

# Analisis Perbandingan Algoritma Pattern Matching Knuth-Morris-Pratt dan Boyer-Moore Untuk Metode Pemrosesan Citra Template Matching

Bagas Aryo Seto - 13520181  
Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung  
E-mail (gmail): 13521081@std.itb.ac.id

**Abstract**— Pemrosesan citra *template matching* adalah teknik penting dalam pengolahan citra yang digunakan untuk mencari dan mencocokkan pola tertentu dalam citra. Dalam makalah ini, proses *pattern matching* digunakan untuk pemrosesan citra *template matching*. Selain itu, dilakukan analisis perbandingan antara dua algoritma populer dalam proses *pattern matching*, yaitu algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore. Algoritma KMP dan Boyer-Moore memiliki pendekatan yang berbeda dalam mencocokkan pola. Algoritma KMP menggunakan informasi dari pencarian sebelumnya untuk mempercepat proses pencarian berikutnya, sedangkan algoritma Boyer-Moore menggunakan informasi dari pola yang ingin dicari dan melakukan penggeseran pintar berdasarkan karakter terakhir yang tidak cocok. Penelitian ini bertujuan untuk membandingkan kinerja kedua algoritma berdasarkan kecepatan eksekusi, banyak proses perbandingan, dan akurasi hasil pencocokan.

**Keywords**—*template matching, pattern matching, Knuth-Morris-Pratt, Boyer-Moore*

## I. PENDAHULUAN

Pada era digital saat ini, pengolahan citra telah menjadi bidang yang sangat penting dan berkembang pesat. Citra digital digunakan dalam berbagai aplikasi, termasuk pengenalan wajah, deteksi objek, pengolahan medis, dan banyak lagi. Salah satu teknik penting dalam pengolahan citra adalah metode pemrosesan citra *template matching*.

Metode pemrosesan citra *template matching* adalah teknik yang digunakan untuk mencari dan mencocokkan pola tertentu dalam citra. Tujuan utamanya adalah untuk menemukan keberadaan objek atau pola tertentu dalam citra yang lebih besar. Dalam metode ini, sebuah citra *template*, yang merupakan citra kecil yang mencerminkan pola yang ingin dicari, dibandingkan dengan citra sumber.

Dalam melakukan pemrosesan citra *template matching*, diperlukan penggunaan algoritma yang efisien dan akurat. Dalam makalah ini, akan digunakan proses *pattern matching* algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore (BM) untuk melakukan pemrosesan citra *template matching*.

Algoritma Knuth-Morris-Pratt adalah algoritma pencocokan pola yang efisien, yang dikembangkan oleh Donald Knuth,

James Morris, dan Vaughan Pratt pada tahun 1977. Algoritma ini didasarkan pada konsep penggunaan informasi yang diperoleh dari pencarian sebelumnya untuk mempercepat proses pencarian berikutnya. Metode ini menghindari pencocokan ulang yang tidak perlu dengan memanfaatkan informasi pola yang telah ditemukan sebelumnya.

Sementara itu, algoritma Boyer-Moore, yang dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977, juga merupakan algoritma pencocokan pola yang efisien. Algoritma ini menggunakan pendekatan berbeda dengan KMP, dengan memanfaatkan informasi dari pola yang ingin dicari dan menggunakan penggeseran pintar berdasarkan karakter terakhir pola yang tidak cocok.

Tujuan dari analisis perbandingan ini adalah untuk membandingkan kinerja kedua algoritma dalam konteks pemrosesan citra *template matching*. Aspek-aspek yang akan dianalisis meliputi kecepatan waktu eksekusi, efisiensi banyak proses perbandingan, dan akurasi hasil pencocokan. Melalui perbandingan ini, diharapkan dapat ditemukan kelebihan dan kekurangan masing-masing algoritma, sehingga dapat memberikan wawasan yang berguna dalam memilih algoritma yang tepat untuk aplikasi pemrosesan citra *template matching*.

