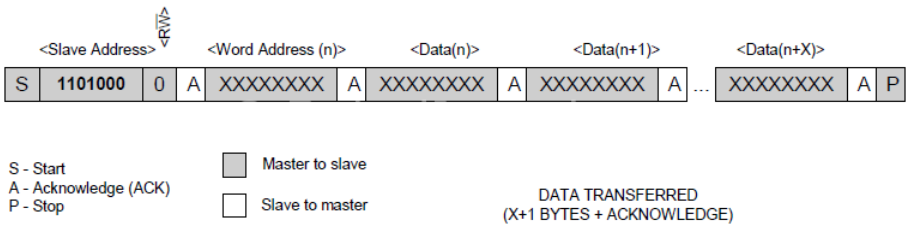
Metode transfer data yang dimiliki I2C ada 2 macam yakni:

1. *Write Register*

   Metode ini dapat dikatakan sebagai proses penulisan data oleh *master* terhadap alamat *register slave*. Berikut ini langkah-langkah proses penulisan *register slave* pada I2C.

   1. Byte pertama yang dikirimkan oleh *master* berupa alamat *slave* yang dituju. *Slave* akan mengirimkan balasan berupa 1-bit *acknowledge* dimana jika *acknowledge* bernilai 1 maka *slave* dapat diakses.
   2. Byte kedua yang dikirimkan oleh *master* berupa alamat *register slave* yang dituju. *Slave* akan mengirimkan balasan berupa 1-bit *acknowledge* dimana jika *acknowledge* bernilai 1 maka *register slave* dapat ditulis.
   3. Byte ketiga yang dikirimkan oleh *master* berupa data yang akan ditulis. *Slave* akan mengirimkan balasan berupa 1-bit *acknowledge* di mana jika *acknowledge* bernilai 1 maka *register slave* berhasil ditulis. Pada Byte keempat dan selanjutnya yang dikirimkan berupa data yang akan ditulis.

   Proses komunikasi data akan berakhir di saat tidak adanya data yang dikirimkan oleh *master* kepada *slave.*




***Gambar II.15 Protokol penulisan data I2C***
Sumber: www.embedjournal.com

2. *Read Register*

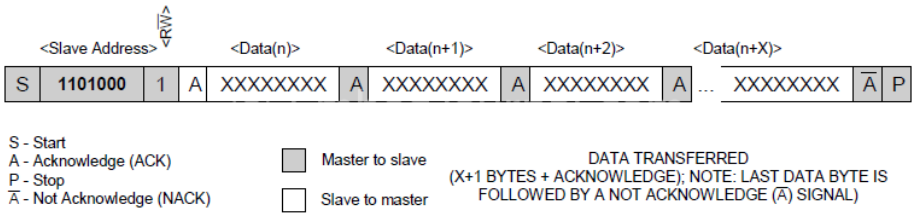
   Metode ini dapat dikatakan sebagai proses permintaan data oleh *master* kepada *slave*. Berikut ini langkah-langkah proses pembacaan *register slave* pada I2C.

   1. Byte pertama yang dikirimkan oleh *master* berupa alamat *slave* yang dituju. *Slave* akan mengirimkan balasan berupa 1-bit

*acknowledge* di mana jika *acknowledge* bernilai 1 maka *slave* dapat diakses.

2. Byte kedua dan selanjutnya adalah proses pengiriman balik oleh *slave* kepada *master* yang di dalamnya berupa data isi tiap *register*. *Master* akan mengirimkan balasan berupa 1-bit *acknowledge* di mana jika *acknowledge* bernilai 1 maka *slave* akan melanjutkan pengiriman data yang ada pada *register* berikutnya.

Proses *Read register* akan berakhir di saat master mengirimkan bit *acknowledge* bernilai 0 atau dapat disebut bit *STOP*.




***Gambar II.16 Protokol pembacaan data I2C***
Sumber: www.embedjournal.com

## 2.6 Multiwii

Multiwii adalah *general purpose software* untuk mengontrol berbagai model *multirotor* yang dikendalikan dengan *radio control*. Awalnya dikembangkan hanya dengan menggunakan sensor *gyroscopes* dan *accelerometer* pada konsol Nintendo Wii, namun sekarang dapat digunakan dengan berbagai sensor. Proyek multiwii ini adalah kesempatan untuk mengembangkan perangkat lunak sendiri dengan platform Arduino. Stabilitas yang dicapai sangat baik untuk FPV dan memungkinkan segala jenis akrobat pada *multirotor*.

Sampai saat ini *software* multiwii mampu mengendalikan *tricopter*, *quadricopter* atau *hexacopter*.

