BAB 2

LANDASAN TEORI

Pada bab ini akan dijelaskan mengenai landasan teori mengenai robot manipulator, teori-teori penunjang sistem baik perangkat keras (*hardware*), maupun perangkat lunak (s*oftware*), serta beberapa teori penunjang lainnya.

2.1 Robot

Robot merupakan salah satu bagian dari bidang *Artificial Intelligence* (AI), teknik, dan psikologi. Teknologi inilah yang menghasilkan robot. Robot diartikan sebagai mesin dengan kecerdasan komputer dan dikontrol oleh komputer, dan memiliki kemampuan fisik seperti manusia. Aplikasi dari robot ini mencakup pemberian kemampuan untuk melihat atau persepsi visual, menyentuh atau kemampuan meraba, kemampuan untuk memegang dan memanipulasi, pengangkutan atau kemampuan fisik untuk bergerak, dan navigasi atau kecerdasan untuk menemukan atau mencapai jalan keluar.

2.1.1 Sejarah Robot

Robot berasal dari kata "*robota*" yang dalam bahasa Ceko yang berarti budak, pekerja atau kuli. Pertama kali kata "*robota*" diperkenalkan oleh Karel Capek dalam sebuah pentas sandiwara pada tahun 1921 yang berjudul RUR (*Rossum's Universal Robot*). Pentas ini mengisahkan mesin yang menyerupai manusia yang dapat bekerja tanpa lelah yang kemudian memberontak dan menguasai manusia. Istilah "*robot*" ini kemudian mulai terkenal dan digunakan untuk menggantikan istilah yang dikenal saat itu yaitu *automation*. Dari berbagai litelatur robot dapat didefinisikan sebagai sebuah alat mekanik yang dapat diprogram berdasarkan informasi dari lingkungan (melalui sensor) sehingga dapat melaksanakan beberapa tugas tertentu baik secara otomatis ataupun tidak sesuai program yang di inputkan berdasarkan logika.

Robot merupakan alat mekanik yang dapat melakukan tugas fisik, baik menggunakan pengawasan dan kontrol manusia, ataupun menggunakan program yang telah didefinisikan terlebih dulu (kecerdasan buatan). Robot biasanya digunakan untuk tugas yang berat, berbahaya, pekerjaan yang berulang dan kotor. Biasanya kebanyakan robot industri digunakan dalam bidang produksi. Penggunaan robot lainnya termasuk untuk pembersihan limbah beracun, penjelajahan bawah air dan luar angkasa, pertambangan, pekerjaan "cari dan tolong" (*search and rescue*), dan untuk pencarian tambang. Belakangan ini robot mulai memasuki pasaran konsumen di bidang hiburan, dan alat pembantu rumah tangga, seperti penyedot debu, dan pemotong rumput.



Gambar 2.1 Robot Asimo

Keunggulan dalam teknologi robot tak dapat dipungkiri, telah lama dijadikan ikon kebanggaan negara-negara maju di dunia. Kecanggihan teknologi yang dimiliki, gedung-gedung tinggi yang mencakar langit, tingkat kesejahteraan rakyatnya yang tinggi, kota-kotanya yang modern, belumlah terasa lengkap tanpa popularitas kepiawaian dalam dunia robot. Pada awalnya, aplikasi robot hampir tak dapat dipisahkan dengan industri sehingga muncul istilah *industrial robot* dan robot

manipulator. Definisi yang populer ketika itu, robot industri adalah suatu robot tangan (*arm robot*) yang diciptakan untuk berbagai keperluan dalam meningkatkan produksi, memiliki bentuk lengan-lengan kaku yang terhubung secara seri dan memiliki sendi yang dapat bergerak berputar (rotasi) atau memanjang/memendek (translasi atau prismatik). Satu sisi lengan yang disebut sebagai pangkal ditanam pada bidang atau meja yang statis (tidak bergerak), sedangkan sisi yang lain yang disebut sebagai ujung (*end effector*) dapat dimuati dengan *tool* tertentu sesuai dengan tugas robot. Dalam dunia mekanikal, manipulator ini memiliki dua bagian, yaitu tangan atau lengan (*arm*) dan pergelangan (*wrist*). Pada pergelangan ini dapat di-*install* berbagai *tool*. Begitu diminatinya penggunaan manipulator dalam industri, menyebabkan banyak perusahaan besar di dunia menjadikan robot industri sebagai unggulan. Bahkan beberapa perusahaan di Jepang masih menjadikan manipulator sebagai produk utamanya, seperti Fanuc Inc. yang memiliki pabrik utamanya di lereng gunung Fuji. Untuk dapat diklasifikasikan sebagai robot, mesin harus memiliki dua macam kemampuan yaitu:

1. Bisa mendapatkan informasi dari sekelilingnya.
2. Bisa melakukan sesuatu secara fisik seperti bergerak atau memanipulasi objek.

Untuk dapat dikatakan sebagai robot sebuah sistem tidak perlu untuk meniru semua tingkah laku manusia, namun suatu sistem tersebut dapat mengadopsi satu atau dua saja sistem yang ada pada diri manusia saja sudah dapat dikatakan sebagai robot. Sistem yang diadopsi berupa sistem penglihatan (mata), sistem pendengaran (telinga) ataupun sistem gerak.

2.1.2 Kegunaan Robot

Secara umum kegunaan robot adalah untuk menggantikan kerja manusia yang membutuhkan ketelitian yang tinggi atau mempunyai resiko yang sangat besar atau bahkan mengancam keselamatan manusia. Sebagai contoh, sesesorang yang bekerja di bagian *welding* di sebuah industri *assembling* kendaraan, akan mempunyai resiko kecelakaan kerja yang cukup tinggi. Maka untuk mengurangi

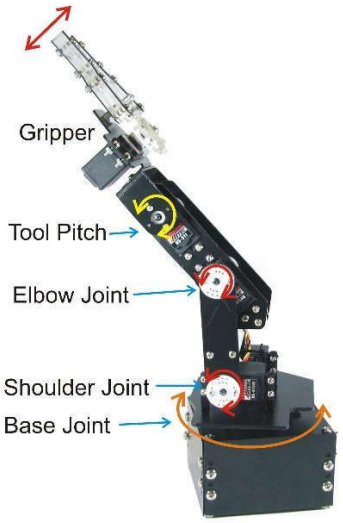
resiko kerja tersebut perlu digunakan robot yang menggantikan kerja manusia di bidang tersebut, sehingga resiko kecelakaan kerja dapat dikurangi bahkan dihilangkan. Ada juga sebagian robot yang sengaja diciptakan untuk menemani manusia di dalam aktifitasnya. Robot-robot ini dapat disebut robot bermain. Robot ini diciptakna unukt membantu manusia yang mengalami kesepian diri sehingga dpaat mempunyai teman. Robot-robot yang termasuk jienis ini termasuk antara lain *Battle Bots*, robot contesti, robot anjing. Secara garis besar robot dapat diklasifikasikan menjadi beberapa jenis, antara lain:

1. Robot indrustri
2. Robot antariksa
3. Robot transportasi
4. Robot perang
5. Robot kendali jarak jauh
6. Robot kedokteran
7. Robot riset
8. Robot bermain, dll

2.2 Manipulator

Manipulator adalah bagian mekanik yang dapat difungsikan untuk memindahkan, mengangkat benda kerja. Manipulator merupakan salah satu komponen penting dalam menunjang aktifitas suatu sistem robotika. Manipulator (sistem mekanik) merupakan "tulang" dari sebuah sistem robotika tersebut. Tanpa sistem mekanik maka suatu sistem tidak mampu menjalankan aktifitasnya. Suatu sistem mekanik didesain sesuai dengan kebutuhan dan efisiensinya. Dikarenakan sebuah sistem robot mempunyai bermacam–macam bentuk dan ukuran, sehingga memiliki beragam kemampuan gerakan. Secara klasik konfigurasi manipulator dapat dibagi dalam empat kelompok, yaitu *polar*, slindris, *cartesian*, dan sendi lengan (*joint – arm*). *Cartesian* dan slindris memiliki tingkat ketelitian yang tinggi akan tetapi ruang gerak terbatas oleh jangkauan lengannya.

Manipulator merupakan salah satu dari empat komponen penunjang sistem robotika yang tidak dapat dipisahkan dari sistem penunjang lainnya. Dikarenakan sebuah sistem robot mempunyai bermacam–macam bentuk, ukuran, dan fungsi yang berbeda maka ada bermacam – macam konfigurasi manipulator yang disesuaikan dengan fungsi dan kebutuhannya sehingga kerja dari manipulator dapat efektif dan efisien. Beberapa konfigurasi manipulator diantaranya : konfigurasi polar, slindris, *cartesian*, dan sendi lengan (*joint – arm*). *Cartesian* dan slindris memiliki tingkat ketelitian yang tinggi akan tetapi ruang gerak terbatas oleh jangkauan lengannya.



Gambar 2.2 Contoh Robot Manipulator

2.2.1 Degrees Of Freedom (DOF)

Dalam sistem manipulator juga dikenal istilah *Degrees Of Freedom* (DOF) yaitu setiap titik sumbu gerakan mekanik pada robot, tidak terhitung untuk *End Effector*, dan juga *Degrees Of Movement* (DOM) adalah kebebasan / kemampuan untuk melakukan sebuah gerakan. Sebagai contoh, robot dengan 6 derajat kebebasan :

1. *Base Rotation* (dudukan untuk berputar).
2. *Shoulder Flex* (lengan atas / pundak).

3. *Elbow Flex* (lengan bawah).
4. *Wrist Pitch* (pergelangan angguk).
5. *Wrist Yaw* (pergelangan sisi).
6. *Wrist Roll* (pergelangan putar).

Tentunya di dalam sistem manipulator juga ada sambungan yang menghubungkan antar komponen satu dengan komponen lainnya. Sambungan tersebut disebut *joint* dan *Link. Joint* memungkinkan terjadinya gerakan pada dua bagian tubuh robot, sedangkan *link* menghubungkan tiap-tiap joint. Tipe–tipe *joint* yaitu:

1. *Linear Joint.*
   Gerakan antara *In* dan *Out*, *link* adalah gerakan *linear* (tipe *L–Joint*).
2. *Orthogonal Joint.*
   Ini juga *Linear Joint*. Tetapi antara *In* dan *Out*, *Link*-nya saling tegak lurus (tipe *O-Joint*).
3. *Rotational Joint.*
   Merupakan penghubung dimana perputaran terjadi tegak lurus terhadap *In* dan *Out Link* (tipe *R-Joint*).
4. *Twisting Joint.*
   Mengakibatkan gerakan berputar, tapi putaran pararel dengan *In* dan *Out Link* (tipe *T-Joint*).
5. *Revolving Joint.*
   Input Link, pararel dengan axis perputaran dari *joint*. *Output* tegak lurus dengan putaran.

Sebagai tambahan konfigurasi fisik dari robot dan kemampuan gerak dasar, ada beberapa sepesifikasi teknik yang lain dimana menjelaskan tentang efesiensi dan efektivitas dalam unjuk kerja pada robot. Beberapa spesifik teknik adalah sebagai berikut :

– ***Work volume*** **(**area kerja**).**

Arti kata *work volume* (area kerja) mengacu pada dimana robot itu dapat bekerja. Secara teknis dapat dikatakan adalah dimana ujung bagian masih digerakkan di bawah *control*.

Work volume diperhitungkan dari :

- Konfigurasi fisik.
- Ukuran.
- Jangkauan lengan.
- Hubungan / *joint Manipulator.*

Fungsi mengetahui *Work volume* :

- *Lay out.*
- Waktu produksi.
- Area kerja dan *safety.*
- Program.

