BAB II

LANDASAN TEORI

2.1 Optical Character Recognition

Optical Character Recognition (OCR) merupakan teknologi pengenalan karakter menggunakan komputer. Meskipun demikian, definisi OCR yang beredar cukup bervariasi. Pada ensiklopedia *online* PCMAG, tertulis bahwa yang dimaksud dengan OCR adalah pengenalan mesin atas karakter hasil cetakan. Teknologi untuk mengenali tulisan tangan bukanlah OCR, melainkan *Intelligent Character Recognition* (ICR). Meskipun demikian, menurut The Free Dictionary, OCR dapat juga mengenali tulisan tangan, sedangkan ICR dapat mengenali tulisan tangan walaupun gaya menulisnya berbeda-beda.

Belum dapat dipastikan siapa yang mengembangkan OCR untuk pertama kalinya. Namun teknologi ini sudah sejak lama ada. Seperti yang ditulis oleh O'Brien dan Haddej (2012), Paul W. Handel mematenkan OCR pada 1931. Alat ini bekerja dengan menembakkan cahaya kepada karakter cetak melalui filter yang bentuknya dibuat menyerupai angka. Apabila cahaya yang menembus filter tersebut bentuknya cocok dengan karakter yang diujikan, maka mesin akan menerima karakter tersebut.

Beberapa bidang yang umum mengaplikasikan OCR diantaranya pemasukan data dari formulir, pengenalan plat mobil, penyortiran paket berdasarkan kode pos, pengkonversian gambar yang didapat dengan memindai buku menjadi teks, pembacaan naskah-naskah tua, pengenalan terhadap penulis dari suatu kalimat (*handwritten biometric recognition*), pengidentifikasian bahasa, dan menurut

Mori dan Malik (2003), untuk pengujian CAPTCHA (*Completely Automated Public Turing test to Tell Computers and Humans Apart*).

2.2 Online Handwriting Character Recognition

Online handwriting character recognition merupakan teknologi pengenalan tulisan tangan seseorang dengan melihat proses menulisnya, bukan hasil tulisannya. Dengan demikian, *online handwriting character recognition* dilakukan pada saat pengguna menulis, berbeda dengan OCR yang dilakukan setelah pengguna selesai menulis (Kumar, 2011). Banyak terdapat salah kaprah terhadap istilah *online*, yang sama sekali tidak berkaitan dengan masalah jaringan. *Online* di sini dimaksudkan untuk membedakannya dengan OCR yang telah dibahas sebelumnya, yang kadang disebut juga *offline character recognition*. *Online handwriting character recognition* disebut juga dengan istilah *dynamic character recognition*, *real-time character recognition*, *intelligent character recognition*, atau cukup *online character recognition*.

Online handwriting character recognition memiliki keunggulan dibandingkan OCR karena *online handwriting character recognition* memungkinkan komputer untuk memperoleh informasi temporal. Oksuz (2003), dan Delaye dan Anquetil (2013) menjelaskan beberapa contoh informasi temporal yang dimaksud, seperti urutan dan jumlah goresan, arah penulisan, kecepatan, sudut kemiringan alat tulis, dan tekanan pada media tulis. Sebaliknya, OCR hanya mampu untuk menganalisis informasi spasial yang terdapat pada naskah yang diamati. Informasi spasial yang dimaksud misalnya jarak antar garis, bentuk garis, posisi garis, dan orientasi garis. Menurut Marukatat dan Artieres (2004), informasi spasial dapat dibagi menjadi

dua bagian besar, yaitu informasi absolut, didapat dari posisi goresan terhadap bidang tulis, dan informasi relatif, didapat dari posisi antar goresan. Karena *online handwriting character recognition* juga dapat memperoleh informasi spasial selain informasi temporal, maka kedua informasi tersebut dapat digabungkan untuk mencapai hasil yang lebih akurat.

Hasil pengenalan dari *online handwriting character recognition* juga dapat langsung ditampilkan sesegera mungkin setelah pengguna menuliskan sebuah huruf (Oksuz, 2003). Dengan demikian, setiap kesalahan yang muncul dapat langsung dikoreksi, dimana tindakan koreksi ini dapat dipelajari oleh program sehingga ke depannya dapat mengenali karakter dengan lebih baik lagi.