### 2.6.1 Multiwii *Serial Protocol* (MSP)

Multiwii *serial protocol* merupakan protokol yang digunakan pada multiwii untuk melakukan komunikasi. Komunikasi pada MSP ini terjadi secara *full duplex*. Multiwii tidak pernah mengirimkan sesuatu (data) dengan sendirinya. Permintaan

harus dilakukan dalam setiap kasus untuk mengambil atau memberikan data. Setiap pesan yang diterima diakui bahkan meski tidak ada data di dalamnya.

Ada 2 pesan utama yang harus diperhatikan yaitu:

1. Melakukan permintaan *message* ke multiwii

   Untuk meminta data sederhana tanpa parameter pada multiwii, format *message* yang harus dikirim ke multiwii adalah seperti berikut:

   *$M<[data length][code][data][checksum]*

   *1 octet '$'*

   *1 octet 'M'*

   *1 octet '<'*

   *1 octet [data length]*

   *1 octet [code]*

   *several octets [data]*

   *1 octet [checksum]*

   [*data length*] dapat bernilai 0 pada bila tidak ada parameter perintah .

2. *Output message* dari multiwii

   Untuk mengirim perintah spesifik/menyuntikkan parameter baru pada multiwii, format *message* yang harus dikirim ke multiwii adalah seperti berikut:

   *$M>[data length][code][data][checksum]*

   *1 octet '$'*

   *1 octet 'M'*

   *1 octet '>'*

   *1 octet [data length]*

   *1 octet [code]*

   *several octets [data]*

   *1 octet [checksum]*

   Bila pesan tidak diketahui maka formatnya menjadi seperti berikut:

   *$M|[0][code][checksum]*

   *1 octet '$'*

   *1 octet 'M'*

*1 octet '|'*

*1 octet 0*

*1 octet [unknown code]*

*1 octet [checksum]*

Maka secara umum format *message* MSP adalah:

<*preamble*>,<*direction*>,<*size*>,<*command*>,<*data*>,<crc>

Di mana:

- *Preamble* = karakter ASCII '$M'
- *Direction* = karakter ASCII '<' jika ***ke*** Multiwii *Controller* atau '>' jika ***dari*** Multiwii *Controller*
- *Size* = jumlah data dalam byte, bentuk biner. Dapat ditulis dengan nilai 0 pada kasus permintaan data ke Multiwii *Controller*
- *Command* = id *message* dengan nilai seperti pada Tabel II.2
- *Data* = data yang dikirim atau diterima dengan besar dan tipe data seperti pada Tabel II.2 Nilai UINT16 memiliki format LSB *first*.
- Crc = *checksum* dengan melakukan XOR pada <*size*>, <*command*> dan setiap byte data

***Tabel II.2 Kode message, data dan tipe data Multiwii***

| ***Command*** | ***ID Message*** | **Arah** | **Data** | **Tipe Data** |
|---|---|---|---|---|
| **MSP_IDENT** | 100 | FC → | VERSION | UINT 8 |
| | | | MULTITYPE | UINT 8 |
| | | | MSP_VERSION | UINT 8 |
| | | | capability | UINT 32 |
| | | | | |
| **MSP_STATUS** | 101 | FC → | cycleTime | UINT 16 |
| | | | i2c_errors_count | UINT 16 |
| | | | sensor | UINT 16 |
| | | | flag | UINT 32 |

| | | | | |
|---|---|---|---|---|
| | | | global_conf.currentSet | UINT 8 |
| | | | | |
| **MSP_RAW_IMU** | 102 | FC → | accx | INT 16 |
| | | | accy | INT 16 |
| | | | accz | INT 16 |
| | | | gyrx | INT 16 |
| | | | gyry | INT 16 |
| | | | gyrz | INT 16 |
| | | | magx | INT 16 |
| | | | magy | INT 16 |
| | | | magz | INT 16 |
| | | | | |
| **MSP_SERVO** | 103 | FC → | Servo*8 | 16 x UINT 16 |
| | | | | |
| **MSP_MOTOR** | 104 | FC → | Motor*8 | 16 x UINT 16 |
| | | | | |
| **MSP_SET_MOTOR** | 214 | → FC | Motor*8 | 16 x UINT 16 |
| | | | | |
| **MSP_RC** | 105 | FC → | rcData[RC_CHANS] | 16 x UINT 16 |
| | | | | |
| **MSP_SET_RAW_RC** | 200 | → FC | rcData[RC_CHANS] | 16 x UINT 16 |
| | | | | |
| **MSP_RAW_GPS** | 106 | FC → | GPS_FIX | UINT 8 |
| | | | GPS_numSat | UINT 8 |
| | | | GPS_coord[LAT] | UINT 32 |
| | | | GPS_coord[LON] | UINT 32 |
| | | | GPS_altitude | UINT 16 |
| | | | GPS_speed | UINT 16 |
| | | | GPS_ground_course | UINT 16 |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **MSP_SET_RAW_GPS** | 201 | → FC | GPS_FIX | UINT 8 |
| | | | GPS_numSat | UINT 8 |
| | | | GPS_coord[LAT] | UINT 32 |
| | | | GPS_coord[LON] | UINT 32 |
| | | | GPS_altitude | UINT 16 |
| | | | GPS_speed | UINT 16 |
| | | | | |
| **MSP_COMP_GPS** | 107 | FC → | GPS_distanceToHome | UINT 16 |
| | | | GPS_directionToHome | UINT 16 |
| | | | GPS_update | UINT 8 |
| | | | | |
| **MSP_ATTITUDE** | 108 | FC → | angx | INT 16 |
| | | | angy | INT 16 |
| | | | heading | INT 16 |
| | | | | |
| **MSP_ALTITUDE** | 109 | FC → | EstAlt | INT 32 |
| | | | vario | INT 16 |
| | | | | |
| **MSP_ANALOG** | 110 | FC → | vbat | UINT 8 |
| | | | intPowerMeterSum | UINT 16 |
| | | | rssi | UINT 16 |
| | | | amperage | UINT 16 |
| | | | | |
| **MSP_RC_TUNING** | 111 | FC → | byteRC_RATE | UINT 8 |
| | | | byteRC_EXPO | UINT 8 |
| | | | byteRollPitchRate | UINT 8 |
| | | | byteYawRate | UINT 8 |
| | | | byteDynThrPID | UINT 8 |
| | | | byteThrottle_MID | UINT 8 |
| | | | byteThrottle_EXPO | UINT 8 |

