Penggunaan Algoritma *Brute Force* dalam *Template Matching* di Program *Face Recognition* Sederhana

Muhammad Reza Ramadhan - 13514107
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
rezaramadhan.m@students.itb.ac.id

***Abstract* — Algoritma yang digunakan dalam *face recognition* merupakan algoritma yang sangat kompleks, mempertimbangkan banyak hal, dan memerlukan waktu dan kekuatan komputasi yang tinggi. Makalah ini akan membahas salah satu algoritma sederhana yang akan digunakan dalam *face recognition* dengan memanfaatkan *Sum of Absolute Difference* dan penilaian berbeda untuk setiap komponen wajah yaitu mata, hidung, mulut, dagu, daun telinga dan tulang pipi.**

Keywords— Face, grayscale, image, pixel, SAD, template matching.

I. PENDAHULUAN

Dengan jumlah manusia di bumi yang pada tahun 2013 mencapai 7,125 miliar, kadang diperlukan cara untuk membedakan satu manusia dengan manusia yang lain. Dengan pengecualian kembar identik, tidak ada manusia yang memiliki bagian tubuh yang sama persis dengan manusia lain. Terdapat beberapa bagian tubuh yang bisa digunakan untuk mengenali perbedaan antara seorang manusia dengan manusia lain seperti sidik jari, lidah, dan wajah.

Dewasa ini, *Face Recoginition* bukanlah hal yang aneh, media sosial seperti Facebook telah menggunakannya untuk mempermudah mengenali foto yang diunggah oleh seorang pengguna; Apple memanfaatkannya dalam perangkat lunak Photo dalam OS X dan iPhoto dalam iOS; Adobe menggunakannya dalam perangkat lunak Lightroom – sebuah perangkat lunak *editor* foto berbayar. Selain kegunaannya dalam media sosial, perangkat lunak *image viewer*, ataupun perangkat lunak *image editor*, *face recognition* juga dapat digunakan sebagai salah satu fitur kemananan, seperti dalam penggunaannya di *lockscreen* Android ataupun dalam CCTV untuk mengenali orang asing yang terekam.

II. TEORI DASAR

A. Pattern Matching

Secara sederhana, *pattern matching* atau *string matching* dapat dideskripsikan sebagai berikut: Diberikan sebuah *text*, yaitu kumpulan karakter (*string*) dengan panjang n karakter; serta sebuah *pattern*, yaitu kumpulan karakter (*string*) dengan panjang m karakter dengan m < n. Proses akan mencari posisi *pattern* pertama kali muncul di dalam *text*.

Contoh dari pattern matching sendiri adalah:

```
Text          : KUKU KAKIKU KAKU-KAKU
Pattern       : AKI
Ditemukan di : karakter ke-7
```

Salah satu metode untuk memecahkan masalah *pattern matching* ini adalah dengan algoritma bruteforce, atau yang sering disebut sebagai algoritma naif. Dengan asumsi bahwa *text* T berada di dalam *array of* karakter dengan panjang n dan *pattern* P berada di dalam *array of* karakter dengan panjang m, maka algoritma bruteforce yang dapat digunakan adalah sebagai berikut:

1. Mula-mula *pattern* P dicocokkan pada awal *text* T.
2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam *text* T sampai:
    a. Semua karakter yang dibandingkan cocok (pencarian berhasil), atau
    b. Dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3. Bila *pattern* P belum ditemukan kecocokannya dan *text* T belum habis, geser *pattern* P satu karakter ke kanan dan ulangi langkah 2.

Berikut adalah contoh penggunaan algoritma bruteforce pada pencarian *pattern AKI* di dalam *text KUKU KAKIKU KAKU KAKU*

```
KUKU KAKIKU KAKU-KAKU
AKI
 AKI
  AKI
   AKI
    AKI
     AKI
      AKI <-ditemukan pada karakter ke-7
```