2.3 Writer-Dependent Character Recognition

Writer-dependent character recognition merupakan teknik pengenalan karakter dimana aplikasi akan menyesuaikan dengan dan belajar dari tulisan tangan penggunanya (Kumar, 2011). Karakter yang ditulis pengguna akan dijadikan masukan untuk pengenalan karakter-karakter selanjutnya. Karena gaya penulisan manusia cenderung tidak sama antara satu dengan lainnya, aplikasi *writer-dependent character recognition* bersifat personal dan belum tentu cocok saat hendak digunakan oleh orang lain. Teknik ini bertolak belakang dengan *writer-independent character recognition*, dimana aplikasi berusaha untuk mengenali tulisan siapapun secara umum, dan tidak secara spesifik belajar dari tulisan seorang pengguna saja.

Menurut LavioLa dan Zeleznik (2007), sebelum aplikasi *writer-dependent character recognition* dapat digunakan, pengguna harus terlebih dulu melatih

aplikasi agar dapat mengenali tulisan pengguna yang bersangkutan, dengan cara menuliskan karakter-karakter contoh untuk dipelajari oleh aplikasi. Setelah itu, aplikasi dapat mengenali tulisan tangan pengguna tersebut. Apabila aplikasi ini hendak digunakan juga oleh pengguna lain, maka pengguna yang baru harus melatih aplikasi ini lagi, karena gaya penulisan pengguna kedua tidak sama dengan gaya penulisan pengguna pertama.

Keunggulan dari *writer-independent character recognition*, menurut LavioLa dan Zeleznik (2007), adalah pengguna dapat langsung menggunakan aplikasi yang bersangkutan, tidak perlu melakukan penyesuaian dulu. Namun, mereka juga menjelaskan keunggulan *writer-dependent character recognition*, diantaranya adalah karakter yang dikenali dapat ditambah atau dimodifikasi, dapat menyesuaikan dengan gaya penulisan pengguna, memiliki akurasi yang lebih tinggi, dan dapat dijadikan basis dalam pengembangan *writer-independent character recognition* maupun *handwritten biometric recognition*. *Writer-dependent character recognition* lebih cocok untuk orang yang gaya menulisnya berbeda dari kebanyakan orang lain, karena orang yang demikian akan sulit dikenali tulisan tangannya apabila metode *writer-independent character recognition* yang digunakan.

2.4 Feature Extraction

Feature extraction merupakan metode yang dipakai dalam bidang *digital image processing* untuk menganalisis suatu gambar digital berdasarkan fitur-fitur yang dimilikinya. Fitur yang dimaksud dapat berupa tingkat kecerahan gambar, warna gambar, ukuran gambar, maupun fitur lain. Menurut Albu (2010), fitur

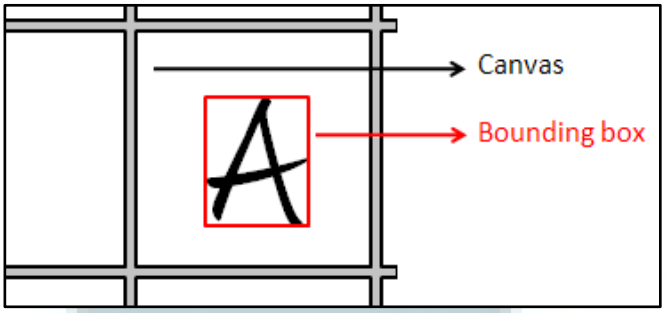
yang dimiliki sebuah gambar dapat diklasifikasikan ke dalam dua kelompok, yakni fitur global dan fitur lokal. Fitur global merupakan fitur keseluruhan yang dimiliki dari sebuah gambar, misalnya rata-rata tingkat kecerahan dan ukuran gambar secara keseluruhan. Fitur lokal merupakan fitur yang hanya dimiliki oleh sebagian kecil dari gambar tersebut, misalnya panjang dari sebuah goresan, sudut yang dibentuk antara dua garis, dan fitur lainnya.

Dalam bidang *character recognition*, setiap karakter dianggap memiliki fitur yang berbeda-beda. Dengan menggunakan metode *feature extraction*, program dapat mengenali sebuah karakter berdasarkan fitur-fitur yang dimiliki karakter tersebut, juga dengan membandingkan fitur yang ada dengan fitur yang dimiliki karakter lainnya. Di antara semua kemungkinan karakter yang ada, maka karakter yang memiliki kesamaan fitur terbanyak dianggap sebagai karakter yang dimaksud untuk ditulis oleh pengguna aplikasi (Singh, 2010).

Berikut ini merupakan uraian beberapa fitur yang dapat digunakan dalam metode *feature extraction*, dikumpulkan dari *paper* yang ditulis oleh Trier dkk. (1995), Heutte, dkk. (1997), Blumenstein, dkk. (2004), LaViola dan Zeleznik (2007), serta Pansare dan Bhatia (2012).