## II. TEORI DASAR

### A. Pattern Matching

*Pattern matching* adalah proses mencari kemunculan atau pencocokan dari suatu pola (pattern) dalam suatu teks atau data. Tujuannya adalah untuk mengidentifikasi apakah ada kesesuaian atau kecocokan antara pola yang dicari dengan bagian-bagian dari teks atau data yang sedang dianalisis.

Proses *pattern matching* melibatkan perbandingan antara pola yang telah ditentukan dengan bagian-bagian dari teks secara sistematis. Pola dapat berupa urutan karakter, pola grafis, pola numerik, atau kombinasi dari semuanya. Teknik *pattern matching* yang tepat akan memungkinkan kita untuk menemukan kejadian atau entitas yang sesuai dengan pola yang dicari, meskipun pola tersebut mungkin tersembunyi di dalam teks yang panjang atau data yang kompleks.

*Pattern matching* memiliki aplikasi yang luas dalam

berbagai bidang, seperti ilmu komputer, bioinformatika, pengolahan gambar, analisis data, dan lain-lain. Beberapa contoh penggunaan *pattern matching* meliputi:

- Pencarian kata kunci dalam teks atau dokumen.
- Identifikasi pola genetik dalam DNA atau RNA dalam bioinformatika.
- Pencocokan objek pada gambar atau pengenalan wajah.
- Analisis pola dalam data statistik untuk mengidentifikasi tren atau anomali.
- Pencarian pola dalam aliran data untuk deteksi dini kejadian yang penting.

Untuk mencapai pencocokan pola yang efektif, digunakan berbagai algoritma dan metode dalam *pattern matching*, seperti algoritma Brute-Force, algoritma Boyer-Moore, algoritma Knuth-Morris-Pratt, algoritma Rabin-Karp, algoritma Aho-Corasick, dan masih banyak lagi. Pemilihan algoritma yang sesuai tergantung pada sifat pola yang dicari dan kompleksitas teks atau data yang dianalisis.

Dalam praktiknya, *pattern matching* dapat menjadi dasar untuk berbagai aplikasi seperti pencarian teks, pemrosesan bahasa alami, pengenalan pola, dan sistem rekomendasi. Kemampuan untuk mengenali dan menemukan pola dalam teks atau data menjadi kunci untuk mendapatkan wawasan dan informasi yang berharga.

### B. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan pola yang efisien untuk mencari keberadaan suatu pola dalam teks. Algoritma ini dikembangkan oleh Donald Knuth, James H. Morris, dan Vaughan Pratt pada tahun 1977.

Ide dasar di balik algoritma KMP adalah penggunaan informasi yang telah diketahui tentang pola untuk menghindari perbandingan ulang yang tidak perlu saat mencocokkan pola dengan teks. Algoritma ini menggunakan pendekatan yang dikenal sebagai "preprocessing" untuk membuat tabel yang disebut "tabel pergeseran" (shift table) atau "tabel lompatan" (jump table) yang memungkinkan pencocokan yang lebih efisien.

Berikut ilustrasi algoritma KMP:

*Text:* meong meoong - madam eva

*Pattern:* meoong

meong **meoong** - madam eva

1. meoong
2. meoong
3. meoong
4. meoong
5. meoong

Keunggulan utama algoritma KMP terletak pada efisiensinya dalam menghindari perbandingan ulang yang tidak perlu. Dengan menggunakan tabel pergeseran yang telah dibangun secara cerdas, algoritma KMP dapat mencapai kompleksitas waktu  $O(n + m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola.

Skenario terbaik untuk algoritma KMP terjadi ketika pola yang dicari memiliki sedikit atau tidak ada kemunculan karakter yang sama di dalamnya. Dalam hal ini, pencocokan pola dapat diselesaikan dengan efisien dalam satu laluan, dan algoritma KMP akan mencapai kompleksitas waktu terbaiknya, yaitu  $O(n)$ . Keunggulan utama KMP dalam skenario ini adalah efisiensi pencocokan tanpa perlu memeriksa kembali karakter yang sudah diperiksa sebelumnya.

Dengan demikian, algoritma KMP sangat berguna dalam berbagai aplikasi yang melibatkan pencarian pola dalam teks, seperti pemrosesan string, pengenalan pola dalam teks, kompresi data, dan banyak lagi.