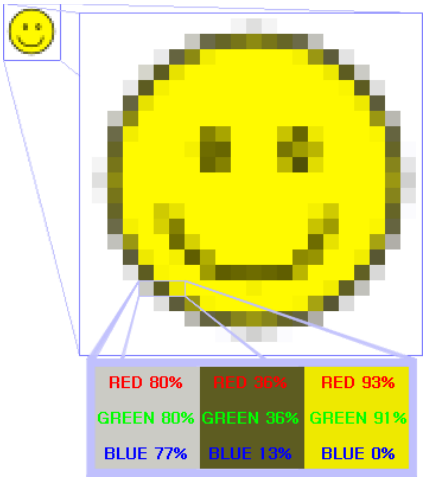
B. Digital Image

Di dalam komputer, gambar dapat disimpan dalam dua bentuk, yaitu vektor dan *raster*. Gambar yang terbuat dari vektor biasanya merupakan karya digital yang dibuat untuk kebutuhan *publishing* seperti logo, poster, dan lukisan digital. Sedangkan

gambar *raster* adalah gambar yang biasa didapatkan dalam foto-foto yang diambil oleh kamera digital.

Dalam gambar *raster*, gambar terdiri dari kumpulan matriks elemen yang bernama *pixel*. *Pixel* adalah komponen terkecil gambar yang membentuk warna dari suatu titik di gambar. *Pixel* sendiri dapat diisi oleh berapa *bit* tergantung kebutuhan. Gambar hitam-putih dapat digambarkan hanya membutuhkan 1 *bit* untuk *pixel*, yaitu 1 untuk menggambarkan hitam dan 0 untuk putih. Sementara gambar yang lebih kompleks akan menggunakan lebih dari satu *bit*.

Pada gambar yang biasa digunakan dalam kehidupan sehari-hari, digunakan konsep RGB yang menggambarkan intensitas warna merah (*Red*/R), hijau (*Green*/G), dan biru (*Blue*/B) yang membentuk sebuah pixel.



Gambar 1: Ilustrasi Pixel

Sumber: http://www.petervaldivia.com/technology/Computer-graphics/image/Rgb-raster-image.png

Pada *digital image*, dikenal juga resolusi yaitu angka yang menggambarkan jumlah *pixel* yang ada di gambar tersebut. Sebagai contoh, gambar berukuran 800x480 berarti memiliki matriks *of pixel* dengan jumlah kolom 800 dan jumlah baris 480.

C. *Template Matching*

Secara singkat, *template matching* adalah bentuk dua dimensi dari *pattern matching.* Pada *template matching*, tujuan utamanya adalah menentukan bagian dari gambar digital (*digital image)* yang cocok atau mirip dengan *template* yang telah ada.

Template matching dapat dianggap sebagai bentuk dua dimensi dari *pattern matching* karena jika *text* dianalogikan sebagai gambar digital, *pattern* dianalogikan sebagai *template* tersedia, serta *array* penyimpanan karakter dianalogikan sebagai matriks penyimpanan *pixel*, maka tidak ada perbedaan signifikan antara *pattern matching* dan *template matching.*

Berikut adalah contoh dari *template matching:*

Image:

Gambar 2: Contoh image
Sumber: http://www.prep-blog.com/wp-content/uploads/2012/11/Crowded-Subway-Station.jpg

Template:

Gambar 3: Contoh template

Pencarian:

Gambar 4: Pencarian

Pada *template matching*, jika dilakukan metode *exact matching* atau metode pecocokan secara pasti, maka akan sangat jarang ditemukan *template* tertentu pada gambar yang ada, kecuali jika template berasal dari gambar yang sama. Karena itulah digunakan beberapa metode dengan toleransi tertentu untuk menentukan posisi *template* didalam gambar.

Salah satu metode tersebut adalah *Sum of Absolute Difference* (SAD). Metode ini menentukan nilai perbedaan pixel yang ada antara sebuah gambar dan templatenya.

Pada metode ini, kita misalkan gambar yang dicari sebagai G dan *template* sebagai T, dan (x, y) menggambarkan nilai *pixel* di koordinat x, y, maka dapat digunakan persamaan

$$SAD(x,y) = \sum_{i=0}^{Trow} \sum_{j=0}^{Tcol} |G(x+i, y+j) - T(i,j)|$$

Dari persamaan diatas, kita hanya perlu mencari nilai minimum dari SAD(x,y) untuk semua nilai x dan y yang mungkin. Kita juga perlu menentukan batas toleransi, misalnya kita hanya akan menerima SAD(x,y) minimum jika nilai minimum tersebut lebih kecil daripada 10, jika semua nilai SAD(x,y) lebih besar daripada 10 maka *template* tidak ditemukan didalam *image*.