| | | | | |
|---|---|---|---|---|
| **MSP_SET_RC_TUNING** | 204 | → FC | byteRC_RATE | UINT 8 |
| | | | byteRC_EXPO | UINT 8 |
| | | | byteRollPitchRate | UINT 8 |
| | | | byteYawRate | UINT 8 |
| | | | byteDynThrPID | UINT 8 |
| | | | byteThrottle_MID | UINT 8 |
| | | | byteThrottle_EXPO | UINT 8 |
| | | | | |
| **MSP_PID** | 112 | FC → | PIDITEMS x conf.pid[] | 3*PIDITEMS x UINT 8 |
| | | | | |
| **MSP_SET_PID** | 202 | → FC | PIDITEMS x conf.pid[] | 3*PIDITEMS x UINT 8 |
| | | | | |
| **MSP_BOX** | 113 | FC → | BOXITEMS x conf.activate[] | BOXITEMS x UINT 16 |
| | | | | |
| **MSP_SET_BOX** | 203 | → FC | BOXITEMS x conf.activate[] | BOXITEMS x UINT 16 |
| | | | | |
| **MSP_MISC** | 114 | FC → | intPowerTrigger1 | UNIT 16 |
| | | | conf.minthrottle | UNIT 16 |
| | | | MAXTHROTTLE | UNIT 16 |
| | | | MINCOMMAND | UNIT 16 |
| | | | conf.failsafe_throttle | UNIT 16 |
| | | | plog.arm | UNIT 16 |
| | | | plog.lifetime | UNIT 32 |
| | | | conf.mag_declination | UNIT 16 |

| | | | | |
|---|---|---|---|---|
| | | | conf.vbatscale | UNIT 8 |
| | | | conf.vbatlevel_warn1 | UNIT 8 |
| | | | conf.vbatlevel_warn2 | UNIT 8 |
| | | | conf.vbatlevel_crit | UNIT 8 |
| | | | | |
| **MSP_SET_MISC** | 207 | → FC | intPowerTrigger1 | UNIT 16 |
| | | | conf.minthrottle | UNIT 16 |
| | | | MAXTHROTTLE | UNIT 16 |
| | | | MINCOMMAND | UNIT 16 |
| | | | conf.failsafe_throttle | UNIT 16 |
| | | | plog.arm | UNIT 16 |
| | | | plog.lifetime | UNIT 32 |
| | | | conf.mag_declination | UNIT 16 |
| | | | conf.vbatscale | UNIT 8 |
| | | | conf.vbatlevel_warn1 | UNIT 8 |
| | | | conf.vbatlevel_warn2 | UNIT 8 |
| | | | conf.vbatlevel_crit | UNIT 8 |
| | | | | |
| **MSP_MOTOR_PINS** | 115 | FC → | 8*PWM_PIN | 8 x UNIT 8 |
| | | | | |
| **MSP_BOXNAMES** | 116 | FC → | string of BOX items | string |
| | | | | |
| **MSP_PIDNAMES** | 117 | FC → | string of PID items | string |
| | | | | |
| **MSP_WP** | 118 | FC → | wp_no | UINT 8 |
| | | | lat | UINT 32 |
| | | | lon | UINT 32 |
| | | | AltHold | UINT 32 |