– ***Precision Of Movement*** **(** keakuratan gerak**).**

– Ada tiga jenis kategori pada keakuratan gerakan dari ujung robot pada suatu penerapan , yaitu :

- *Spatial Resolution.*

  Dapat diartikan sebagai gerakan terkecil yang masih dapat dikontrol oleh si pemrogram, sehingga *spatial resolution* adalah jumlah dari resolusi kontrol dengan ketidak akuratan mekanik.
- *Accuracy* (akurasi).

  Adalah kemampuan dari ujung robot untuk mencapai titik yang dituju. Dengan kata lain akurasi adalah setengah resolusi *spatial*.
- *Repeatability* (pengulangan).

  Adalah kemampuan dari ujung robot untuk mencapai titik yang sebelumnya dikontrol. *Repeatability* umumnya lebih kecil dari akurasi.

– ***Speed Of Movement*** **(**kecepatan gerak**).**

– ***Weight Carrying Capacity*** **(**daya angkat beban**).**

Adalah kemampuan robot untuk memindahkan beban. Faktor yang menjadikan p e r t i m b a n g a n , yaitu :

- Jenis tugas.
- Jenis barang.
- Produktivitas.

– ***Type Of Drive System*** **(**jenis pengerak**).**

Ada tiga jenis dasar penggerak robot, yaitu :

- *Hydraulic.*

  Menggunakan fluida / oli, kurang dalam segi kebersihan, beresiko kebakaran.

- *Pneumatic.*

  Menggunakan tekaanan udara merupakan jenis yang termurah, terpraktis dan *fixed points*.

- *Electric.*

  Yang dimaksud adalah motor listrik. Ada dua jenis motor, yaitu motor DC dan motor stepper. Ciri khasnya adalah kecepatan.

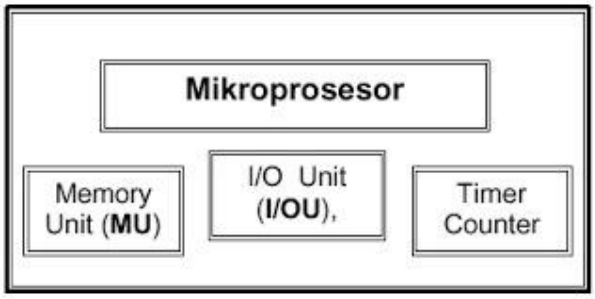
  Selain penggerak di atas, untuk mencapai presisi, kecepatan serta gerakan yang diinginkan, robot selalu dilengkapi dengan *gear* dan *cam*.

  Selain itu hal terakhir dalam sistem manipulator adalah ***end of effectors***. *End of effectors* memiliki tujuan untuk melaksanakan tugas tertentu. Faktor-faktor yang penting dalam *end effector* adalah sebagai berikut :

  - Tugas.
  - *Design.*
  - Kontrol program.
  - Ukuran area kerja.
  - Waktu siklus.
  - Keselamatan kerja.

2.3 Mikrokontroler

Mikrokontroler adalah sebuah IC (*Integrated Circuits*) yang didalamnya terdapat bagian-bagian yang ada pada sistem komputer. Mikrokontroler populer pertama yang dibuat oleh Intel pada tahun 1976 adalah mikrokontroler 8-bit Intel 8748. Mikrokontroler tersebut adalah bagian dari keluarga mikrokontroler MCS-48. Sebelumnya, *Texas Instruments* telah memasarkan mikrokontroler 4-bit pertama, yaitu TMS 1000 pada tahun 1974. TMS 1000 yang mulai dibuat sejak 1971 adalah mikrokomputer dalam sebuah cip, lengkap dengan RAM dan ROM**[6]**. Mikrokontroler sering disebut mikrokomputer cip tunggal. Mikrokontroler mempunyai satu atau beberapa tugas yang sangat spesifik berbeda dengan PC yang memiliki baragam fungsi.



Gambar 2.3 Blok Diagram Mikrokontroler

Mikrokontroler memiliki fitur-fitur berikut:

1. CPU mulai dari *prosessor* 4bit sampai 64bit.
2. *Input* / *Output* (I/O) antar muka jaringan seperti *port serial* dan *port pararel*.
3. Antar muka komunikasi serial yang lain seperti I^2C, *serial peripheral Interface* dan *Controller Area Network*.
4. Periferal, seperti pewaktu / *timer* dan *watchdog*.
5. RAM untuk penyimpanan data.
6. ROM, EPROM, EEPROM untuk penyimpanan data sementara.
7. Pengubah analog ke digital. **[6]**

Mikrokonktroler digunakan dalam produk dan alat yang dikendalikan secara automatis, seperti sistem kontrol mesin, *remote control*, mesin kantor, peralatan rumah tangga, alat berat, dan mainan. Dengan mengurangi ukuran, biaya, dan konsumsi tenaga dibandingkan desain menggunakan mikroprosesor memori dan alat *input / output* yang terpisah, kehadiran mikrokontroler membuat kontrol elektrik untuk berbagai proses menjadi lebih ekonomis. Dengan penggunaan mikrokontroler ini maka:

1. Sistem elektronik akan menjadi lebih ringkas,
2. Rancang bangun sistem elektronik dapat dilakukan lebih cepat karena sebagian besar sistem merupakan perangkat lunak yang mudah dimodifikasi,
3. Gangguan yang terjadi lebih mudah ditelusuri karena sistemnya yang kompak.

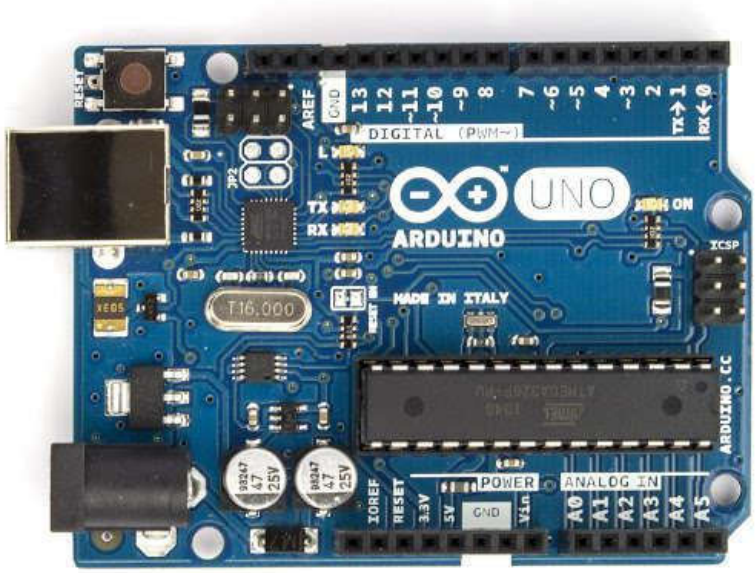
Namun, mikrokontroler tidak sepenuhnya dapat mereduksi komponen IC TTL dan CMOS yang seringkali masih diperlukan untuk aplikasi kecepatan tinggi atau sekedar menambah jumlah saluran masukan dan keluaran (I/O). Dengan kata lain, mikrokontroler adalah versi mini atau mikro dari sebuah komputer karena mikrokontroler telah mengandung beberapa periferal yang langsung bisa dimanfaatkan, misalnya port *paralel*, *port serial*, komparator, konversi digital ke analog (DAC), konversi analog ke digital dan sebagainya hanya menggunakan sistem minimum yang sederhana.

2.3.1 Mikrokontroler ATmega328

Arduino Uno adalah papan sirkuit berbasis mikrokontroler ATmega328. IC (integrated circuit) ini memiliki 14 input/output digital (6 output untuk PWM), 6 analog input, resonator kristal keramik 16 MHz, Koneksi USB, soket adaptor, pin header ICSP, dan tombol reset. Hal inilah yang dibutuhkan untuk mensupport mikrokontrol secara mudah terhubung dengan kabel power USB atau kabel *power supply* adaptor AC ke DC atau juga *battery*.

Uno berbeda dari semua board mikrokontrol diawal-awal yang tidak menggunakan chip khusus driver FTDI USB-to-serial. Sebagai penggantinya penerapan USB-to-serial adalah ATmega16U2 versi R2 (versi sebelumnya ATmega8U2). Versi Arduino Uno Rev.2 dilengkapi resistor ke 8U2 ke garis ground yang lebih mudah diberikan ke mode DFU.

Bahasa "UNO" berasal dari bahasa Italia yang artinya SATU, ditandai dengan peluncuran pertama Arduino 1.0, Uno pada versi 1.0 sebagai referensi untuk Arduino yang selanjutnya, seri Uno versi terbaru dilengkapi USB.



Gambar 2.4 Arduino Uno R3 ATmega328

Tabel 2.1 Spesifikasi Arduino Uno

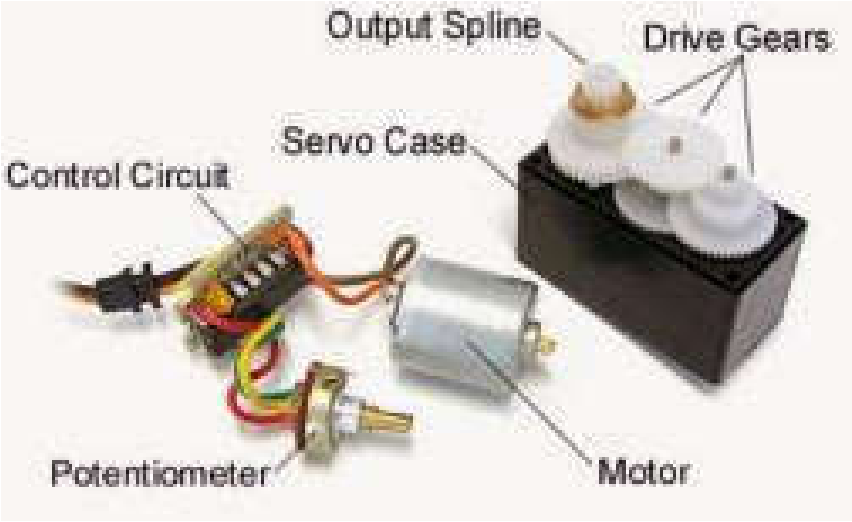
Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
[redacted]	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6

DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

2.4 Motor Servo

Motor servo adalah jenis aktuator elektromekanis yang tidak berputar secara kontinyu seperti motor DC/AC atau motor stepper, motor servo digunakan untuk posisi dan memegang beberapa objek. Mereka digunakan dimana rotasi kontinyu tidak diperlukan sehingga mereka tidak digunakan untuk megendalikan roda (kecuali servo ini dimodifikasi). Sebaliknya mereka dgunakan dimana sesuatu yang dibutuhkan untuk pindah keposisi tertentu dan kemudian berhenti dan betahan disana. Yang paling umum digunakan adalah untuk posisi kemudi pesawat terbang, kapal dan lain-lain. Servo dapat digunakan secara efektif di sini karena kemudian tidak perlu bergerak 360 derajat penuh atau mereka tidak memerlukan rotasi terus menerus seperti roda. Servo dapat diperintahkan untuk memutar dengan sudut seperti roda. Servo dapat diperintah untuk memutar dengan sudut tertentu (katakanlah 30 derajat) dan selanjutnya kemudi tetap bertahan di sana. Servo juga menggunakan mekanisme umpan balik, sehingga dapat merasakan kesalahan dalam posisinya dan mengoreksinya. Ini disebut *servomechanism*. Jadi, jika aliran udara memberikan tekanan pada kemudi dan terjadi defleksi/penyimpangan maka servo akan menerapkan kekuatan dalam arah yang berlawan dan mencoba untuk memperbaiki kesalahan. Katakanlah jika kita meminta servo untuk berada dan mengunci dirinya untuk posisi 30 derajat dan kemudian mencoba untuk memutar dengan tangan kita, maka servo akan berusaha keras sebaik mungkin untuk mengatasi gaya dan menjaga servo terkunci dalam sudut yang ditentukan.