Metode SAD ini merupakan metode yang mudah untuk diimplementasi dan dipahami, namun bisa terlihat bahwa metode ini akan membutuhkan waktu pemecahan masalah yang cukup lama.

III. Pemecahan Masalah

Dalam pemecahan masalah kali ini, masalah akan sangat dibatasi karena *face recognition* yang bisa dipecahkan oleh program ini sangatlah kecil jangkauannya. Beberapa batasan yang penulis tentukan adalah:

1. Ukuran/resolusi antara wajah seseorang di *image* dengan wajah seseorang di *template* harus sama.
2. Pengujian hanya berhasil jika pada *image* wajah seseorang tepat menghadap ke depan dan tidak terputar dengan sudut besar.
3. Pengujian berhasil pada data uji dengan ekspresi wajah yang relatif sama.

Kedua batasan pertama diatas akan dapat diatasi dalam dengan menggunakan program lain yang akan mendeteksi wajah manusia, mengubahnya menjadi resolusi yang sesuai, serta memutarnya hingga terdapat derajat rotasi yang sesuai.

A. *Pengolahan Gambar Awal*

Untuk mengatasi perbedaan warna antara *template* dan image, maka kita perlu merubah *image* yang ada menjadi bentuk grayscale, yaitu gambar yang hanya menyimpan intensitas warna hitam saja. Setiap *pixel* pada gambar *grayscale* hanya membutuhkan 8*bit* saja, dibandingkan dengan 24*bit* yang ada pada gambar RGB. Selain itu, dengan berubahnya warna menjadi *grayscale* maka akan mempermudah cara perhitungan di metode *Sum of Absolute Difference*.

Cara untuk mentransformasikan sebuah gambar menjadi *grayscale* adalah membuat sebuah gambar baru dengan 8*bit pixel* dan mengisi intensitas dari *pixel* tersebut dengan mengikuti persamaan:

$$I = 0.2989 * R + 0.5870 * G + 0.1140 * B \text{ [4]}$$

Persamaan diatas didapat dari analisis kekuatan intensitas persepsi tiga warna dari mata manusia. Mata manusia lebih sensitif terhadap warna hijau dibandingkan dengan warna biru.

Berikut adalah contoh perubahan gambar dari warna ke *grayscale* dengan mengikuti persamaan diatas.

Gambar 5: Perubahan Gambar Berwarna ke Grayscale

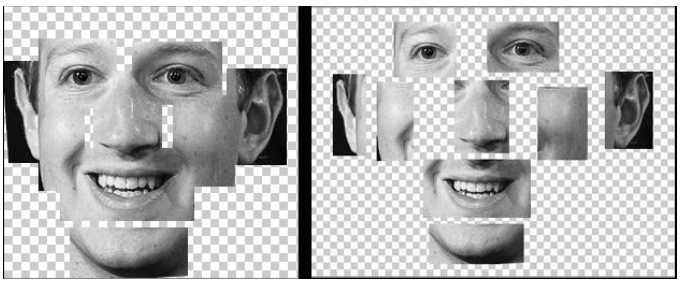
Sumber: https://fbnewsroomus.files.wordpress.com/2014/05/mark-zuckerberg-headshot.jpg

B. *Analisa Setiap Komponen Wajah*

Wajah manusia sangatlah kompleks, jika proses *face recognition* hanya berdasarkan wajah manusia secara keseluruhan maka kemungkinan terjadi kesalahan sangat besar. Karena itulah dalam program ini digunakan analisis yang terpisah untuk setiap komponen wajah. Dengan demikian, diharapkan akan mengurangi kemungkinan terjadinya kesalahan jika posisi setiap komponen sedikit tergeser.

Selain itu, pencarian terpisah untuk komponen wajah akan membuat toleransi terhadap gambar jika salah satu komponen wajah tertutup atau tidak terlihat. Program akan menganalisa beberapa kondisi khusus dimana salah satu komponen tidak terlihat pada gambar, seperti saat menggunakan *eyepatch*, atau ketika daun telinga tertutup rambut.

Template yang disimpan didalam *database* wajah pada program ini bukan merupakan gambar dari wajah seseorang secara penuh, namun merupakan *tuple* yang terdiri dari komponen dari wajah seseorang yaitu kedua mata dan alis, batang hidung, tulang pipi, daun telinga, serta dagu. *Template* tersimpan ini juga tentu merupakan data *grayscale* dari gambar wajah sebenarnya.