1. Wilayah ternormalisasi

   Wilayah ternormalisasi merupakan rasio dari luas *bounding box* sebuah karakter terhadap *canvas. Canvas* merupakan sebuah bujur sangkar dimana sebuah karakter dapat ditulis (untuk selanjutnya, istilah *canvas* dapat dipertukarkan dengan ‘bidang tulis’), sedangkan *bounding box* merupakan persegi panjang terkecil yang sisinya sejajar dengan *canvas*, sehingga seluruh tulisan berada dalam persegi panjang ini.



Gambar 2.1. *Canvas* dan *bounding box.*
(Sumber: National Instruments Corporation, dengan perubahan)

2. Aspek rasio

   Aspek rasio merupakan rasio dari panjang *bounding box* sebuah karakter dengan tingginya.

3. Pusat massa

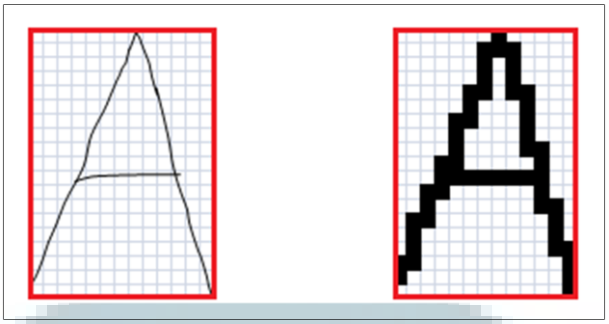
   Titik pusat massa merupakan sebuah titik, dimana apabila dari titik tersebut ditarik suatu vektor ke setiap pixel hitam pada gambar, resultan dari vektor-vektor tersebut adalah nol.

4. Arah penulisan

   Terdapat delapan pengelompokan menyangkut arah penulisan, yaitu ke atas, bawah, kiri, kanan, dan ke keempat arah diagonal. Masing-masing pixel dapat diberikan nilai sesuai arah penulisan yang terjadi pada pixel tersebut.

5. Zonasi

   Zonasi membagi *bounding box* sebuah karakter menjadi N x M kotak-kotak yang berukuran lebih kecil, kemudian menandai kotak yang mendapatkan setidaknya sebuah pixel hitam. Zonasi dapat digunakan untuk membantu ekstraksi fitur-fitur lain, misalnya fitur histogram.



Gambar 2.2. Contoh hasil zonasi pada karakter ‘A’.
(Sumber: Bahri, 2011, dengan perubahan)

6. Histogram kolom

   Sebuah histogram dapat dibentuk dengan mencacah jumlah pixel hitam yang terdapat pada setiap kolom (atau baris). Kolom yang memiliki pixel hitam terbanyak dan tersedikit dapat dijadikan fitur untuk *feature extraction*.

7. Histogram arah penulisan

   Sebuah histogram dapat pula dihasilkan dengan mencacah nilai arah penulisan seperti yang didapatkan dari poin 5 di atas.

8. Titik potong (*junction*)

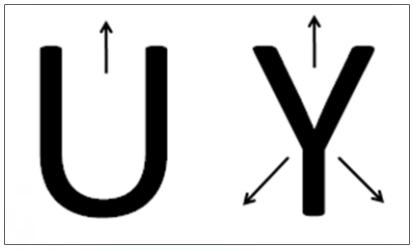
   Jumlah dan lokasi titik potong antara dua garis atau lebih dalam penulisan karakter merupakan fitur yang juga dapat diperhitungkan.

9. Lubang (*holes*)

   Lubang merupakan daerah bidang tulis yang tidak digores, tetapi dikelilingi goresan.

10. Keterbukaan

    Beberapa karakter terbuka ke arah tertentu, misalnya ‘U’ terbuka ke atas, dan ‘Y’ terbuka ke atas, kiri bawah, dan kanan bawah.



Gambar 2.3. Contoh keterbukaan.
(Sumber: Bahri, 2011, dengan perubahan)

11. Jumlah goresan

    Jumlah goresan ditentukan dari berapa kali ujung jari atau pena (selanjutnya disebut alat tulis) menyentuh bidang tulis.

12. Jarak goresan

    Jarak goresan merupakan jarak antara titik dimana alat tulis mulai disentuhkan ke bidang tulis dengan titik dimana alat tulis diangkat dari bidang tulis. Setiap goresan diproses terpisah.

13. Panjang goresan

    Panjang goresan merupakan jarak yang ditempuh oleh ujung alat tulis dari mulai disentuhkan ke bidang tulis sampai diangkat dari bidang tulis. Setiap goresan diproses terpisah.