Servo juga digunakan untuk mengontrol kemudi mobil RC, lengan dan kaki robot. Servo memiliki banyak jenis tetapi disini kita akan berkonsentrasi pada servo hobi kecil. Serbo hobi memiliki motor dan mekanisme kontrol yang dibangun dalam satu unit. Mereka memiliki 3 konektor kawat. Salah satunya adalah untuk suplai positif, dan lainnya untuk ground dan yang terakhir untuk sinyal kontrol.[3]



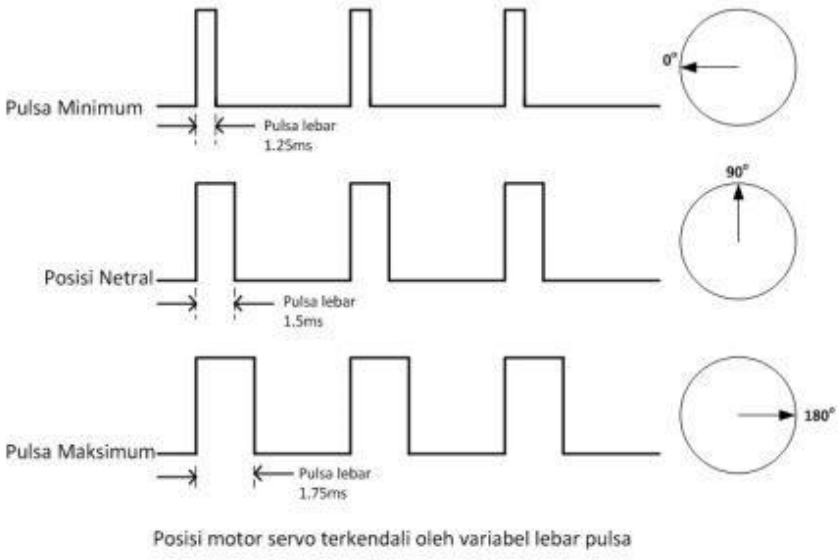
Gambar 2.5 Bentuk Servo

Ada dua jenis motor servo, yaitu motor servo AC dan DC. Motor servo AC lebih dapat menangani arus yang tinggi atau beban berat, sehingga sering diaplikasikan pada mesin-mesin industri. Sedangkan motor servo DC biasanya lebih cocok untuk digunakan pada aplikasi-aplikasi yang lebih kecil. Dan bila dibedakan menurut rotasinya, umumnya terdapat dua jenis motor servo yang dan terdapat di pasaran, yaitu motor servo rotation 180 dan servo rotation continuous.

- Motor servo standar (servo rotasi 180) adalah jenis yang paling umum dari motor servo, dimana putaran poros outputnya terbatas hanya 90 kearah kanan dan 90 kearah kiri. Dengan kata lain total putarannya hanya setengah lingkaran atau 180 .
- Motor servo *rotation continuous* merupakan jenis motor servo yang sebenarnya sama dengan jenis servo standard, hanya saja perputaran

porosnya tanpa batasan atau dengan kata lain dapat berputar terus, baik ke arah kanan maupun kiri.

Motor servo dikendalikan dengan mengirimkan pulsa melalui kabel control dengan variable lebar pulsa terkirim atau biasa disebut "*Pulse Width Modulation (PWM)*". Ada minimum lebar pulsa dan maksimum lebar pulsa dan tingkat perulangan. Sebuah motor servo biasanya hanya dapat mengubah 90° di kedua arah untuk total 180° gerakan. Posisi netral motor didefinisikan sebagai posisi di mana servo memiliki jumlah yang sama dari potensi rotasi di kedua searah jarum jam atau berlawanan arah jarum jam arah. PWM yang dikirim ke motor menentukan posisi poros, dan berdasarkan durasi dari pulsa yang dikirim melalui kabel kontrol rotor akan berubah ke posisi yang diinginkan.



Gambar 2.6 Alur pulsa pada servo

Motor servo mengharapkan untuk mendapat pulsa setiap 20 milidetik (ms) dan panjang pulsa akan menentukan seberapa jauh motor berubah. Sebagai contoh, pulsa 1.5ms akan membuat pergantian motor ke posisi 90 °. Lebih pendek dari 1.25ms bergerak ke 0 ° dan lebih lama/panjang dari 12ms akan memutar servo sejauh 180 °.

2.5 Kecerdasan Buatan

Kecerdasan buatan sering disebut *Artificial Intelligence* (AI), kecerdasan *artificial*, *intelijensia artificial*, atau *intelijensia* buatan. Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudung pandang mereka masing-masing. Ada yang fokus pada logika berfikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI yang diperoleh dari beberapa *textbook* berbeda kedalam empat kategori [7], yaitu :

1. *Thinking Humanly*

   Pendekatan ini dilakukan dengan dua cara sebagai berikut :

   a. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berfikir.
   b. Melalui eksperimen-eksperimen psikologi.

2. *Acting Humanly*

   Pada tahun 1950, Alan Turing merancang suatu kajian bagi komputer berintelijensia untuk menguji apakah komputer tersebut mampu mengelabuhi seorang manusia yang mengintrogasinya melalui telepaty (komunikasi berbasis teks jarak jauh). Jika interrogator tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari *Turing test*. Komputer tersebut perlu memiliki kemampuan *Natural Language Processing*, *Knowledge Representation, Automated Reasoning*, *Machine Learning*, *Computer Vision*, dan *Robotic*. Turing Test sengaja menghindari interaksi fisik antara interrogator dan komputer karena simulasi fisik manusia tidak memerlukan *intelijensia*.

3. *Thinking Rationally* Terdapat dua masalah dalam pendekatan ini, yaitu :

   a. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam formal term yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.

b. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. *Acting Rationally*

Membuat Inferensi yang logis merupakan bagian dari suatu *rational agent*. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka agent dapat melakukan aksi berdasarkan kesimpulan tersebut.

2.6 Pengolahan Citra Digital

Pengolahan citra digital artinya melakukan pemrosesan gambar berdimensi dua melalui komputer digital. Menurut Efford (2000), pengolahan citra adalah istilah umum untuk berbagai teknik yang keberadaanya untuk memanipulasi dan memodifikasi citra dengan berbagai cara [8]. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital dapat diolah menggunakan perangkat lunak tertentu. Misalnya citra yang gelap dapat diolah supaya menjadi lebih terang, merubah format warna citra menjadi *grayscale*, *resize* citra dan lain sebagainya.

2.6.1 Grayscale

Citra *grayscale* menangani gradasi warna hitam dan putih, yang tentu saja menghasilkan efek warna abu-abu. Warna gambar dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar antara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih [8]. Berikut adalah rumus konversi citra berwana (RGB) menjadi *grayscale* :

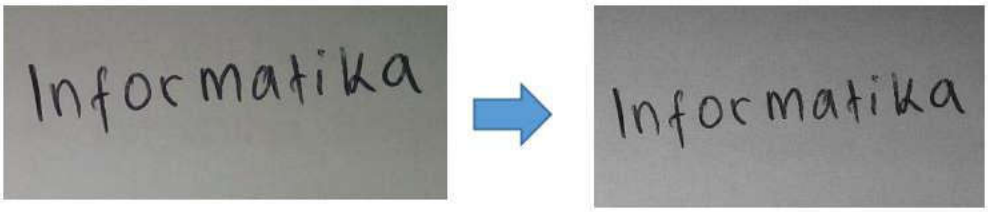
I = (0.2989 R) + (0.5870 G) + (0.1141 B) (2. 1) Keterangan :

R = komponen nilai merah (*Red*) dari suatu titik *pixel*

G = komponen nilai hijau (*Green*) dari suatu titik *pixel*

B = komponen nilai biru (*Blue*) dari suatu titik *pixel*

Persamaan di atas merupakan salah satu rumus yang digunakan untuk mengkonversi citra berwarna menjadi *grayscale*. Persamaan 2.1 dipilih dalam penelitian ini karena mata manusia secara alami lebih sensitif terhadap cahaya merah dan hijau. Maka dari itu, warna-warna ini diberi bobot yang lebih tinggi untuk memastikan bahwa keseimbangan intensitas relatif dalam citra *grayscale* yang dihasilkan mirip dengan citra warna RGB [9].



Gambar 2.7 Contoh Hasil *Grayscale*

2.6.2 Sauvola Threshold

Sauvola threshold merupakan metode *threshold* yang mampu memberikan peningkatan dengan cara menghitung ambang menggunakan rentang nilai dinamis dari nilai standar deviasi citra *grayscale* [10]. *Sauvola* termasuk dalam *local threshold* dimana nilai ambang ditentukan oleh nilai *pixel* tetangga.

$$T(x,y) = m(x,y) * (1 + k * (\frac{s(x,y)}{R} - 1)) \quad (2.2)$$

$$m(x,y) = \frac{\sum_{i=min}^{i=max} \sum_{j=min}^{j=max} img(i,j)}{i*j} \quad (2.3)$$

$$s(x,y) = \sqrt{\frac{\sum_{i=min}^{i=max} \sum_{j=min}^{j=max} (img(i,j) - m(x,y))^2}{(i*j)-1}} \quad (2.4)$$

Keterangan :

R : Nilai maksimum dari standar deviasi (128 untuk citra *grayscale*) k : *Kernel* dengan nilai antara 0.2 – 0.5.

m : Fungsi yang menghasilkan nilai rata-rata dari sejumlah *pixel* citra.

s : Fungsi yang menghasilkan nilai standar deviasi dari sejumlah *pixel* citra T : Fungsi yang menghasilkan nilai *threshold* (ambang)

x : Nilai koordinat lebar citra

y : Nilai koordinat tinggi citra

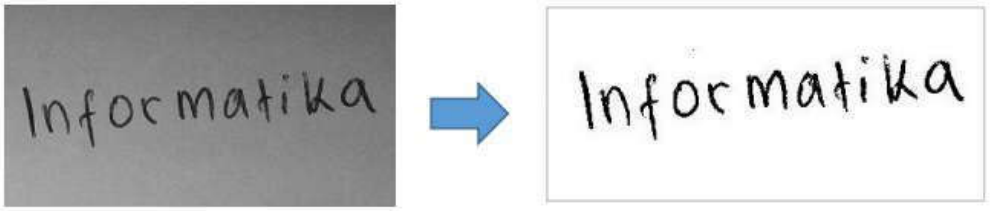
Setelah nilai ambang T(x,y) sudah didapatkan, selanjutnya masukkan ke persamaan di bawah ini.

$$f(x,y) = \begin{cases} 0, & \text{img}(x,y) < \text{T}(x,y) \\ 255, & \text{img}(x,y) \geq \text{T}(x,y) \end{cases} \quad (2.5)$$

Keterangan :

f : Fungsi yang menghasilkan nilai 0 atau 255

img : Nilai *grayscale* citra



Gambar 2.8 Contoh hasil *Sauvola Threshold*

2.6.3 Operasi Morfologi

Kata morphologi (*morphology*) secara sederhana dapat diartikan sebagai bentuk dan struktur suatu objek atau dalam deskripsi lainnya disebutkan bahwa morphologi adalah susunan dan hubungan antar bagian pada suatu objek. Morphologi di dunia digital dapat diartikan sebuah cara untuk mendeskripsikan ataupun menganalisa bentuk dari objek digital [11]. Pada operasi *Morphologi* menggunakan dua input himpunan yaitu suatu citra dan suatu kernel atau yang disebut *Structuring Elements* (SE). Dalam penelitian ini menggunakan operasi morfologi dilasi, erosi dan *closing*.