Gambar 6: Komponen Wajah Yang Akan Dicari

Sumber: (sebelum diolah) http://cp91279.biography.com/1000509261001/1000509261001_1822909398001_BIO-Biography-29-Innovators-Mark-Zuckerberg-115956-SF.jpg

C. *Proses Pencarian*

Pada dasarnya, pencarian akan dilakukan dengan metode pencarian nilai *Sum of Absolute Difference* (SAD) terkecil antara *template* dengan *image*. Nilai hasil perhitungan SAD disimpan pada *list* sebagai tempat penyimpanan sementara. Pada akhir pencarian, akan dilakukan pecarian nilai SAD terkecil untuk penentu apakah pencarian ditemukan atau tidak. Pencarian

dilakukan beberapa kali sesuai dengan jumlah komponen berbeda yang ada di dalam *template.*

Sebagai contoh, berikut adalah proses yang akan terjadi ketika pencarian komponen batang hidung:

1. Simpan *template* pada bagian kiri atas gambar, dimana merupakan koordinat pixel 0,0.
   a. Hitung nilai SAD pada koordinat tersebut.
   b. Cek hasil perhitungan, apakah 0 atau tidak
      - Jika 0, maka ditemukan posisi sama persis, pencarian ketemu. Lanjutkan ke langkah 2
      - Jika tidak sama dengan 0, masukkan nilai hasil perhitungan pada sorted list. Lanjut ke langkah c
   c. Geser template satu koordinat ke kanan, jika tidak memungkinkan, geser template satu koordinat ke bawah dan simpan *template* di ujung paling kiri. Jika pergeseran ke bawah tidak mungkin lanjutkan ke langkah 2.
   d. Kembali ke langkah b.
2. Cari nilai paling kecil di *list* yang berisi nilai SAD.
3. Bandingkan dengan nilai toleransi, putuskan apakah pencarian berhasil atau tidak. Jika berhasil simpan koordinat tempat ditemukannya.

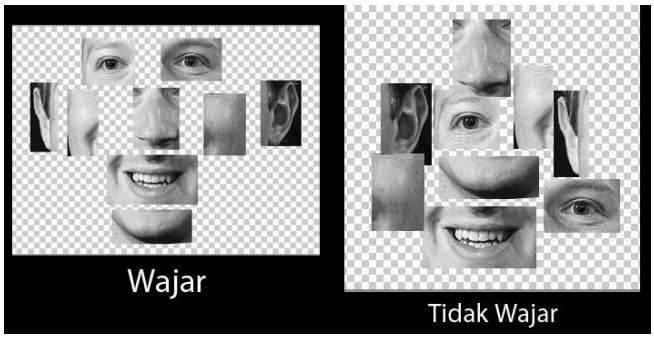
Proses pencarian tersebut akan berulang beberapa kali hingga seluruh komponen wajah telah dicari. Setelah seluruh komponen wajah dicari dalam *image*, maka akan terdapat data dimana komponen tersebut ditemukan

D. Pemrosesan Hasil Pencarian

Setelah dilakukan pencarian, program akan mengecek komponen apa saja yang tidak ditemukan pada pencarian di *image.* Program akan mengecek apakah terdapat kasus khusus dimana komponen tersebut mungkin tidak terlihat, jika tidak terdapat kasus khusus tertentu maka program akan menyatakan bahwa pencarian tidak cocok.

Jika seluruh komponen berhasil diidentifikasi pada *image*, maka program akan mengecek tempat ditemukannya komponen tersebut untuk mengecek apakah lokasi ditemukannya wajar atau tidak.

Lokasi ditemukan akan dianggap wajar jika memenuhi beberapa kondisi, diantaranya adalah dua buah mata ada dalam koordinat horizontal yang perbedaannya masuk dalam batas toleransi, tulang pipi berada di koordinat vertikal di bawah koordinat vertikal mata, mulut berada dibawah hidung, dan sebagainya.



Gambar 7: Beberapa Hasil Pencarian Komponen

IV. ANALISIS PROGRAM

A. Kelebihan

Program ini dapat mengatasi beberapa kasus dimana komponen tertutupi oleh aksesoris tertentu sehingga tidak bisa terbaca oleh gambar. Selain itu, program juga mudah untuk diimplementasikan pada bahasa apapun.