14. Kelurusan

    Fitur kelurusan digunakan untuk membedakan karakter yang ditulis dengan garis lengkung dan garis lurus.

15. Rasio sisi dan alas

    Rasio sisi merupakan rasio dari posisi dimana alat tulis mulai disentuhkan ke bidang tulis dengan panjang *bounding box* karakter yang sedang ditulis. Rasio sisi juga dapat ditentukan menggunakan posisi saat alat tulis diangkat dari bidang tulis. Dengan cara yang sama, rasio alas dihitung menggunakan tinggi *bounding box*, bukan lebarnya.

16. Keterhubungan antargoresan

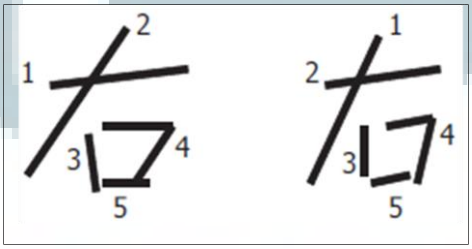
    Goresan-goresan akan disebut terhubung apabila mereka memiliki setidaknya sebuah titik perpotongan. Keterhubungan bersifat asosiatif, dengan artian apabila goresan 1 dan 2 terhubung, dan goresan 2 dan 3 terhubung, maka goresan 1 dan 3 juga terhubung. Goresan-goresan yang terhubung membentuk suatu himpunan yang anggotanya saling terhubung. Keterhubungan antargoresan merupakan jumlah himpunan tersebut pada suatu karakter.

17. Kecepatan gores

    Kecepatan gores diukur dari waktu yang dibutuhkan untuk menyelesaikan sebagian maupun seluruh karakter yang ditulis.

18. Posisi relatif penulisan

    Beberapa karakter dibedakan berdasarkan urutan penulisannya dan posisi relatif dari goresan sebelumnya.



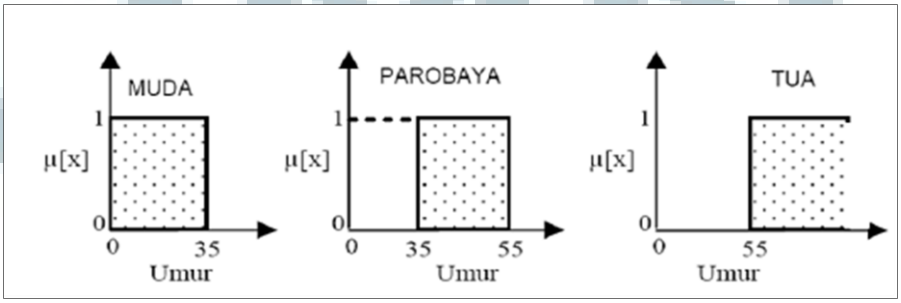
Gambar 2.4. Contoh variasi urutan penulisan.
(Sumber: 蔡, 2012)

2.5 Fuzzy Logic

Menurut Stanford Encyclopedia of Philosophy, *Fuzzy logic* muncul dari pemikiran Lotfi A. Zadeh mengenai *fuzzy set*. *Fuzzy set* adalah kumpulan objek yang memiliki nilai keanggotaan yang bersifat kontinu (Zadeh, 1965). Ini berbeda dari *crisp set*, yang hanya memiliki dua nilai keanggotaan, yaitu ‘ya’ (bernilai 1)

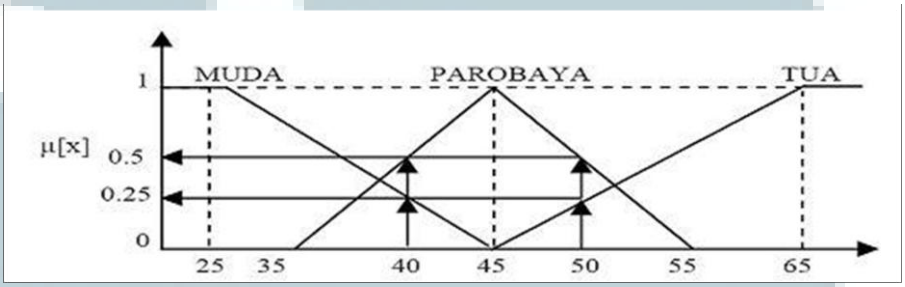
dan ‘tidak’ (bernilai 0). Pada *fuzzy set*, setiap objek boleh memiliki nilai keanggotaan riil di antara 0 hingga 1, inklusif (termasuk kedua ujung rentangnya, yaitu 0 dan 1). Nilai keanggotaan dapat dianggap sebagai nilai kebenaran, sehingga semakin tinggi nilai keanggotaan, maka ia semakin ‘benar’ untuk digolongkan ke dalam kelompok yang bersangkutan. Logika Matematika dimana suatu kebenaran dapat bernilai di antara 0 hingga 1 inilah yang disebut dengan *fuzzy logic*.