2.6.3.1 Operasi Dilasi

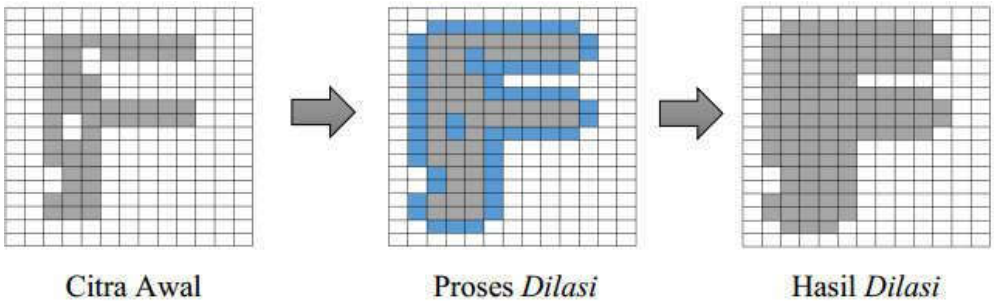
Operasi *dilasi* digunakan untuk memperbaiki citra yang pecah. Bila suatu objek (citra input) dinyatakan dengan A dan SE dinyatakan dengan B serta Bx menyatakan translasi B sedemikian sehingga pusat B terletak pada x. Operasi *dilasi* A dengan B dapat dinyatakan sebagai berikut [11] .

$$(\quad ,) = \qquad = \{ : \qquad \neq \} \tag{2.6}$$

Dengan menyatakan himpunan kosong.

Keterangan :
D : Fungsi yang menghasilkan citra *dilasi*
A : Citra input
B : *Structuring Elements* (SE)

Proses *dilasi* dilakukan dengan membandingkan setiap *pixel* citra input dengan nilai pusat SE dengan cara (*superimpose*) SE dengan citra sehingga pusat SE tepat dengan posisi *pixel* citra yang di proses. Jika paling sedikit ada 1 pixel pada SE sama dengan nilai *pixel* objek (*foreground*) citra maka *pixel* input diset nilainya dengan nilai *pixel foreground* dan bila semua *pixel* yang berhubungan adalah *background* maka maka input *pixel* diberi nilai *background*. Proses serupa dilanjutkan dengan menggerakkan (translasi) SE *pixel* demi *pixel* pada citra input. Berikut adalah contoh matriks citra *dilasi* dengan menggunakan *structure element* 3x3.



Gambar 2.9 Contoh Dilasi

2.6.3.2 Operasi Erosi

Proses *erosi* merupakan kebalikan dari proses *dilasi*. Jika dalam proses *dilasi* menghasilkan objek yang lebih luas maka dalam proses *erosi* akan menghasilkan objek yang menyempit (mengecil). Operasi *erosi* dapat dinyatakan sebagai berikut [11].

$$(,) = \qquad = \{ : \subset \} \tag{2.7}$$

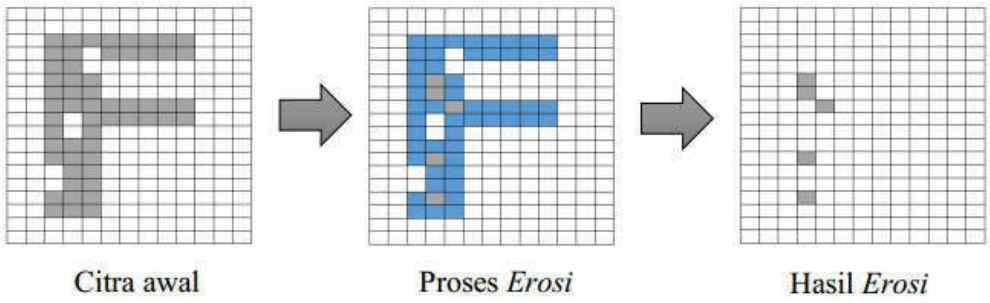
Keterangan :

A : Citra input

B : *Structuring Elements* (SE)

E : Fungsi yang menghasilkan citra *erosi*

Sama Seperti *dilasi*, proses *erosi* dilakukan dengan membandingkan setiap *pixel* citra input dengan nilai pusat SE dengan cara melapiskan SE dengan citra sehingga pusat SE tepat dengan posisi *pixel* citra yang diproses. Jika semua *pixel* pada SE tepat sama dengan semua nilai *pixel* objek (*foreground*) citra maka *pixel* input diset nilainya dengan nilai *pixel foreground*, bila tidak maka input *pixel* diberi nilai *pixel background*. Proses serupa dilanjutkan dengan menggerakkan SE *pixel* demi *pixel* pada citra *input*. Berikut adalah contoh *erosi* dengan menggunakan *structure element* 3x3.



Gambar 2.10 Contoh Erosi

2.6.3.3 Operasi *Closing*

Operasi *closing* merupakan penggabungan antara operasi *erosi* dan *dilasi*. Hanya saja operasi *dilasi* dilakukan terlebih dahulu kemudian baru diikuti dengan operasi *erosi*. Operasi *closing* dapat dinyatakan sebagai berikut [11].

$$C(\ ,\ ) = \bullet\ = E(D(A,-B),-B) \qquad (2.\ 8)$$

Keterangan :

C : Fungsi yang menghasilkan citra *closing*

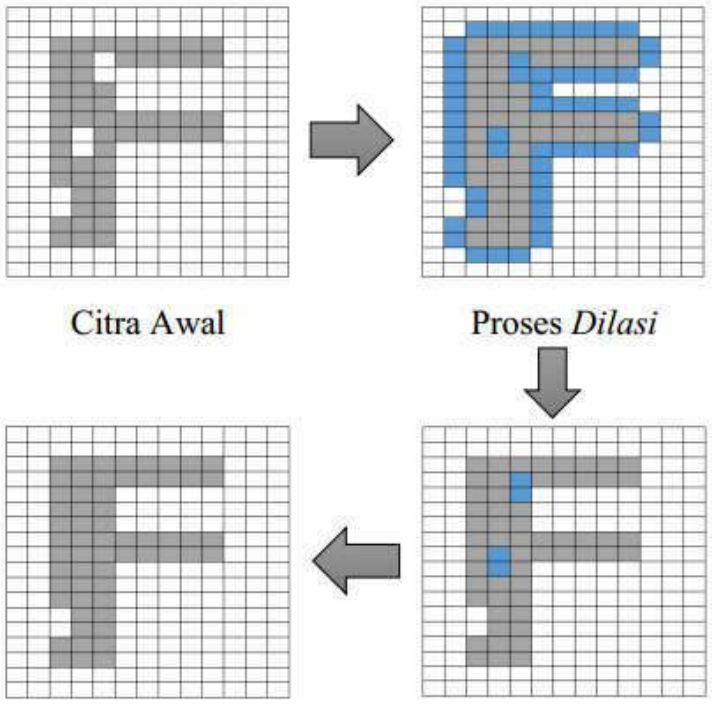
A : Citra input

B : *Structuring Elements* (SE)

E : Hasil *erosi*

D : Hasil *dilasi*

Hasil operasi *closing* hampir mirip seperti hasil operasi dilasi yakni memperbesar batas luar dari objek *foreground* dan juga menutup lubang kecil yang terletak di tengah objek, namun hasil operasi *closing* tidak sebesar hasil *dilasi*. Hasil dilasi akan menyebabkan pembengkakan bentuk keseluruhan objek. Efek ini dapat dikurangi dengan menerapkan proses *erosi* setelah proses *dilasi* tersebut.

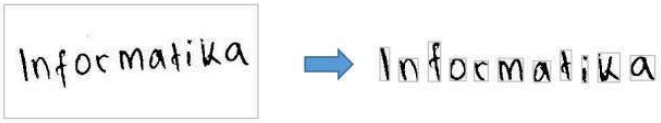


Gambar 2.11 Contoh *Closing*

2.6.4 Segmentasi

Segmentasi merupakan teknik untuk membagi suatu citra menjadi beberapa daerah (*region*) dimana setiap daerah memiliki kemiripan atribut [11]. Sampai sejauh mana pembagian tersebut dilakukan tergantung pada masalah yang dihadapi. Idealnya, langkah segmentasi tersebut dihentikan pada saat objek yang diinginkan

sudah berhasil dipisahkan. Dalam penelitian ini segmentasi dilakukan dengan memotong baris dan kolom citra. Pemotongan pertama dilakukan untuk semua baris, kemudian dilanjutkan dengan pemotongan kolom.



Gambar 2.12 Contoh Segmentasi

2.6.5 *Resize* Citra

Resize citra artinya adalah mengubah besarnya ukuran citra digital dalam *pixel*. Adakala ukurannya berubah menjadi lebih kecil dari file aslinya dan ada kalanya sebaliknya. Proses *Resize* citra dapat mempengaruhi *size* citra lebih besar atau lebih kecil dan juga kualitas citra menjadi lebih baik atau lebih buruk.



Gambar 2.13 Contoh *Resize*

Contoh *resize* di atas mengubah citra yang awalnya memiliki ukuran 35x32 *pixel* menjadi 20x40 *pixel*.

2.6.6 Binerisasi

Citra biner adalah citra dengan setiap *pixel* hanya dinyatakan dengan sebuah nilai dari dua kemungkinan yaitu nilai 0 dan 1. Citra yang dikonversi menjadi biner biasanya dari citra dengan warna hitam putih [8]. Citra jenis ini banyak dipakai dalam pengolahan citra misalnya digunakan untuk memperoleh tepi dalam suatu objek.

$$f(x) = \begin{cases} 0, & x = 0 \\ 1, & x = 255 \end{cases} \qquad (2.9)$$

Keterangan :

f : fungsi yang menghasikan angka

biner x : Nilai suatu *pixel* citra

2.7 Teknik Learning (Belajar)

Teknik *learning* merupakan salah satu bagian dari ilmu kecerdasan buatan. Dalam learning kita tidak harus tahu aturan yang berlaku dalam sistem yang dibuat, melainkan aturan yang diharapkan bisa secara otomatis ditemukan. Proses belajar dalam teknik learning menggunakan data-data masukan sebagai pengalaman yang baru untuk dapat meningkatkan performasi dari sistem. Program komputer yang sanggup belajar adalah program yang bisa meningkatkan performasinya melalui pengalaman [7]. Ada beberapa metode yang menggunakan teknik *learning* seperti *decision tree*, jaringan syaraf tiruan, algoritma genetika dan lain-lain.

2.8 JST (Jaringan Saraf Tiruan)

JST (Jaringan Syaraf Tiruan) adalah prosesor tersebar parallel (*parallel distributed processor*) yang sangat besar yang memiliki kecendrungan untuk menyimpan pengetahuan yang bersifat pengalaman dan membuatnya siap untuk digunakan [18]. JST menyerupai otak manusia dalam dua hal, yaitu :

1. Pengetahuan diperoleh jaringan melalui proses belajar
2. Kekuatan hubungan antar sel syaraf (*neuron*) yang dikenal sebagai bobotbobot sinaptik digunakan untuk menyimpan pengetahuan.

JST merupakan salah satu upaya manusia untuk memodelkan cara kerja atau fungsi sistem syaraf manusia dalam melaksanakan tugas tertentu. Pemodelan ini didasari oleh kemampuan otak manusia dalam mengorganisasikan sel-sel penyusunnya yang disebut *neuron*, sehingga mampu melaksanankan tugas-tugas tertentu, khususnya pengenalan pola dengan efektivitas yang sangat tinggi.

2.8.1 Model Sel Syaraf (*Neuron*)

Sel syaraf (*neuron*) adalah unit pemrosesan informasi yang merupakan dasar dari operasi JST. Terdapat tiga elemen dasar dari model *neuron* [7], yaitu :

1. Sekumpulan sinapsis atau jalur hubungan, dimana masing-masing sinapsis memiliki bobot atau kekuatan hubungan.

2. Suatu *adder* (instruksi) untuk menjumlahkan sinyal-sinyal input yang diberi bobot oleh sinapsis *neuron* yang sesuai. Operasi-operasi yang digambarkan di sini mengikuti aturan *linear combiner*.
3. Suatu fungsi aktivasi untuk membatasi amplitudo *output* dari setiap *neuron*.