B. Kekurangan

Kecepatan eksekusi adalah kekurangan terbesar dari program ini. Karena penggunaan metode *brute force* dan *Sum of Differences* yang memastikan bahwa setiap *pixel* yang ada di gambar pasti dihitung berulang-ulang. Kompleksitas dari algoritma ini sendiri dapat dihitung dengan:

Jika diberikan resolusi *image* adalah m x n dan resolusi *template* adalah p x q, maka proses perhitungan SAD akan terjadi sebanyak $(m - p)$ x $(n - q)$ kali. Sementara itu, untuk setiap perhitungan SAD pasti dilakukan operasi pengurangan sebanyak pq kali. Setelah itu, operasi ini dilakukan berulang-ulang untuk setiap komponen wajah yang ada. Jika banyaknya komponen wajah adalah sebuah konstanta c, maka, kompleksitas waktu program ini terhadap jumlah pengurangan yang terjadi adalah

$$T(n) = pq(m - p)(n - q)*c$$

Jika diasumsikan m >> p dan n >> q, maka didapat kompleksitas waktunya adalah

$$O(mn)$$

Sebagai simulasi dari lamanya kecepatan eksekusi program ini, misalkan terdapat gambar yang diambil oleh kamera beresolusi 5MP (2560x1960), terdapat pula 9 buah *template* yang secara rata-rata memiliki resolusi 70x40. Maka program akan melakukan operasi pengurangan sebanyak

$$jml = 70 * 40 * (2560 - 70) * (1960 - 40) * 9$$

$$jml = 1{,}2x10^{11}$$

Jika diasumsikan satu operasi pengurangan memakan waktu 5 *nanosecond*, maka untuk menganalisa suatu image oleh suatu *template* tertentu akan memakan waktu sebanyak:

$$totalWaktu = 1{,}2x10^{11} * 5x10^{-9}$$

$$totalWaktu = 600$$

Didapat total waktu sebanyak 600 detik atau setara dengan 10 menit. Jelas ini menunjukan bahwa program ini tidak bisa digunakan dalam kehidupan sehari-hari dimana proses *face recognition* diperlukan waktu yang cepat untuk mendapatkan hasil yang diinginkan.

Keterbatasan untuk tidak mampu mengenali jika terdapat sudut yang signifikan antara wajah di *image* dengan batas normal juga merupakan kelemahan dari program ini. Selain itu, terdapat juga ketidakmampuan program untuk menangani berbagai ekspresi wajah. Dua buah kekurangan ini sendiri sangat menjadi kekurangan besar bagi program karena pada kehidupan sehari-hari manusia sering berfoto dengan berbagai ekspresi dan dengan sudut rotasi tertentu.

Beberapa batasan dari program ini bisa dipenuhi oleh program lain yang memiliki tujuan untuk menyelesaikan masalah tersebut. Namun hal ini menjadi salah satu kekurangan program dimana program akan sangat bergantung terhadap hasil yang telah diolah oleh program lain.

C. Pengembangan Lebih Jauh

Untuk memperbaiki beberapa kekurangan dari program ini, terdapat beberapa langkah yang bisa dilakukan, diantaranya adalah membuat program tambahan yang bertujuan khusus untuk mengolah *image* sebelum diproses menggunakan *template matching* metode Sum of Absolute Differences.

Ide penulis adalah membuat program untuk memperoleh data lokasi wajah-wajah yang berada di foto, ukuran dari wajah-wajah yang ada, serta sudut dari wajah tersebut ke rotasi normal. Jika ketiga informasi tersebut didapatkan, maka kita dapat mengolah *image* menjadi beberapa *array of image*. Setiap elemen *array of image* adalah sebuah *image* yang hanya akan mengandung gambar wajah orang saja, tidak dengan *background* dari gambar awal.

Jika *image* awal bisa direduksi menjadi *array of image*, maka bisa terjadi pengurangan signifikan pada proses perhitungan dan pencarian nilai SAD untuk setiap *template*. Dengan demikian akan mengurangi waktu eksekusi.