Contoh penggunaan *fuzzy set* dan *crisp set*, seperti yang ditulis oleh Sri Kusumadewi dkk. (2006), adalah pengelompokan umur seseorang. Misalnya seseorang dapat digolongkan ke dalam tiga kategori menurut umurnya, yaitu muda, parobaya, dan tua. Misal didefinisikan bahwa yang termasuk muda adalah orang berumur di bawah 35 tahun, parobaya adalah orang berumur 35 hingga 55 tahun, sedangkan tua jika di atas 55 tahun. Definisikan pula μ[x] merupakan nilai keanggotaan untuk kelompok x. Maka, apabila seseorang berumur 35 tahun kurang satu hari (atau bahkan satu detik), ia memiliki μ[muda] = 1 dan μ[parobaya] = 0. Namun, keesokan harinya ia akan memiliki nilai μ[muda] = 0 dan μ[parobaya] = 1. Ini adalah kelemahan penggunaan *crisp set*, dimana perbedaan yang sangat kecil dapat mengakibatkan lonjakan pada nilai keanggotaan.



Gambar 2.5. Penggunaan *crisp set.*
(Sumber: Kusumadewi, 2006)

Hal ini yang coba diperbaiki oleh Zadeh dengan memperkenalkan *fuzzy set*. Nilai keanggotaan yang sifatnya kontinu membuat tidak adanya lonjakan saat ada perbedaan kecil dalam pengambilan data (Zadeh, 1965). Pada gambar 2.6, dapat dilihat bahwa penggunaan fuzzy set dapat membuat seseorang berumur 40 tahun memiliki μ[muda] = 0.25 dan μ[parobaya] = 0.5. Dalam percakapan sehari-hari dapat dikatakan bahwa orang berumur 40 tahun 'tidak terlalu muda' dan 'cukup parobaya'. Fungsi μ[x] bersifat subjektif dan tergantung dari pengamat.



Gambar 2.6. Penggunaan *fuzzy set.*
(Sumber: Kusumadewi, 2006)

2.6 Unicode

Unicode merupakan suatu standar dalam dunia informatika untuk merepresentasikan suatu karakter tertentu. Dalam artikel yang ditulis Joel Spolsky (2003), dijelaskan bahwa untuk merepresentasikan sebuah karakter, perlu dilakukan pemetaan dari sebuah karakter, ke suatu nilai tertentu. Misalnya, karakter 'A' dipetakan ke nilai 1, 'B' ke 2, dan seterusnya. Pemetaan yang dilakukan haruslah bersifat seragam, dalam artian harus ada kesepakatan antara para penggunanya. Apabila skema pemetaan yang digunakan berbeda, maka karakter yang ingin ditampilkan akan dipetakan ke karakter yang keliru, sehingga tulisan hasil akhirnya menjadi tidak dapat dibaca. Dalam contoh di atas, karakter

‘A’ harus selalu dipetakan ke nilai 1, dan tidak boleh ada karakter lain selain ‘A’ yang dipetakan ke nilai 1.

Artikel yang sama menceritakan, Unicode merupakan solusi yang dicetuskan pada akhir tahun 1980-an. Sebelum ada Unicode, pernah juga ada suatu standar dalam pemetaan karakter, yang dinamakan ASCII (*American Standard Code for Information Interchange*). Skema pemetaan ASCII memerlukan 7 bit, beroperasi di antara rentang nilai 0 sampai 127, cukup untuk huruf Latin kecil maupun huruf kapital, angka, serta tanda baca yang umum digunakan. Namun, prosesor komputer bekerja dengan 8 bit sekaligus, dan beroperasi pada rentang 0 hingga 255. Ini menghasilkan ruang kosong untuk 128 karakter baru, yang awalnya ditujukan sebagai cadangan. Tidak seperti ASCII, 128 karakter baru ini tidak memiliki suatu standar yang berlaku di seluruh dunia. Akibatnya ada beberapa karakter berbeda yang dipetakan ke nilai yang sama, dan ada beberapa karakter yang sama tetapi dipetakan ke nilai yang berbeda. Misalnya komputer IBM awalnya memetakan karakter ‘α’ (Yunani *alpha*) ke nilai 224, sementara komputer IBM Rusia memetakan karakter ‘Я’ (Cyrillic) juga ke nilai 224.