2.8.2 Arsitektur Jaringan

Arsitektur jaringan merupakan pola dimana *neuron-neuron* pada JST disusun berhubungan erat dengan algoritma belajar yang digunakan untuk melatih jaringan. Secara umum arsitektur jaringan dibagai menjadi empat [7], yaitu :

1. *Single-Layer Feedforward Networks*
   Suatu JST berlapis adalah jaringan *neuron* yang diorganisaikan dalam bentuk lapisan-lapisan. Pada bentuk jaringan berlapis yang paling sederhana, hanya terdapat *input layer* dengan *node* sumber yang terproyeksi ke dalam *output layer* dari *neuron*, tetapi tidak sebaliknya. Jaringan ini adalah jaringan jenis *feedforward* yang tepat.
2. *Multi-Layer Feedforward Networks*
   Kelas ke dua dari *feedforward neural network* adalah jaringan dengan satu atau lebih lapis tersembunyi (*hidden layer*), dengan c*omputation nodes* yang berhubungan disebut *hidden neurons* atau *hidden units.*
3. *Recurrent Networks*
   Recurrent Neural Network adalah jaringan yang mempunyai minimal satu *feedback loop.* Suatu *recurrent network* bisa terdiri dari satu lapisan *neuron* tunggal dengan masing-masing *neuron* memberikan kembali outputnya sebagai input pada semua *neuron* yang lain.
4. *Lattice Structure*
   Sebuah *lattice* terdiri dari satu dimensi, dua dimensi, atau lebih array *neuron* dengan himpunan *node* sumber yang bersesuaian dan memberikan sinyal input ke array. Dimensi *lattice* pada jumlah dimensi ruang dimana *graph* berada.

Dalam penelitian ini, arsitektur jaringan yang digunakan adalah *Single Layer Feed forward Networks* dimana nilai masukan yang masuk pada jaringan langsung diolah tanpa melalui hidden layer dan kemudian setelah itu masuk ke lapisan output untuk mendapatkan hasilnya.

2.8.3 Proses Belajar JST

Belajar dalam konteks JST adalah proses dimana parameter-parameter bebas JST diadaptasikan melalui suatu proses perangsangan berkelanjutan oleh lingkungan dimana jaringan berada. Proses belajar dalam JST ada dua [7], yaitu :

1. *Supervised Learning* (Belajar Dengan Pengawasan)

   Supervised atau *active learning* adalah proses belajar membutuhkan guru. Yang dimaksud guru disini adalah sesuatu yang memiliki pengetahuan tentang lingkungan. Guru bisa direpresentasikan sebagai sekumpulan sampel input-output. Pembangunan pengetahuan dilakukan oleh guru dengan memberikan respon yang diinginkan kepada JST. Parameterparameter jaringan berubah-ubah berdasarkan vektor latih dan sinyal kesalahan (sinyal kesalahan adalah perbedaan antara keluaran JST dan respon yang diinginkan). Proses perubahan ini dilakukan secara berulangulang, selangkah demi selangkah, dengan tujuan agar JST bisa memiliki kemampuan yang mirip dengan gurunya. Dengan kata lain, JST dilatih untuk dapat memetakan sekumpulan sampel *input-output* dengan akurasi yang tinggi. Pada proses belajar ini, output yang diharapkan telah diketahui sebelumnya.

2. *Unsupervised Learning* (Belajar Tanpa Pengawasan)

   Unsupervised atau *self-organized learning* tidak membutuhkan guru untuk memantau proses belajar. Dengan kata lain, tidak ada sekumpulan sampel *input-output* atau fungsi tertentu untuk dipelajari oleh jaringan. Salah satu contoh *unsupervised learning* adalah *competitive learning*. Pada metode ini, tidak ditentukan hasil yang seperti apakah yang diharapkan selama proses pembelajaran.

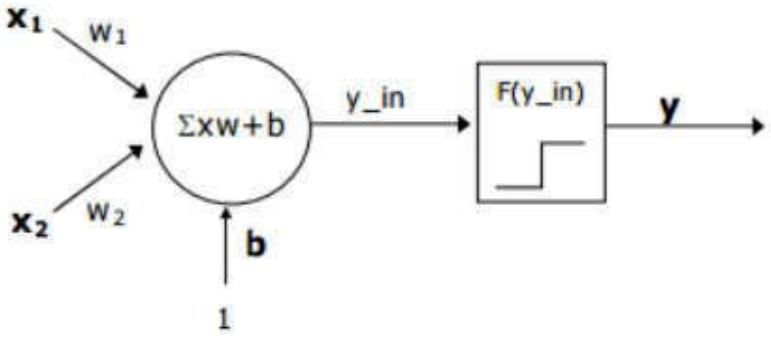
Proses belajar yang digunakan dalam penelitian ini yaitu *supervised learning* (pembelajaran terawasi). *Supervised learning* digunakan karena target kelas yang didapatkan sudah ditentukan sebelumnya.

2.8.4 Pembelajaran Dengan Pengawasan (*Supervised Learning*)

Ada banyak metode pembelajaran yang termasuk dalam *supervised learning* dan sudah pernah diimplementasikan dalam beberapa kasus. Namun dalam penelitian ini, metode pembelajaran yang digunakan yaitu *Learning Vector Quantization* (LVQ). Di bawah ini dijelaskan beberapa metode pembelajaran yang termasuk ke dalam *supervised learning*.

1. *Hebb Rule*

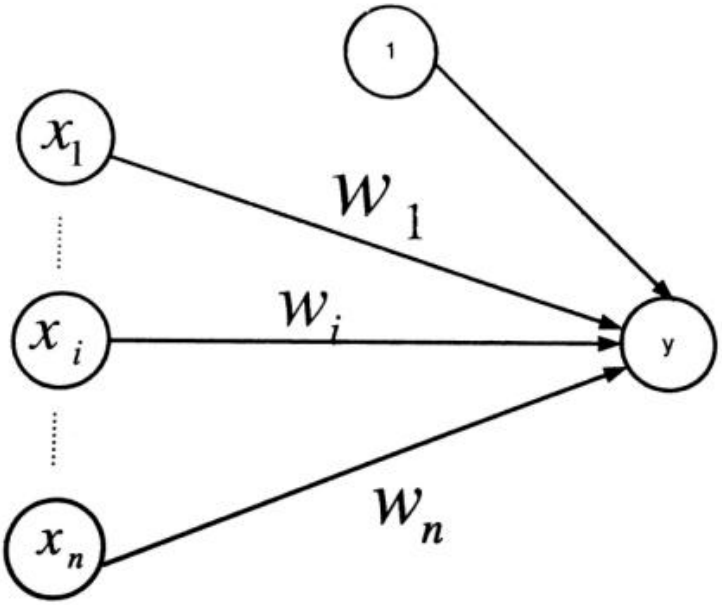
   Dalam metode *hebb rule* proses pembelajaran terjadi dengan cara memodifikasi dari bobot sinapsis atau bobot sinaptik, jika dua neuron yang saling berhubungan satu dengan yang lain dalam waktu dan kondisi yang sama, maka bobot dari kedua neuron tersebut akan dinaikkan.



Gambar 2.14 Jaringan *Hebb Rule*

2. *Perceptron*

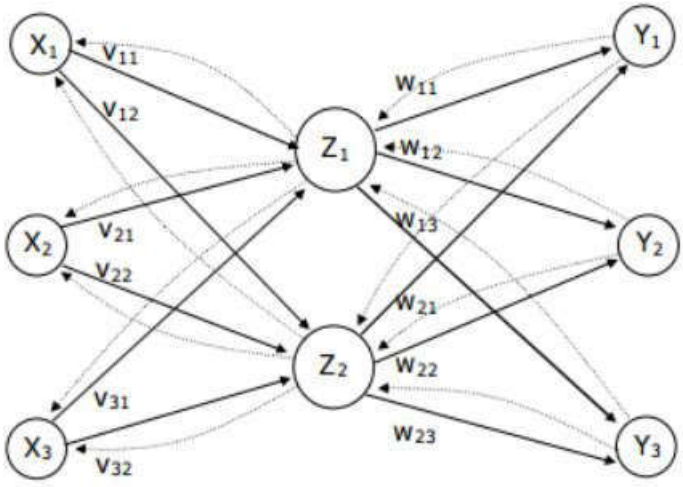
   Metode *perceptron* merupakan suatu metode pelatihan yang lebih kuat dari metode *hebb,* terutama dalam iterasi dapat membuat *output* dari bobot menjadi konvergen.



Gambar 2.15 Jaringan *Perceptron*

3. *Backpropagation*

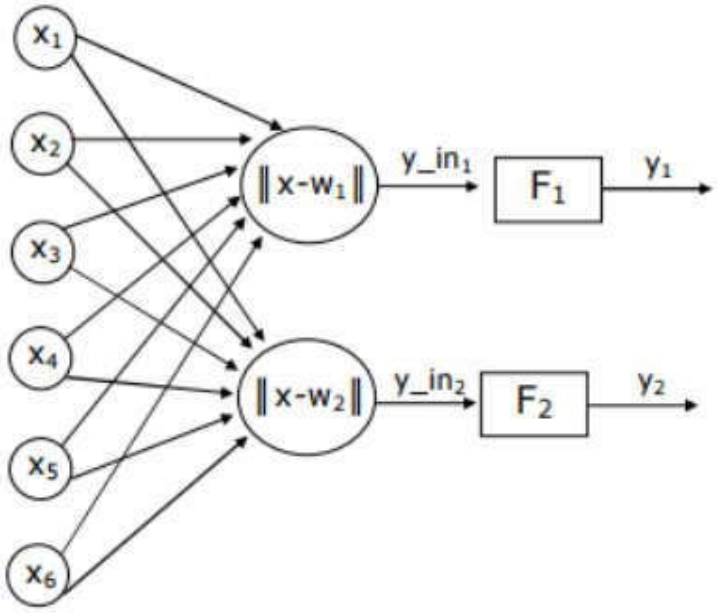
Pelatihan menggunakan metode *backpropagation* terdiri dari 3 langkah yaitu pelatihan pola input secara *feedforward*, perhitungan dan *backpropagation* dari kumpulan kesalahan dan penyesuaian bobot. Berikut adalah arsitektur dari jaringan *backpropagation* pada gambar 2.15



Gambar 2.16 Jaringan Backpropagation

4. *Learning Vector Quantization (LVQ)*

Learning Vector Quantization (LVQ) merupakan metode klasifikasi pola yang masing-masing unit output mewakili kategori atau kelas tertentu. Dalam metode ini, suatu lapisan kompetitif akan secara otomatis belajar untuk mengklasifikasikan vektor-vektor data input. Kelas-kelas yang didapatkan sebagai hasil lapisan kompetitif ini hanya tergantung pada jarak antara vektor input. Jika dua vektor mendekati sama, maka lapisan kompetitif akan meletakkan kedua vektor tersebut ke dalam kelas yang sama. Gambar 2.16 merupakan arsitektur dari metode *Learning Vector Quantization*.



Gambar 2.17 Jaringan Learning Vector Quantization

Sedangkan algoritma metode LVQ adalah sebagai berikut.

1. Inisialisasi : bobot (W) Maksimum epoch (MaxEpoch), error minimum yang diharapkan (Eps), *learning rate (α)*.
2. Masukkan : Input
   : x(m,n) Target m
   : T(1,n)
3. Tetapkan kondisi awal :

epoch = 0

err=1

4. Kerjakan jika: (epoch<MaxEpoch) atau (α > eps)
   a. epoch=epoch+1
   b. kerjakan untuk i=1 sampai n
      i. tentukan J sedemikian hingga ||x – wj|| minimum (sebut sebagai C_j)

$$. = \Sigma \quad 7 \quad (\quad 4) \qquad (2.10)$$

      ii. Perbaiki w_j dengan ketentuan jika T = C_j maka :

$$w\ (baru) = w \quad (lama) + \alpha[x - w\ (lama)] \qquad (2.11)$$

   c. jika T ≠ C_j maka :

$$w\ (baru) = w \quad (lama) - \alpha[x - w \quad (lama)] \qquad (2.12)$$

      kurangin nilai α

Berikut adalah keterangan rumus dari algoritma di atas.