| | | | | |
|---|---|---|---|---|
| | | | heading | UINT 16 |
| | | | time to stay | UINT 16 |
| | | | nav flag | UINT 8 |
| | | | | |
| **MSP_SET_WP** | 209 | → FC | wp_no | UINT 8 |
| | | | lat | UINT 32 |
| | | | lon | UINT 32 |
| | | | AltHold | UINT 32 |
| | | | heading | UINT 16 |
| | | | time to stay | UINT 16 |
| | | | nav flag | UINT 8 |
| | | | | |
| **MSP_BOXIDS** | 119 | FC → | ID*CHECKBOXITEMS | CHECKBOXITEMS x UINT 8 |
| | | | | |
| **MSP_SERVO_CONF** | 120 | FC → | 8 x conf.servoConf[] | 8 x [UINT 16, UINT 16, UINT 16, UINT 8] |
| | | | | |
| **MSP_SET_SERVO_CONF** | 212 | → FC | 8 x conf.servoConf[] | 8 x [UINT 16, UINT 16, UINT 16, UINT 8] |
| | | | | |
| **MSP_ACC_CALIBRATION** | 205 | → FC | no param | |
| | | | | |
| **MSP_MAG_CALIBRATION** | 206 | → FC | no param | |
| | | | | |
| **MSP_RESET_CONF** | 208 | → FC | no param | |
| | | | | |
| **MSP_SELECT_SETTING** | 210 | → FC | global_conf.currentSet | UINT 8 |
| | | | | |
| **MSP_SET_HEAD** | 211 | → FC | magHold | INT 16 |
| | | | | |
| **MSP_BIND** | 240 | → FC | no param | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **MSP_EEPROM_WRITE** | 250 | → FC | no param | |

## 2.7 Altimeter

Altimeter atau pengukur ketinggian merupakan alat yang digunakan untuk mengukur ketinggian suatu benda di atas sesuatu yang tetap. Pengukuran ketinggian disebut altimetri, yang terkait dengan istilah batimetri, pengukuran kedalaman di bawah air.

Ketinggian dapat ditentukan berdasarkan pengukuran tekanan atmosfer. Semakin besar ketinggiannya maka semakin rendah tekanannya. Ketika barometer ini dilengkapi dengan kalibrasi *nonlinier* untuk menunjukkan ketinggian, maka instrumen ini disebut *pressure* altimeter atau *barometric* altimeter. Sebuah *pressure* altimeter adalah altimeter yang ditemukan di sebagian besar pesawat, dan *skydivers* dengan versi *wrist-mounted* untuk tujuan serupa. Pejalan kaki dan pendaki gunung menggunakan *wrist-mounted* atau *hand-held* altimeter selain alat navigasi lainnya seperti peta, kompas magnetik, atau penerima GPS.

Persamaan (II.9) adalah persamaan untuk melakukan kalibrasi altimeter

$$z = c\,T \log(P_o/\mathrm{P}) \qquad \textbf{(II.9)}$$

di mana c adalah konstanta, T adalah temperatur absolut, P adalah tekanan di ketinggian z, dan $P_o$ adalah tekanan di permukaan laut. Konstanta c tergantung pada percepatan gravitasi dan massa molar udara. Namun, yang ada yang harus diwaspadai bahwa jenis altimeter bergantung pada "densitas ketinggian" dan pembacaan yang dapat bervariasi pada ratusan *feet* karena perubahan mendadak dalam tekanan udara, seperti dari suhu dingin yang menimpa dari depan altimeter, tanpa ada perubahan yang sebenarnya di ketinggian.

## 2.8 GPS

*Global Positioning System* (GPS) adalah sistem navigasi berbasis ruang yang menyediakan informasi lokasi dan waktu di segala kondisi cuaca, di mana saja di bumi atau dekat Bumi pada area yang tak terhalangi atau *line of sight* dengan 4 atau lebih satelit GPS. Sistem menyediakan kemampuan penting untuk pengguna militer, sipil, dan komersial di seluruh dunia. Pemerintah Amerika Serikat

menciptakan sistem, memelihara, dan membuatnya dapat diakses secara bebas kepada siapa pun dengan penerima GPS.