Gambar 8: Ilustrasi Output Program Tambahan

Sumber: http://i.dailymail.co.uk/i/pix/2014/03/13/article-2580055-1C04F6CB00000578-561_634x352.jpg

Sebagai contoh, jika sebuah image 5MP(2560x1960) dapat direduksi menjadi *array of image* berisi 3 buah *image* dengan rata-rata resolusi 200x300, terdapat pula 9 buah *template* yang secara rata-rata memiliki resolusi 70x40. Maka program akan melakukan operasi pengurangan sebanyak

$$jml = (70 * 40 * (200 - 70) * (300 - 40) * 9) * 3$$

$$jml = 2{,}5 * 10^9$$

Jika diasumsikan satu operasi pengurangan memakan waktu 5 *nanosecond*, maka untuk menganalisa suatu *image* oleh suatu *template* tertentu akan memakan waktu sebanyak:

$$totalWaktu = 2{,}5x10^9 * 5x10^{-9}$$

$$totalWaktu = 12$$

Didapat total waktu minimum 12 detik untuk menganalisa sebuah *image* yang memiliki resolusi 5MP. Terdapat peningkatan kecepatan eksekusi hingga 50 kali lipat. Proses kalkulasi ini memang memakan waktu yang cukup lama, tapi ini jauh lebih baik dibandingkan dengan proses sebelumnya. Sayangnya, metode ini akan kurang tepat digunakan pada gambar yang berisi sekumpulan orang yang berfoto bersama. Pada kondisi seperti ini tidak banyak komponen dari gambar yang akan tereduksi.

Selain mengurangi waktu pemrosesan, pengembangan program lebih jauh ini juga akan mengatasi beberapa batasan lain sehingga program akan lebih fleksibel mengatasi data image masukan.

Batasan "Ukuran/resolusi antara wajah seseorang di image dengan wajah seseorang di template harus sama." akan tidak berlaku lagi jika kita bisa mendapatkan data lokasi dan ukuran wajah yang terdapat di *image*. Dengan kedua data tersebut, kita bisa melakukan *scaling* terhadap *image* yang disimpan di *array* maupun *template* saat dilakukan perbandingan untuk mendapatkan resolusi yang ideal sehingga bisa dilakukan perbandingan dengan metode SAD.

Batasan "Pengujian hanya berhasil jika pada *image* wajah seseorang tepat menghadap ke depan dan tidak terputar dengan sudut besar." dapat teratasi jika kita mendapat data sudut rotasi wajah tersebut terhadap wajah normal. Untuk mengatasinya, kita perlu melakukan rotasi terhadap *image* sebelum dimasukkan ke dalam *array of image* untuk mendapatkan data yang bisa diolah dengan perbandingan biasa.

V. KESIMPULAN

Face recognition merupakan salah satu komputasi kompleks yang hingga saat ini belum terdapat algoritma yang efektif dan sederhana untuk memecahkan masalah ini. Sebagian besar algoritma *face recognition* memerlukan pemahaman mendalam mengenai grafika komputer serta karakteristik dari wajah manusia.

Dalam makalah ini, penulis mencoba menggunakan konsep *pattern matching* dan fakta bahwa setiap manusia memiliki komponen wajah yang berbeda-beda untuk memecahkan masalah *face recognition.* Hasil yang didapat adalah algoritma yang sederhana namun tidak efektif. Algoritma yang didapat diharapkan dapat memecahkan kondisi ideal dari beberapa perbandingan wajah yang ada.

REFERENCES

[1] *Template Matching* - http://docs.adaptive-vision.com/current/studio/machine_vision_guide/TemplateMatching.html diakses pada 6 Mei 2016.

[2] *Sum of Absolute Differences (SAD)* | siddhant ahuja (sid) - https://siddhantahuja.wordpress.com/tag/sum-of-absolute-differences-sad/ diakses pada 8 Mei 2016.

[3] Munir, Rinaldi (2005), Diktat Kuliah IF2211 Strategi Algoritmik.

[4] *Matlab Tutorial : Digital Image Processing 3 - Grayscale Image I* - http://www.bogotobogo.com/Matlab/Matlab_Tutorial_Digital_Image_Processing_3_Grayscale_RGB.php diakses pada 8 Mei 2016

VI. PERNYATAAN

Dengan ini penulis menyatakan bahwa makalah yang penulis tulis ini adalah tulisan penulis sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Mei 2015

Muhammad Reza Ramadhan
13514107