Maka, masih menurut Spolsky (2003), Unicode pun dicetuskan untuk menyeragamkan perbedaan ini. Selain menyediakan sebuah standar, Unicode memiliki kemampuan untuk menyimpan jauh lebih banyak karakter dibandingkan skema pemetaan sebelumnya. Pada artikel yang ditulis oleh Tero (2012), tercatat telah 110116 karakter berbeda yang didukung Unicode, dan jumlah ini akan terus diperbanyak. Meskipun Unicode menyimpan lebih banyak karakter, hal ini dapat dilakukan karena setiap karakter tidak lagi dipetakan ke suatu nilai 8 bit, tetapi jauh lebih besar dari itu. Perlu diingat, Unicode bukanlah suatu *character set*.

Unicode hanya berfungsi untuk memetakan sebuah karakter ke suatu nilai tertentu. Implementasi tentang bagaimana mengirimkan dan menerima nilai ini tidak ditangani Unicode. Beberapa skema *encoding* yang jamak digunakan bersama Unicode ialah UTF-8, UTF-16, dan UTF-32, yang pembahasannya di luar cakupan penelitian ini.

2.7 Android

Dalam lingkup penelitian ini, yang dimaksud dengan Android adalah sebuah sistem operasi berbasis Linux untuk perangkat *mobile*. Menurut Android(8), Android mendukung teknologi layar sentuh, *multi-touch*, *bluetooth*, *tethering*, akselerometer, termometer, barometer, magnetometer, illuminometer, GPS, dan giroskop, di samping fitur-fitur yang umum ditemukan pada smartphone lainnya, seperti media penyimpanan, pengiriman pesan, dan dukungan multimedia.

Pada awalnya, Android dikembangkan oleh Android Inc., yang didirikan pada Oktober 2003 oleh Andy Rubin, Rich Miner, Nick Sears, dan Chris White. Perusahaan ini kemudian dibeli oleh Google pada 2005 (Businessweek, 2005). Ponsel komersil pertama yang dapat menjalankan Android adalah HTC Dream, yang dirilis pada Oktober 2008. Sejak itu, Android terus diperbarui berkali-kali. Versi termutakhir saat penelitian ini dibuat ialah 4.4.2 KitKat, yang diluncurkan pada 9 Desember 2013. Menurut artikel yang dilansir oleh HighlightPress pada November 2013, lebih dari 80% *smartphone* yang diproduksi menggunakan Android sebagai sistem operasi.

Sebuah aplikasi Android dapat terdiri atas nol atau lebih *activity*. Dokumentasi Android(1) menjelaskan, *activity* merupakan komponen aplikasi yang

menyediakan tempat untuk interaksi pengguna, misalnya mengambil foto atau mengirimkan pesan. Setiap *activity* biasanya menutupi seluruh layar aplikasi, tetapi kadang ukurannya lebih kecil daripada itu. Sebuah *activity* (disebut *main activity*) biasanya akan diperlihatkan pertama kali segera setelah pengguna mengaktifkan suatu aplikasi. Setiap *activity* dapat memanggil *activity* lainnya untuk dapat melakukan aksi yang berbeda.

Sebuah *activity* dapat memiliki beberapa *view*. Dalam dokumentasi Android(9), *view* merupakan komponen antarmuka aplikasi dimana *user* langsung berinteraksi dengan *view* tersebut. Dengan demikian, pengguna berinteraksi dengan *view*, sementara *activity* hanyalah tempat meletakkan *view*. Contoh *view* bawaan Android misalnya *TextView*, *Button*, *ImageView*, *EditText*, dan *ListView*.

Dalam Android, untuk mengatur letak *view* dapat digunakan *layout* (Android(4)). Adapun *layout* yang disediakan secara bawaan contohnya *LinearLayout*, *RelativeLayout*, dan *AbsoluteLayout*. Android juga menyediakan pilihan bagi pengembang untuk dapat membuat *view* maupun *layout* sendiri (*custom view, custom layout*) apabila dirasa bahwa *view* atau *layout* bawaan Android tidak sesuai dengan kebutuhan.