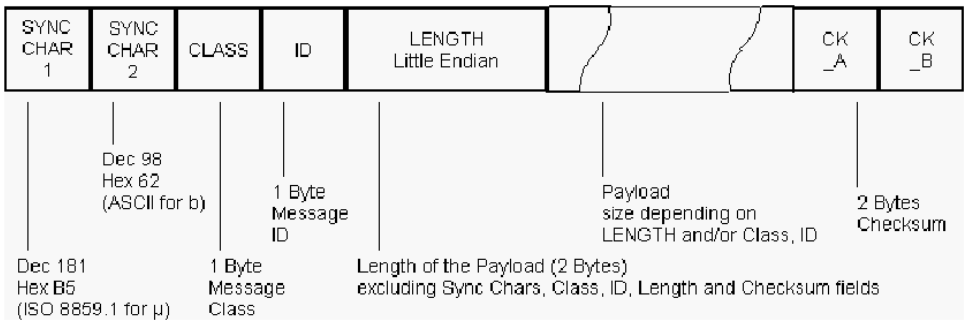
Konsep GPS didasarkan pada waktu. Satelit membawa jam atom yang sangat stabil yang disinkronkan satu sama lain dengan jam di darat. Demikian juga dengan lokasi satelit dimonitor secara presisi. Penerima GPS memiliki jam namun tidak disinkronkan dengan waktu yang benar, dan kurang stabil. Satelit GPS terus mengirimkan waktu dan posisi mereka saat ini. Sebuah penerima GPS memonitor beberapa satelit dan memecahkan persamaan untuk menentukan posisi yang tepat dari penerima dan deviasi dari waktu yang benar. Minimal, empat satelit harus di lihat dari penerima untuk menentukan posisi yang tepat. Jumlah empat terdiri atas tiga koordinat posisi dan penyimpangan jam dari waktu satelit.

### 2.8.1 Protokol UBX

U-blox GPS *receiver* menggunakan protokol khusus u-blox untuk mengirimkan data GPS ke komputer *host* menggunakan *port* RS232 *asynchronous*. Protokol ini memiliki fitur utama sebagai berikut:

- *Compact* - menggunakan 8 Bit Data Binary
- *Checksum Protected* - menggunakan algoritma *checksum low-overhead*
- Modular - menggunakan identifier pesan 2-tahap (Class dan Message ID)

Protokol UBX memiliki struktur paket yang berbeda dengan struktur paket pada protokol NMEA. Struktur paket dasar protokol UBX ditunjukkan pada Gambar II.17 (U-blox 6 Receiver Description, 2013:84) .




***Gambar II.17 Struktur paket UBX***
Sumber: U-blox 6 Receiver Description (2013:84)

- Setiap Pesan dimulai dengan 2 Bytes: 0xB5 0x62
- 1 Byte *Class Field*. *Class* mendefinisikan subset dasar pesan
- 1 Byte ID *Field* mendefinisikan pesan yang mengikutinya
- 2 Byte *Length Field*. *Length* didefinisikan sebagai panjang *payload*. Tidak termasuk *Sync Chars*, *Length Field*, *Class*, ID atau CRC *fields*. Nilai format *length field* panjang adalah *unsigned* 16-Bit integer di format Little Endian.
- *Payload* adalah variabel *length field*.
- CK_A dan CK_B adalah checksum 16 Bit.

*Class* adalah pengelompokan pesan yang terkait satu sama lain. Tabel Tabel II.3 memberikan nama pendek, deskripsi dan definisi *class* ID.

***Tabel II.3 UBX Class ID***

| **Nama** | ***Class*** | **Deskripsi** |
|---|---|---|
| NAV | 0x01 | Hasil Navigasi: Posisi, Kecepatan, Waktu, Acc, *Heading*, DOP, SVs *Used* |
| RXM | 0x02 | Pesan *Receiver Manager*: Status Satelit, Status RTC |
| INF | 0x04 | Pesan Informasi: Pesan *Printf-Style*, dengan IDs Seperti *error*, Peringatan, Himbauan |
| ACK | 0x05 | Messages Ack/Nack: Sebagai jawaban kepada pesan input CFG |
| CFG | 0x06 | Pesan *Input* Konfigurasi: Mengatur *Dynamic Model*, Mengatur *DOP Mask*, Mengatur Baud Rate, dan sebagainya |
| MON | 0x0A | Pesan *Monitoring*: Status komunikasi, *CPU Load*, Penggunaan *Stack*, Status Tugas |
| AID | 0x0B | Pesan *AssistNow Aiding*: Ephemeris, Almanac, data *input* A-GPS lainnya |
| TIM | 0x0D | Pesan Pewaktuan: *Timepulse Output*, *Timemark Results* |
| ESF | 0x10 | Pesan gabungan sensor eksternal: pengukuran sensor eksternal dan status informasi |

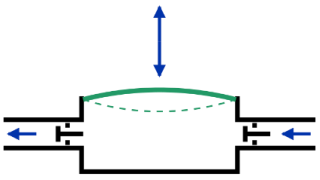
## 2.9 Pompa Diafragma

Sebuah pompa diafragma (juga dikenal sebagai pompa Membran) adalah pompa perpindahan positif yang menggunakan kombinasi dari aksi resiprokasi pada diafragma karet, termoplastik atau teflon dan katup di kedua sisi diafragma (*check valve*, *butterfly valves*, *flap valves*, atau bentuk lain dari katup *shut-off*) untuk memompa cairan atau udara.