T : Target kelas data latih

c : Jarak selisih bobot terkecil

x : Data latih

W : Data bobot kelas

j : Indeks suatu data

m : Banyak data latih

n : Banyak data pada lapisan input / jumlah pixel citra

2.9 UML (Unified Modeling Language)

UML (*Unified Modeling Language*) adalah bahasa grafis untuk mendokumentasi, menspesifikasikan, dan membangun sistem perangkat lunak [12]. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha bersama dari banyak pihak, didukung oleh kakas-kakas yang diintegrasikan lewat XML. Standar UML dikelola oleh OMG (*Object Management Group*). UML bukanlah :

2.11.1 Bahasa pemrograman visual, tetapi bahasa pemodelan visual.

2.11.2 Spesifikasi Kakas, tetapi spesifikasi bahasa pemodelan.

2.11.3 Proses, Tetapi yang memungkinkan proses-proses.Java Netbeans

Teknik pemodelan dalam UML ada dua jenis diagramnya [12], yaitu :

1. Diagram struktur

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek statik dari sistem. Diagram struktur UML terdiri dari : Diagram kelas (*Class diagram*), Diagram objek (*Object diagram*), Diagram Komponen (*Component diagram*), dan Diagram deployment (*Deployment diagram*).
2. Diagram perilaku

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek dinamis dari sistem. Diagram prilaku dalam UML terdiri dari : Diagram use case (*Use case diagram*), Diagram sekuen (*Sequence diagram*), Diagram kolaborasi (*Collaboration diagram*), Diagram aktivitas (*Activity Diagram*).

2.9.1 *Use Case Diagram*

Use Case diagram merupakan salah satu diagram untuk memodelkan aspek tingkah laku sistem. Masing-masing *use case diagram* menunjukkan sekumpulan *use case*, aktor dan hubungannya. *Use case diagram* penting untuk memvisualisasikan, menspesifikasikan, dan mendokumentasikan kebutuhan perilaku sistem. Diagram-diagram *use case* merupakan pusat pemodelan prilaku sistem, subsistem dan kelas [12]. *Use case* adalah interaksi antara aktor eksternal dan sistem, hasil yang dapat diamati oleh aktor, berorientasi pada tujuan, dideskripsikan di *use case diagram* dan teks. Berikut adalah simbol yang ada dalam *use case diagram*.

Tabel 2.2 Simbol Use Case Diagram

SIMBOL	NAMA SIMBOL	FUNGSI
	Aktor	Pihak yang mengakses *use case*
	*Use Case*	Mewakili apa yang sistem bisa lakukan
	Association	Merelasikan aktor dengan *use case*
	*System Boundary*	Menggambarkan batasan sistem terhadap lingkungannya

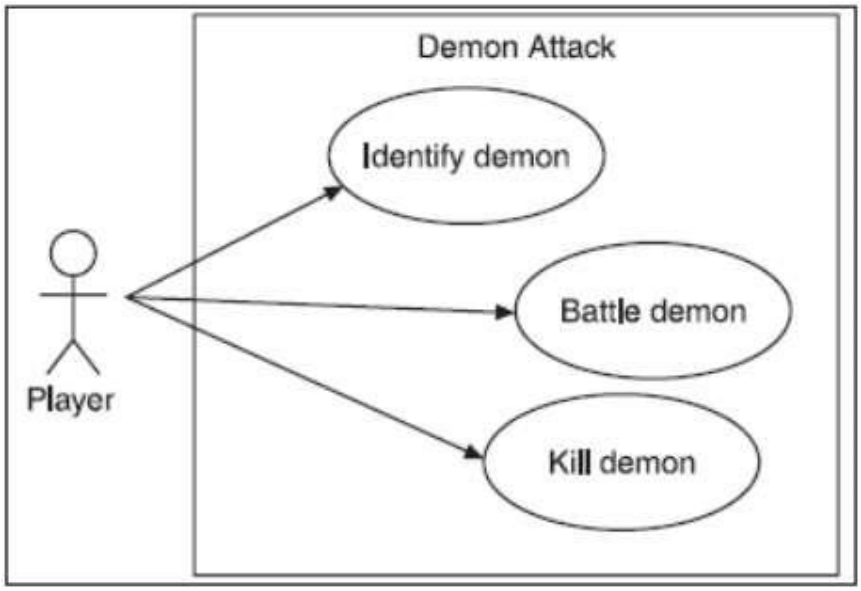
Relasi antar *use case* dengan *use case* lainnya ada tiga jenis, yaitu :

1. Relasi *Include*

   Merupakan relasi antar *use case* dimana jika *use case* A meng-*include usecase* B, maka *use case* B akan diimplementasikan setiap kali *use case* A diimplementasikan. Direpresentasikan dengan garis putus-putus bertuliskan <<include>> ke arah *use case* yang akan di-*include*.
2. Relasi *Extend*

   Merupakan relasi antar *use case* dimana jika *use case* A di extend oleh *usecase* B maka *use case* B akan bisa saja diimplementasikan atau tidak setiap kali *use case* A diimplementasikan. Direpresentasikan dengan garis putusputus bertuliskan <<extend>> ke arah *use case* yang akan di-*extend*.
3. Relasi *Generalization*

   Digunakan untuk membuat *use case* lebih spesifik.



Gambar 2.18 Contoh *Use Case Diagram*

Gambar 2.13 di atas merupakan contoh *use case diagram* yang terdiri dari:

1. *Actor* : *Player*
2. *Use Case* : *Identify daemon, Battle demon, Kill demon*
3. *System Boundary* : *Demon Attack*

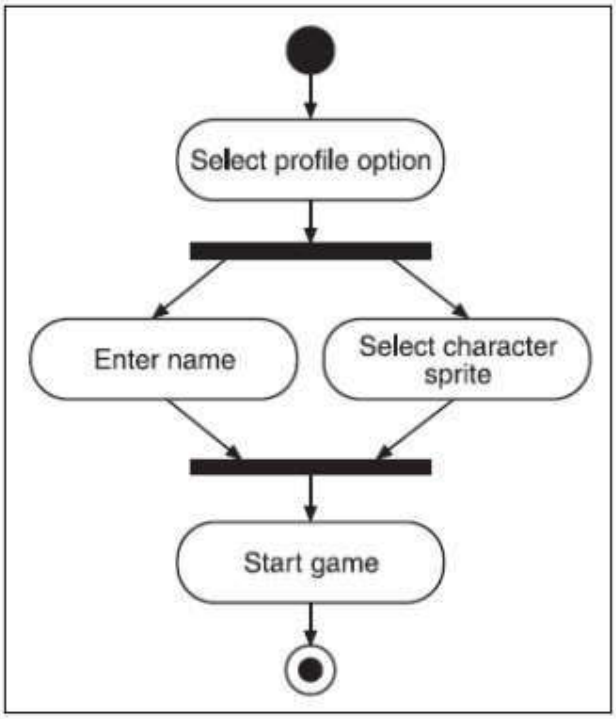
2.9.2 Activity Diagram

Activity diagram adalah diagram *flowchart* yang diperluas yang menunjukkan aliran kendali satu aktivitas ke aktivitas lain. Kita menggunakan diagram ini memodelkan aspek dinamis sistem. Aktivitas adalah eksekusi notatomik yang berlangsung di *state machine*. *Activity diagram* mendeskripsikan aksi-aksi dan hasilnya. *Activity diagram* berupa operasi-operasi dan aktivitasaktivitas di *use case*. *Activity diagram* dapat digunakan untuk :

1. Pandangan dalam yang dilakukan di operasi.
2. Pandangan dalam bagaimana objek-objek bekerja.
3. Pandangan dalam di aksi-aksi dan pengaruhnya pada objek-objek.
4. Pandangan dalam dari suatu *use case*.

5. Logik dari proses bisnis.

Diagram aktivitas merupakan jenis khusus dan diagram *statechart*. *State* adalah aksi-aksi yang menuju state berikutnya setelah selesainya aksi itu.



Gambar 2.19 Contoh *Activity Diagram*

Dari gambar 2.18 di atas merupakan contoh *activity diagram* dengan alur :

1. Lingkaran hitam penuh menandakan memulai aktivitas
2. Kemudian melakukan aktivitas *Select profile option*
3. Masuk ke proses *fork* untuk menjalankan aktivitas secara pararel kemudian menjalankan aktivitas di bawahnya
4. Masuk ke proses *join* untuk menggabungkan proses yang dipisahkan oleh *fork*
5. Melakukan aktivitas *Start game*
6. Lingkaran hitam sebagian menandakan akhir dari suatu aktivitas.

2.9.3 Class Diagram

Class diagram merupakan diagram paling umum dipakai di semua pemodelan berorientasi objek. Pemodelan kelas merupakan pemodelan yang paling utama di pendekatan berorientasi objek. Pemodelan kelas menunjukkan kelas-kelas yang ada di sistem dan hubungan antar kelas-kelas itu, atribut-atribut dan operasi-operasi di setiap kelas [12]. *Class diagram* menunjuk aspek statik sistem terutama untuk mendukung kebutuhan fungsional sistem. Meskipun *class diagram* serupa dengan model data, namun kelas-kelas tidak hanya menunjuk struktur informasi tapi juga mendeskripsikan perilaku. Salah satu maksud *class diagram* adalah untuk mendefinisikan fondasi bagi diagram-diagram lain dimana aspek-aspek lain dari sistem ditunjukkan. Elemen-elemen yang penting dalam *class diagram* adalah sebagai berikut [12].

1. Kelas Kelas merupakan elemen terpenting di sistem berorientasi objek. Kelas mendeskripsikan satu blok pembangunan sistem. Berikut adalah bagian dari kelas :
   a) Nama
      Nama kelas harus unik karena akan menjadi *identifier* di program.
   b) Atribut
      Atribut adalah properti bernama di kelas yang mendeskripsikan range nilai yang dipunyai instan kelas. Kelas dapat mempunyai sejumlah atribut atau tidak sama sekalai.
   c) Operasi
      Operasi adalah implementasi layanan yang dapat diminta pada sembarang objek kelas itu untuk mempengaruhi prilaku sistem.
   d) *Access modifier*
      Access Modifier dalam kelas digunakan untuk membatasi akses dari kelas lain yang ingin mengakses atribut atau operasi dari suatu kelas. Ada tiga *Access Modifier* yang digunakan yaitu *public* (+), *protected*(#) dan *private* (-).

2. Antarmuka (*Interface*)

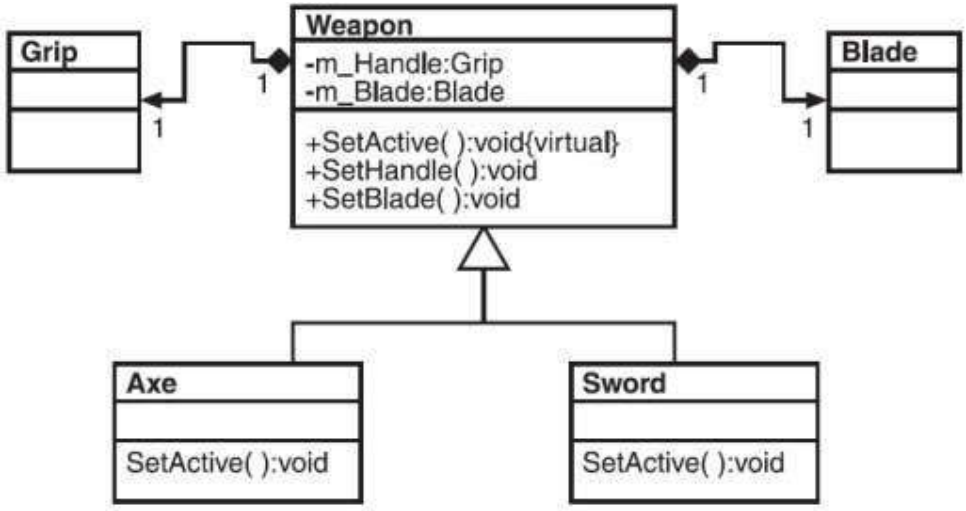
   Antarmuka (*Interface*) merupakan korelasi operasi yang menspesifisikan layanan dari suatu kelas atau komponen. Antarmuka mendeskripsikan perilaku tampak secara eksternal dari elemen.