2.8 Drawing View

Drawing view merupakan sebuah *view* bagi *user* untuk menulis pada layar perangkat, untuk kemudian hasil tulisannya (tidak perlu suatu alfabet tertentu, boleh saja coretan abstrak) ditampilkan pada *view* tersebut. Android tidak menyediakan *view* khusus untuk itu, tetapi Android menyediakan sebuah *class Canvas* yang dapat digunakan untuk menyimpan dan menampilkan citra digital

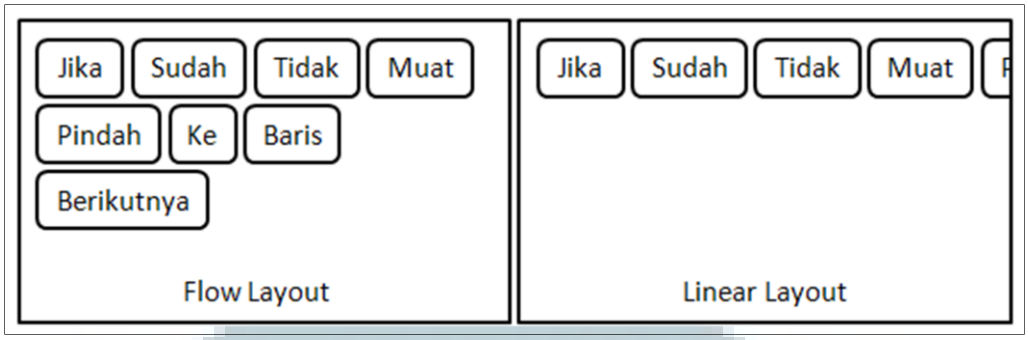
dalam bentuk *bitmap*. *Drawing view* dapat dibuat secara *custom* dengan membuat *view* baru yang diturunkan dari *class View*, dan memberikan variabel bertipe data *Canvas* kepada *view* baru tersebut (Smith, 2013).

Selain *class Canvas*, *drawing view* juga memerlukan *class* berikut: *Path*, *Paint*, dan *Bitmap*. Menurut Android(2), Android(3), Android(6), dan Android(7), *Class Path* berguna untuk merekam alur goresan ketika pengguna menulis. *Class Paint* berfungsi untuk menyimpan warna, ketebalan garis, *alpha* (tingkat transparansi), maupun karakteristik lain dari garis yang akan tampil saat *user* menulis. Analoginya, apabila *class Canvas* merupakan kertasnya, maka *class Paint* merupakan penanya. *Class Bitmap* menyimpan data dari goresan-goresan yang sudah ada. Data ini digunakan *class Canvas* untuk menampilkan citra digital.

2.9 Flow Layout

Menurut dokumentasi Oracle, *Flow Layout* merupakan jenis *layout* untuk menyusun objek-objek membentuk sebuah baris, dimana tidak akan dibuat baris baru, kecuali baris sebelumnya sudah penuh. Selain mengisi baris, implementasi *flow layout* dapat juga dibuat mengisi kolom.

Sampai pada API 19 (KitKat), Android masih belum mendukung penggunaan *flow layout*. Penggunaan *linear layout* yang dimodifikasi dapat dijadikan alternatif *flow layout*. *Linear layout* sendiri pada dasarnya berbeda dengan *flow layout*, karena dalam penggunaan *linear layout*, tidak akan pernah dibuat baris baru, meskipun baris sebelumnya sudah penuh (Android(5)).



Gambar 2.7. Ilustrasi *flow layout* dan *linear layout.*
(Sumber: Oracle dan ApmeM, dengan perubahan).

2.10 Algoritma Midpoint Line

Algoritma ini disebut juga dengan *Bresenham's line algorithm*, yakni nama penemu algoritma ini. Tujuan algoritma ini ialah menemukan himpunan pixel yang paling mendekati suatu segmen garis lurus (Sukhendu Das, 2010). Algoritma ini dapat bekerja dengan cepat dan akurat karena semua perhitungannya dilakukan hanya dengan bilangan bulat, tanpa *floating number* yang memberatkan komputasi. Gambar 2.8 merupakan *pseudocode* dari algoritma tersebut.

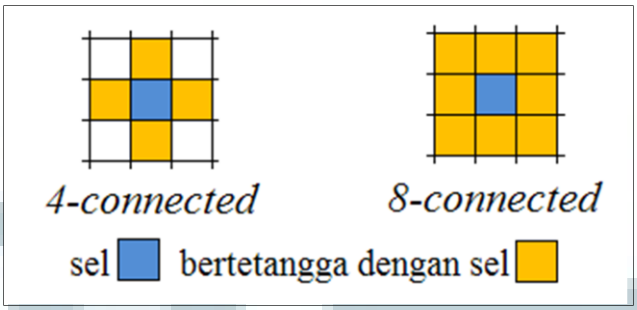
```
function line(x1, y1, x2, y2)
  if x1 > x2 then
    line(x2, y2, x1, y1)
    end function
  end if

  dx := x2-x1
  dy := ABS(y2-y1)
  if y1 > y2 slope := -1
  else slope := 1

  incE := 2*dy
  incNE := 2*dy - 2*dx
  d := 2*dy - dx
  y := y1

  loop
    putpixel(x,y)
    if d > 0 then
      d = d + incNE
      y = y + slope
    else
      d = d + incE
    end if
  end loop
end function
```