Ada tiga jenis utama dari pompa diafragma:

1. Pompa diafragma yang mana diafragma dibungkus dengan salah satu sisinya yang akan memompa cairan, dan sisi lainnya di isi udara atau cairan hidrolik. Diafragma tertekuk, menyebabkan volume ruang pompa meningkat dan menurun. Sepasang katup *non-return* mencegah arus balik dari cairan.
2. Pompa diafragma yang memanfaatkan perpindahan positif volumetrik yang mana penggerak utama dari diafragma adalah elektro-mekanik, bekerja melalui engkol atau roda gigi motor, atau murni mekanis, seperti dengan tuas atau *handle*. Metode ini menekuk diafragma melalui aksi mekanis sederhana, dan satu sisi diafragma terbuka untuk udara.
3. Pompa diafragma yang memanfaatkan satu atau lebih diafragma dengan cairan yang akan dipompa pada kedua sisinya. Diafragma tertekuk, menyebabkan volume berubah.

Ketika volume ruang dari kedua jenis pompa meningkat (diafragma bergerak naik), tekanan berkurang, dan cairan ditarik ke dalam *chamber*. Ketika tekanan ruang kemudian meningkat dari volume yang menurun (diafragma bergerak turun), cairan yang sebelumnya ditarik dipaksa keluar. Akhirnya, diafragma bergerak naik sekali lagi menarik cairan ke dalam *chamber*, menyelesaikan siklus seperti yang diilustrasikan pada Gambar II.18.



***Gambar II.18 Diagram pompa diafragma***
Sumber: www.en.wikipedia.org

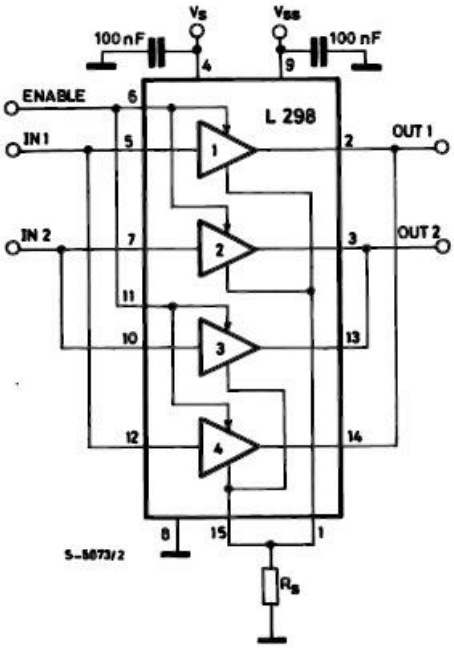
Karakteristik pompa diafragma:

- Memiliki daya hisap yang baik, beberapa pompa bertekanan rendah dengan *flow rate* rendah, sedangkan yang lainnya memiliki *flow rate* tinggi, tergantung pada efektivitas kerja diameter diafragma dan panjang stroke.
- Cocok untuk debit tekanan hingga 1,200 bar.
- Memiliki karakteristik yang baik saat beroperasi pada keadaan kering.
- Dapat digunakan untuk membuat jantung buatan.
- Digunakan untuk membuat pompa udara untuk filter pada tangki ikan kecil.
- Efisiensinya hingga 92%.
- Dapat menangani cairan yang sangat kental.

## 2.10 *Driver* Motor DC Dengan IC L298

L298 adalah sirkuit monolitik terintegrasi yang memiliki tegangan tinggi, driver *dual full-bridge* arus tinggi yang dirancang agar mampu bekerja pada *logic level* TTL yang rendah (standar) dan menginduksi beban seperti relay, solenoid, motor dc dan motor *stepper*. Dua *input enable* disediakan untuk mengaktifkan atau menonaktifkan perangkat secara independen dari sinyal *input*. Emitter dari transistor yang paling bawah dari setiap *bridge* terhubung bersama dan terminal eksternal yang sesuai dapat digunakan untuk koneksi dari sebuah *sensing* resistor eksternal. Sebuah pasokan daya tambahan disediakan sehingga *logic* TTL tetap bekerja pada tegangan rendah.

L298 memiliki dua buah rangkaian H-Bridge di dalamnya, sehingga dapat digunakan untuk men-*drive* dua buah motor DC. Masing-masing dapat mengantarkan arus hingga 2A. Namun, dalam penggunaannya, L298 dapat digunakan secara paralel seperti yang ditunjukkan pada Gambar II.19 (STMicroelectronics, 2000), sehingga kemampuan menghantarkan arusnya menjadi 4A. Konsekuensi dari pemasangan H-Bridge *driver* motor DC L298 dengan mode paralel adalah perlunya 2 buah H-Bridge *driver* motor DC L298 bilamana mengendalikan 2 motor DC.




***Gambar II.19 Rangkaian L298 mode paralel***
Sumber: STMicroelectronics (2000)

Berikut adalah pin-pin yang dihubungkan dalam modus operasi paralel:

- OUT1 dihubungkan dengan OUT4.
- OUT2 dihubungkan dengan OUT3.
- IN1 dihubungkan dengan IN4.
- IN2 dihubungkan dengan IN3.
- *ENABLE* A dihubungkan dengan *ENABLE* B.