3. Kolaborasi

   Kolaborasi merupakan pendefinisian suatu interaksi dan sekelompok peran elemen-elemen lain yang bekerja sama untuk menyediakan suatu perilaku kooperatif yang lebih besar dari penjumlahan seluruh elemen. Kolaborasi memiliki struktur, prilaku dan dimensi. Kolaborasi ini merepresentasikan implementasi pola tertentu yang membentuk sistem.

4. Hubungan (*Relationship*)

   Hubungan antar kelas di diagram kelas terdiri dari *Asosiasi*, *Generalisasi* dan *Depedency*.

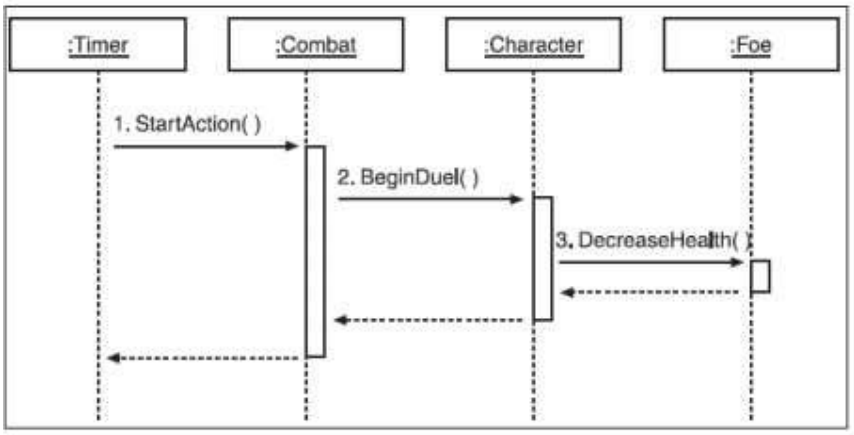


Gambar 2.20 Contoh *Class Diagram*

2.9.4 Sequence Diagram

Sequence diagram digunakan untuk memodelkan skenario penggunaan. Skenario penggunaan adalah barisan kejadian yang terjadi selama satu eksekusi sistem. Cakupan skenario dapat beragam, dari mulai semua kejadian di sistem atau

hanya kejadian pada objek-objek tertentu. *Sequence diagram* menunjukkan objek sebagai garis vertikal dan tiap kejadian sebagai panah horizontal dari objek pengirim ke objek pengirim ke objek penerima. Gambar 2.20 merupakan contoh *sequence diagram*.



Gambar 2.21 Contoh *Sequence Diagram*

Dari contoh *sequence diagram* di atas memiliki empat kelas, dimana :

1. Kelas *Timer* memanggil fungsi *StartAction()* yang ada pada kelas *Combat*
2. Kelas *Combat* memanggil fungsi *BeginDuel()* yang terdapat pada kelas *Character*.
3. Kelas *Character* memanggil fungsi *DecreaseHealth()* yang terdapat pada kelas *Foe*.
4. Dari hasil pemanggilan fungsi, kemudian nilainya direturnkan kembali sampai ke kelas *Timer*

2.10 Pengujian Sistem

Pengujian adalah proses pemeriksaan atau evaluasi sistem atau komponen sistem secara manual atau otomatis untuk memverikasi apakah sistem memenuhi kebutuhan-kebutuhan yang dispesifikan atau mengidentifikasi perbedaanperbedaan antara hasil yang diharapkan dengan hasil yang terjadi [12]. Pengujian seharusnya meliputi tiga konsep berikut.

1. Demonstrasi validitas perangkat lunak pada masing-masing tahap di siklus pengembangan sistem.
2. Penentuan validitas sistem akhir dikaitkan dengan kebutuhan pemakai.
3. Pemeriksaan perilaku sistem dengan mengeksekusi sistem pada data sampel pengujian.

Awalnya pengujian diartikan sebagai aktivitas yang dapat atau hanya dilakukan setelah pengkodean (kode program selesai). Namun, pengujian seharusnya dilakukan dalam skala lebih luas. Pengujian dapat dilakukan begitu spesifikasi kebutuhan telah dapat didefinisikan. Evaluasi terhadap spesifikasi dan perancangan juga merupakan teknik di pengujian. Kategori pengujian dapat dikategorikan menjadi dua [12], yaitu :

1. Berdasarkan ketersediaan logik sistem, terdiri dari *Black box testing* dan *White box testing*.
2. Berdasarkan arah pengujian, terdiri dari Pengujian *top down* dan Pengujian *bottom up*.

2.10.1 Pengujian Black Box

Konsep *black box* digunakan untuk merepresentasikan sistem yang cara kerja di dalamnya tidak tersedia untuk diinspeksi. Di dalam *black box*, item-item yang diuji dianggap "gelap" karena logiknya tidak diketahui, yang diketahui hanya apa yang masuk dan apa yang keluar dari *black box* [12]. Pada pengujian *black box*, kasus-kasus pengujian berdasarkan pada spesifikasi sistem. Rencana pengujian dapat dimulai sedini mungkin di proses pengembangan perangkat lunak. Teknik pengujian konvensional yang termasuk pengujian "black box" adalah sebagai berikut [12].

1. *Graph-based testing*
2. *Equivalence partitioning*
3. *Comparison testing*
4. *Orthogonal array testing*

Pada pengujian *black box*, kita mencoba beragam masukan dan memeriksa keluaran yang dihasilkan. Kita dapat mempelajari apa yang dilakukan kotak, tapi tidak mengetahui sama sekali mengenai cara konversi dilakukan. Teknik pengujian *black box* juga dapat digunakan untuk pengujian berbasis skenario, dimana isi dalam sistem mungkin tidak tersedia untuk diinspeksi tapi masukan dan keluaran yang didefinisikan dengan *use case* dan informasi analisis yang lain.

2.10.2 Pengujian Akurasi

Akurasi merupakan seberapa dekat suatu angka hasil pengukuran terhadap angka sebenarnya (*true value* atau *reference value*). Tingkat akurasi diperoleh dengan persamaan sebagai berikut.

$$GK = \frac{MO \quad SU \quad XQ}{MO \quad SQT \quad SYW} Q\ 100\% \qquad (2.15)$$

2.11 Bahasa Pemrograman

Bahasa pemrograman diperlukan untuk menjalankan instruksi-instruksi apa yang harus dilakukan komputer. Komputer tidak bisa memahami bahasa manusia, sehingga diperlukan penggunaan bahasa komputer di dalam program komputer. Penelitian ini menggunakan bahasa pemrograman Java untuk pengembangan aplikasi dari pengolahan citra, bahasa pemrograman C untuk pengembangan *hardware* dari penggunaan arduino dan bahasa pemrograman SQL untuk mengelola *database*.

2.11.1 Bahasa Pemograman Java

Bahasa pemrograman Java dikembangkan oleh sebuah tim yang diketuai oleh James Gosling di *Sun Microsystem*. Java awalnya dikenal dengan Oak, yang didesain pada tahun 1991 untuk chip-chip yang tertanam pada peralatan-peralatan elektronik. Pada tahun 1995, diberi nama baru Java yang didesain ulang untuk mengembangkan aplikasi-aplikasi internet. Java telah menjadi sangat populer. Perkembangannya yang sangat cepat dan penerimaannya di kalangan pengguna

dapat dijejak dari karakteristik perancangannya, khususnya dari janji pengembang Java bahwa begitu anda menciptakan suatu program maka anda bisa menjalankannya di mana saja [13].

Java memiliki banyak fitur, bahasa pemrograman bertujuan umum yang dapat digunakan untuk mengembangkan aplikasi-aplikasi tingkat tinggi. Saat ini Java tidak lagi hanya digunakan untuk pemrograman web, tetapi juga untuk aplikasi-aplikasi *standalone* bebas *platform* pada *server*, *desktop*, dan divais-divais bergerak (*mobile*) [13]. Bahasa Java juga telah digunakan untuk mengembangkan kode dalam berkomunikasi dan mengendalikan robot di Mars. Banyak perusahaan yang sebelumnya meremehkan keunggulan Java, namun sekarang malah menggunakannya untuk mengembangkan aplikasi-aplikasi terdistribusi yang dapat diakses oleh banyak konsumen melalui internet. Java merupakan bahasa pemrograman yang tangguh terbukti handal pada banyak aplikasi. Terdapat tiga edisi Java [13], yaitu :

1. Java SE (*Standard Edition*) Digunakan untuk mengembangkan aplikasi-aplikasi pada sisi client atau *applet*.
2. Java EE (*Enterprise Edition*) Digunakan untuk mengembangkan aplikasi-aplikasi pada sisi *server*, seperti *Java servlets* dan *Java server pages*.
3. Java ME (*Micro Edition*) Digunakan untuk mengembangkan aplikasi-aplikasi untuk divais bergerak, seperti telepon genggam.

2.11.2 Bahasa Pemrograman C

Bahasa pemrograman C merupakan salah satu bahasa pemrograman komputer. Dibuat pada tahun 1972 oleh Dennis Ritchie untuk Sistem Operasi Unix di *Bell Telephone Laboratories*. Meskipun C dibuat untuk memprogram sistem dan jaringan komputer namun bahasa ini juga sering digunakan dalam mengembangkan software aplikasi. C juga banyak dipakai oleh berbagai jenis *platform* sistem operasi dan arsitektur komputer, bahkan terdapat beberepa compiler yang sangat populer telah tersedia. C secara luar biasa memengaruhi bahasa populer lainnya, terutama C++ yang merupakan extensi dari C.

Standar bahasa C yang asli adalah standar dari UNIX. Sistem operasi, kompiler C dan seluruh program aplikasi UNIX yang esensial ditulis dalam bahasa C. Kepopuleran bahasa C membuat versi-versi dari bahasa ini banyak dibuat untuk komputer mikro. Untuk membuat versi-versi tersebut menjadi standar, ANSI (*American National Standard Institutes*) membentuk suatu komite (ANSI Committee X3J11) pada tahun 1983 yang kemudian menetapkan standar ANSI untuk bahasa C. Standar ANSI ini didasarkan kepada standar UNIX yang diperluas.