Gambar 2.8. *Pseudocode* untuk algoritma *midpoint line.*
(Malik, dengan perubahan)

2.11 Algoritma Flood Fill

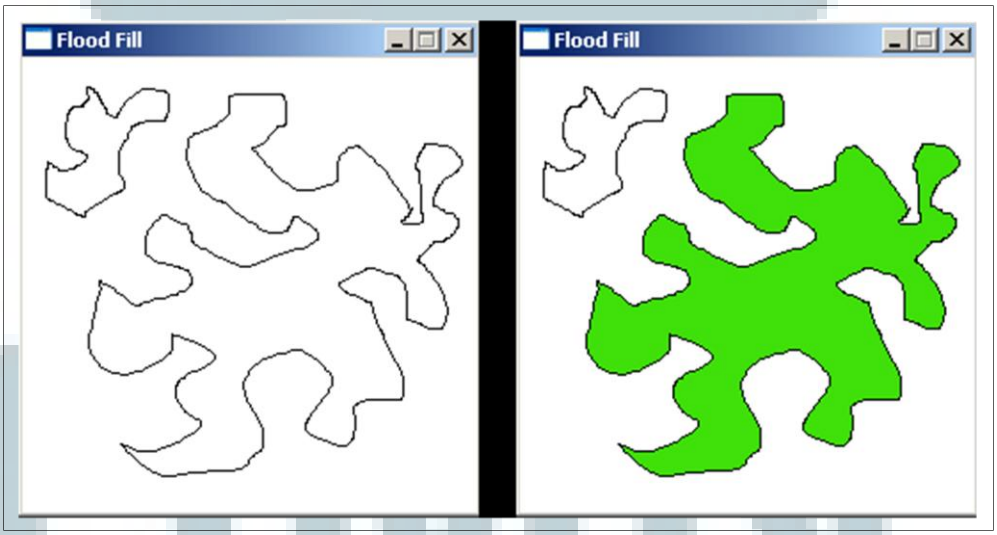
Menurut Vandevenne (2004), *Flood fill* merupakan sebuah algoritma untuk menemukan sel mana saja yang terkoneksi pada sebuah sel tertentu dalam sebuah *array* 2D (atau lebih). Sebuah sel dikatakan terkoneksi dengan sel lain apabila sel kedua merupakan tetangga dari sel pertama, atau sel kedua merupakan tetangga dari sebuah sel lain yang juga terkoneksi dengan sel pertama. Pada umumnya, yang dimaksud dengan tetangga dari sebuah sel pada *array* 2D adalah sel yang berada tepat di kiri, kanan, atas, dan bawah dari sel tersebut. Penggunaan definisi tetangga seperti ini disebut *four-connected*. Seringkali juga digunakan definisi tetangga *eight-connected*, dimana terdapat delapan sel yang bertetangga dengan

suatu sel, yaitu empat sel seperti definisi *four-connected*, ditambah sel pada keempat arah diagonal.



Gambar 2.9. Ilustrasi tetangga secara *4-connected* dan *8-connected*.
(Sumber: Wolfram Mathworld(1) dan Wolfram Mathworld(2), dengan perubahan)

Algoritma ini biasa digunakan untuk mengimplementasikan *bucket tool* pada aplikasi pengolahan gambar, seperti pada Ms. Paint (Vandevenne, 2004). Dalam hal ini, *canvas* untuk menggambar merupakan *array* 2D, dan setiap pixel merupakan sel penyusun *array* 2D tersebut. Definisi tetangga yang digunakan adalah *four-connected*.



Gambar 2.10. Contoh penggunaan *bucket tool* untuk mewarnai.
(Sumber: Vandevenne, 2004)

Sumber yang sama juga menyebutkan bahwa algoritma *flood fill* dapat dijalankan secara rekursif, maupun iteratif dengan bantuan sebuah *stack*. *Pseudocode* dari algoritma ini terdapat pada gambar 2.11.

```
function floodfill(x, y, newColor, oldColor)
  if pixel[y][x] = oldColor then
    pixel[y][x] := newColor
    floodfill(x+1, y, newColor, oldColor)
    floodfill(x, y+1, newColor, oldColor)
    floodfill(x-1, y, newColor, oldColor)
    floodfill(x, y-1, newColor, oldColor)
  end if
end function
```

Gambar 2.11. *Pseudocode* untuk algoritma *flood fill.*
(Cohen-Or, 2013, dengan perubahan)