OUT1/OUT4 dan OUT2/OUT3 dihubungkan dengan motor DC yang akan dikendalikan.

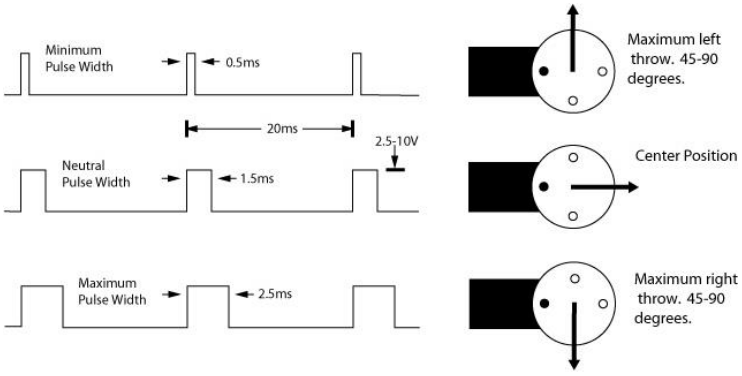
## 2.11 Kontrol Servo

Kontrol servo dari penerima radio kontrol pada servo dilakukan dengan mengirimkan sinyal PWM (*pulse width modulation*) pada masing-masing servo.

Servo kecil terhubung melalui tiga kabel standar: dua kabel untuk *power supply* arus DC dan satu lagi untuk kontrol, di mana jalur ini membawa pulsa. Parameter untuk pulsa terdiri dari lebar pulsa minimum, lebar pulsa maksimum, dan tingkat pengulangan. Untuk memberi batas putaran pada servo, netral didefinisikan sebagai posisi di mana servo memiliki jumlah yang persis sama

dengan putaran potensial mekanik (potensiometer) pada arah yang sama dengan jarum jam maupun berlawanan arah jarum jam.

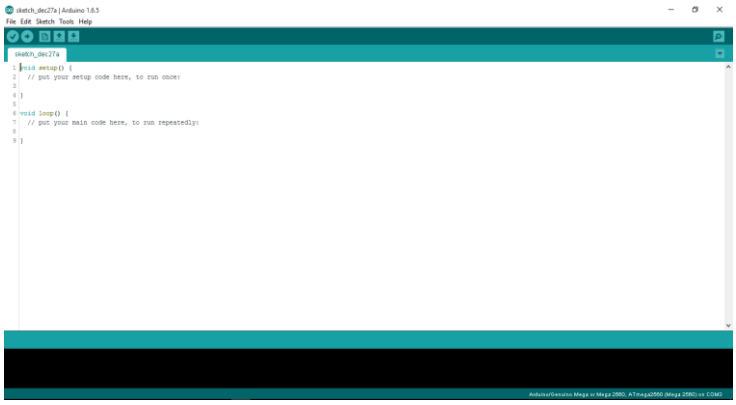
Lebar minimum dan maksimum pulsa dapat beragam dari setiap pabrik pembuatnya. Namun, bagaimanapun nilai untuk posisi netral berkisar 1,5ms. Biasanya lebar minimum pulsa bervariasi dari 0,5ms hingga 1ms, sedangkan varian dari lebar maksimal pulsa antara 2ms hingga 3ms. Umumnya frekuensi sinyal pulsa pada servo adalah 50Hz, namun ini pun dapat bervariasi antara 30Hz hingga 200Hz. Tegangannya pun bervariasi dari 2.5V hingga 10V. Gambar II.20 menunjukkan variasi posisi servo dengan variasi pulsa yang diberikan.




***Gambar II.20 Lebar pulsa motor servo***
Sumber: www.mitchr.me

## 2.12 Arduino *Software* IDE

Program Arduino dapat ditulis dalam bahasa pemrograman apa pun dengan *compiler* yang menghasilkan kode biner. Proyek Arduino menyediakan IDE (*integrated development environment*) Arduino, yang merupakan aplikasi yang ditulis di Java. Arduino IDE dirancang untuk memperkenalkan pemrograman untuk artis dan pendatang baru lainnya yang tidak terbiasa dengan pengembangan perangkat lunak. Arduino IDE termasuk editor kode dengan fitur seperti *syntax highlighting*, *brace matching*, dan *automatic indentation*, dan memberikan mekanisme sederhana dengan satu-klik untuk mengkompilasi dan memuat program untuk *board* Arduino. Sebuah program yang ditulis dengan IDE untuk Arduino disebut "*sketch*". Arduino IDE mendukung bahasa pemrograman C dan C++ dengan aturan khusus untuk mengorganisir kode.