Bahasa C mempunyai kemampuan lebih dibanding dengan bahasa pemrograman yang lain. Bahasa C merupakan bahasa pemrograman yang bersifat portabel, yaitu suatu program yang dibuat dengan bahasa C pada suatu komputer akan dapat dijalankan pada komputer lain dengan sedikit (atau tanpa) ada perubahan yang berarti. Bahasa C merupakan bahasa yang biasa digunakan untuk keperluan pemrograman sistem, antara lain untuk membuat:

1. Assembler
2. Interpreter
3. Compiler
4. Sistem Operasi
5. Program bantu (utility)
6. Editor
7. Paket program aplikasi

Compiler Bahasa C dapat menghasilkan *file hex* yang bisa di uploadkan ke dalam memori *flash* mikrokontroler. Ukuran *file hex* yang dihasilkan oleh *compiler* merupakan salah satu perhatian utama pemrogram mikrokontroler karena mikrokontroler mempunyai memori *on-chip flash* yang terbatas. Sedangkan bahasa *Assembly* menghasilkan *file hex* yang lebih kecil dibandingkan bahasa C, namun demikian pemrograman dalam bahasa *Assembly* sering melelahkan / membosankan dan memakan waktu. Di sisi lain, pemrogram dalam bahasa C lebih mudah menuliskannya, tetapi *file hex* yang dihasilkan lebih besar daripada bahasa

Assembly. Berikut beberapa alasan utama menggunakan bahasa C dibandingkan bahasa *Assembly*:[6]

- Lebih mudah dan sedikit memakan waktu dalam penulisan bahasa C dibandingkan bahasa *Assembly*.
- Bahasa C lebih mudah dimodifikasi dan diperbarui (*update*)
- Kita dapat menggunakan kode yang tersedia pada *function library*
- Kode C bersifat *portable* terhadap mikrokontroler lainnya dengan hanya melakukan sedikit atau tanpa modifikasi

2.11.3 SQL

SQL (*Structured Query Language*) merupakan sebuah bahasa permintaan *database* yang tersetruktur yang dibuat sebagai bahasa yang dapat merelasikan beberapa tabel dalam *database* maupun merelasikan antar *database* [14]. Bahasa SQL ditulis langsung dalam sebuah program *database* sehingga pengguna dapat melihat langsung permintaan yang diinginkan dan sekaligus melihat hasilnya. Ada beberapa program *database server* yang menggunakan standar *query* berupa SQL yaitu : Oracle, PostgreSQL, MySQL, MsQL, SQL *Server*, dan lain-lain. Dalam penelitian ini SQL diimplementasikan pada *database server* MySQL.

2.12 Aplikasi

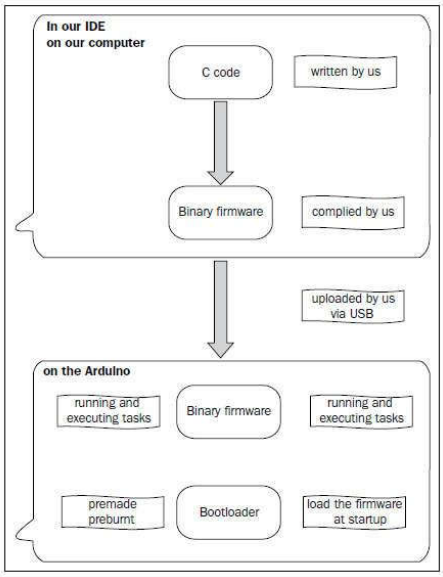
Aplikasi merupakan perangkat lunak siap pakai yang dapat digunakan untuk menjalankan perintah-perintah dari para pengguna aplikasi tersebut. Dalam penelitian ini digunakan aplikasi untuk mendukung penelitian yang dilakukan meliputi Ardunio IDE, Xampp dan NetBeans.

2.12.1 Arduino IDE

Arduino adalah pengendali mikro *single-board* yang bersifat *open-source*, dirancang untuk memudahkan penggunaan elektronik dalam berbagai bidang. *Hardware*-nya memiliki prosesor Atmel AVR dan *software*-nya memiliki bahasa pemrograman sendiri. Arduino tidak lagi membutuhkan perangkat keras terpisah

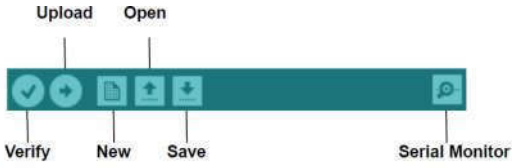
(disebut *programmer* atau *downloader*) untuk memuat atau meng-upload kode baru ke dalam mikrokontroler. Cukup dengan menggunakan kabel USB untuk mulai menggunakan Arduino. Selain itu, Arduino IDE menggunakan bahasa pemrograman **C++** dengan versi yang telah disederhanakan, sehingga lebih mudah dalam belajar pemrograman.

Dibandingkan dengan IDE yang lain seperti *Eclipse*, *Xcode* atau *Microsoft Visual Studio*, Arduino IDE lebih praktis. Didalam Arduino IDE terdapat *editor*, *compiler*, *loader* dan *serial monitor*. Tidak ada fitur yang tambahan seperti *Debugger* atau *Code Completion*.



Gambar 2.22 Diagram alur proses dari arduino IDE[4]

Arduino IDE menyediakan antar muka yang memudahkan pengguna untuk menulis kode, mengeksekusi kode dan mengirimkan kode yang kita buat ke dalam perangkat keras.



Gambar 2.23 Toolbar dalam Arduino IDE

Terdapat 6 ikon dari kiri kekanan yang harus kita ketahui pada Arduino IDE karena tombol tersebut sering digunakan[15]:

- *Verify* – berfungsi untuk mengecek / memverifikasi error pada koding.
- *Upload* – berfungsi mengirimkan koding / perintah yang sudah kita buat ke dalam perangkat arduino
- *New* – Membuat file *sketch* baru
- *Open* – Membuka file *sketch* yang berisi koding yang sebelumnya sudah pernah kita buat / simpan
- *Save* – Menyimpan file *sketch* yang sudah dibuat
- *Serial Monitor* – Menampilkan *serial monitor*

Selain menu toolbar yang sudah dijelaskan tadi ada menu lain yang harus diketahui:

- *Auto Format* : memberikan format kode yang benar dengan standar indentasi
- *Archive Sketch* : Kompresi *sketch* yang dipilih dengan semua file
- *Board* : Menampilkan data semua *board* arduino yang sudah mendukung dalam IDE
- *Serial Port* : Menampilkan data *port serial* yang tersedia dalam sistem
- *Programmer* : Menampilkan semua alat *programmer* yang mendukung

2.12.2 Xampp

Xampp merupakan *software web server apache* yang di dalamnya sudah tersedia *database server mysql* dan *support php programming*. *Xampp* juga mudah digunakan, gratis dan mendukung instalasi Linux dan Windows. Salah satu keuntungan *xampp* yaitu instalasi yang dilakukan satu kali dan sudah tersedia *Apache Web Server*, *MySql Database Server*, *PHP Support, FileZilla*, dan modul lainnya. Versi Xampp yang digunakna dalam penelitian ini adalah versi 5.6.19-0. Kemudian fitur yang digunakan adalah fitur MySQL dan *Apache Web Server*.

2.12.3 NetBeans

Netbeans merupakan sebuah aplikasi *Integrated Development Environment* (IDE) yang berbasiskan Java dari Sun Microsystems yang berjalan di atas *swing*.

Swing merupakan sebuah teknologi Java untuk mengembangkan aplikasi desktop yang dapat berjalan pada berbagai macam platform seperti windows, linux, Mac OS x dan Solaris. Netbeans merupakan *software development* yang *Open Source*, dengan kata lain *software* ini di bawah pengembangan bersama. NetBeans merupakan sebuah proyek kode terbuka yang sukses dengan pengguna yang sangat luas, komunitas yang terus tumbuh, dan memiliki hampir 100 mitra. Sun Microsystems mendirikan proyek kode terbuka NetBeans pada bulan Juni 2000 dan terus menjadi sponsor utama. Dalam penelitian ini menggunakan NetBeans dengan versi 8.0.1.

2.12.4 *Processing*

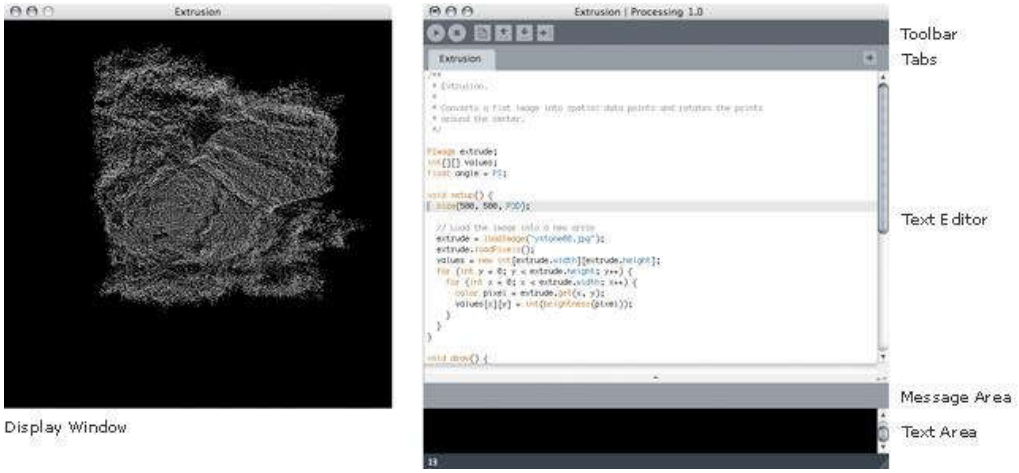
Processing adalah bahasa pemrograman dan lingkungan pemrograman (*development environment*) *open source* untuk memprogram gambar, animasi dan interaksi. Digunakan oleh pelajar, seniman, desainer, peneliti, dan *hobbyist* untuk belajar, membuat prototipe, dan produksi. *Processing* digunakan untuk mengajarkan dasar-dasar pemrograman komputer dalam konteks rupa dan berfungsi sebagai buku sketsa perangkat lunak (*software*) dan *tool* produksi profesional. *Processing* bebas untuk didownload dan tersedia untuk GNU / Linux, Mac OS X, dan Windows.

Processing adalah suatu projek terbuka yang diinisiasi oleh Ben Fry dan Casey Reas. Berkembang dari ide-ide yang dieksplorasi di *Aesthetics and Computation Group* (ACG) di MIT Media Lab. Projek ini kini terus diperbaiki dan dikelola oleh sejumlah kecil tim voluntir.

Processing mengaitkan konsep *software* pada prinsip-prinsip bentuk rupa, gerak, dan interaksi. *Processing* mengintegrasikan suatu bahasa pemrograman, lingkungan pemrograman, dan metodologi pengajaran ke dalam sistem terpadu.

Lingkungan Pemrograman *Processing* terdiri dari teks editor terintegrasi dan jendela tampilan untuk menampilkan program. Jika tombol “run” ditekan, program akan mengkompilasi dan berjalan di jendela grafis (jendela tampilan). Dari

jendela lingkugan utama, bisa menjalankan, menghentikan, menyimpan, membuka, dan mengeksport file.



Gambar 2.24 Tampilan utama dari Processing

Dalam *Processing* struktur program dapat dibuat dalam tiga tingkat kompleksitas: Mode Statik, Mode Aktif, dan Mode *Java*. Mode Statik digunakan untuk membuat gambar statik. Contoh berikut menggambar sebuah segi empat kuning di layar.

```
size(200, 200);
background(255);
noStroke(); fill(255,
204, 0); rect(30, 20,
50, 50);
```

Mode Aktif menyediakan bagian *setup*() opsional yang akan berjalan ketika program mulai berjalan. Bagian *draw*() akan berjalan selamanya sampai progam dihentikan. Contoh ini menggambar segi empat yang mengikuti posisi *mouse* (disimpan dalam variabel *mouseX* dan *mouseY*). Perhatikan bahwa panggilan ke *method background*() terletak di *setup*() karena hanya diperlukan sekali.

```
void setup(){ size(200,
      200);
      rectMode(CENTER);
      noStroke();
      fill(255, 204, 0);
}
```

```
void draw(){
     background(255);
     rect(width-mouseX, height-mouseY, 50,
     50); rect(mouseX, mouseY, 50, 50);
}
```

Mode *Java* adalah yang paling fleksibel, namun tidak tersedia sampai dengan rilis *Processing* 1.0 *Beta*. Mode ini memungkinkan menulis program *Java* secara lengkap di dalam Lingkugan *Processing*. Contoh ini sama dengan di atas, namun ditulis dalam *style Java*:

```
public class MyDemo extends BApplet
     { void setup(){
          size(200, 200);
          background(255);
          rectMode(CENTER);
          noStroke();
          fill(255, 204, 0);
     }
     void draw(){
          rect(width-mouseX, height-mouseY, 50,
          50); rect(mouseX, mouseY, 50, 50);
     }
}
```