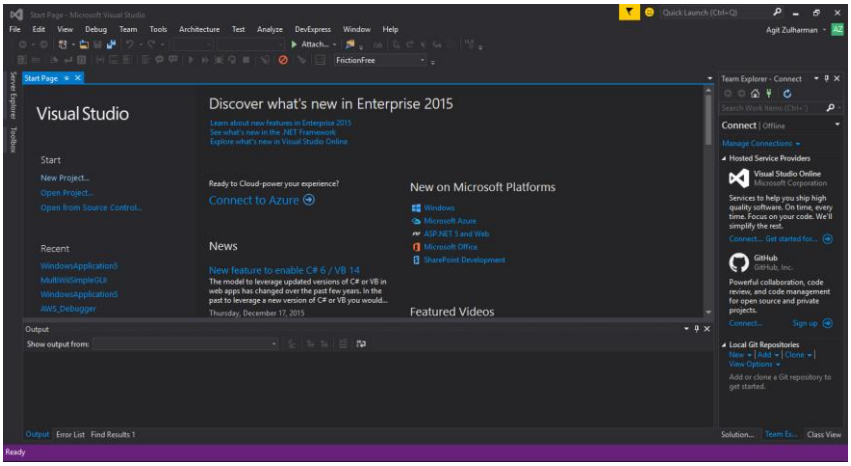


***Gambar II.21 Arduino IDE***

## 2.13 Visual Studio 2015

Microsoft Visual Studio adalah sebuah IDE (*integrated development environment*) dari Microsoft. Microsoft visual studio digunakan untuk mengembangkan program komputer bagi Microsoft Windows, serta situs web, aplikasi web dan layanan web. Visual Studio menggunakan platform pengembangan perangkat lunak Microsoft seperti Windows API, Windows *Forms*, Windows *Presentation Foundation*, Windows *Store* dan Microsoft Silverlight. Hal ini dapat menghasilkan *native code* dan *managed code*.

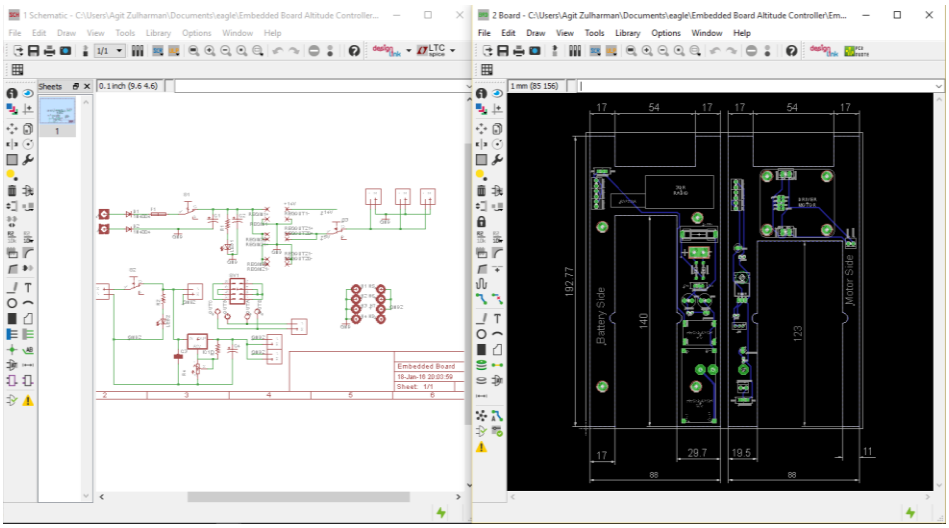
Visual Studio mendukung bahasa pemrograman yang berbeda dan menyediakan kode editor dan *debugger* untuk mendukung (untuk berbagai tingkat) semua bahasa pemrograman, juga memberikan layanan bahasa. Bahas pemrograman yang terkandung pada visual studio antara lain C, C ++ dan C ++ / CLI (melalui Visual C ++), VB.NET (melalui Visual Basic .NET), C # (melalui Visual C #), dan F# (sebagai Visual Studio 2010). Dukungan untuk bahasa lain seperti M, Python, dan Ruby tersedia melalui layanan bahasa dan di-*instal* secara terpisah. Visual studio juga mendukung XML / XSLT, HTML / XHTML, JavaScript dan CSS.




*Gambar II.22 Microsoft visual studio IDE*

## 2.14 Eagle CadSoft 7.2.0 Professional

Eagle merupakan sebuah perangkat lunak yang berguna untuk membuat desain PCB (*Printable Circuit Board*). Eagle berisikan skema dan *layout* PCB dari rangkaian yang akan dibuat. Skema dari PCB akan digambarkan terlebih dahulu untuk kemudian diekspor menjadi sebuah *layout* PCB. Gambar Gambar II.23 menunjukkan tampilan dari perangkat lunak Eagle.




*Gambar II.23 Eagle cadsoft 7.2.0 professional*