### C. Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah sebuah algoritma pencocokan pola yang efisien yang digunakan untuk mencari pola tertentu dalam sebuah teks. Algoritma ini dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977.

Prinsip dasar dari algoritma Boyer-Moore adalah melakukan pencocokan dari akhir ke awal pola yang dicari dalam teks, dan melakukan pergeseran berdasarkan informasi yang diperoleh dari karakter yang tidak cocok.

Algoritma Boyer-Moore didasarkan pada dua teknik, yakni:

1. The looking glass technique, yakni membandingkan karakter pada pattern dan text dimulai dari belakang-ke depan (dimulai dari karakter terakhir pattern)
2. The character-jump technique, yakni jika terjadi mismatch pada karakter  $x$  di text  $T[i]$  dengan karakter  $y$  di pattern  $P[j]$ , maka ada 3 kasus yang dicoba secara berurutan, yakni:
  - a. Jika  $P$  berisi karakter  $x$  di kiri dari lokasi mismatch, maka sejajarkan  $x$  dengan kemunculan terakhir  $x$  pada pattern.
  - b. Jika  $P$  berisi karakter  $x$  hanya di kanan dari lokasi mismatch, maka geser pattern sebanyak 1 karakter ke kanan.
  - c. Jika  $P$  tidak berisi karakter  $x$ , maka sejajarkan  $P[1]$  dengan  $T[i+1]$ .

Berikut ilustrasi algoritma BM:

*Text:* meong meoong - madam eva

*Pattern:* madam eva

meong meoong - **madam eva**

1. madam eva

2.            madam eva
3.                    madam eva

Algoritma Boyer-Moore cenderung lebih efisien pada pola yang panjang. Ketika pola memiliki banyak karakter yang tidak cocok dengan teks, algoritma ini dapat melakukan pergeseran yang besar dan mengurangi perbandingan yang perlu dilakukan. Hal ini membuat algoritma ini cocok untuk mencari pola yang panjang dalam teks yang besar. Kompleksitas waktu terbaik algoritma Boyer-Moore adalah  $O(n/m)$ , yaitu ketika pola tidak cocok dengan teks.

#### D. Algoritma Brute Force Dalam Pattern Matching

Brute force adalah metode sederhana yang secara sistematis memeriksa semua kemungkinan solusi untuk mencapai tujuan tertentu. Dalam konteks komputasi, algoritma brute force mencoba semua kemungkinan langkah atau kombinasi untuk mencari solusi yang diinginkan tanpa menggunakan pengetahuan atau heuristik yang lebih canggih.

Algoritma brute force dalam *pattern matching* adalah metode sederhana yang digunakan untuk mencari pola tertentu (pattern) di dalam sebuah teks atau string. Algoritma ini bekerja dengan membandingkan pola dengan setiap kemungkinan substring di dalam teks secara berurutan.

Berikut ilustrasi algoritma KMP:

*Text:* meong meoong - madam eva

*Pattern:* meoong

meong **meoong** - madam eva

1. meoong
2. meoong
3. meoong
4. meoong
5. meoong
6. meoong
7. meoong

Algoritma brute force ini bekerja dengan cara mencoba semua kemungkinan substring dalam teks yang sesuai dengan panjang pola. Ini berarti kompleksitas waktu algoritma ini adalah  $O(m * n)$ , di mana  $m$  adalah panjang pola dan  $n$  adalah panjang teks. Dalam kasus terburuk, algoritma ini harus membandingkan pola dengan setiap karakter dalam teks.

#### E. Citra Digital

Citra adalah gambar pada suatu bidang dua dimensi. Agar sebuah citra dapat diolah dengan komputer digital, maka suatu

citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Representasi citra dengan fungsi kontinu menjadi nilai-nilai diskrit disebut digitalisasi citra. Citra yang dihasilkan dari proses digitalisasi citra tersebut adalah citra digital. Pada umumnya citra digital berbentuk persegi panjang dengan tinggi dan lebarnya merupakan dimensi ukuran atau resolusi dari citra.[2]

Setiap elemen pada citra digital disebut dengan *image element*, *picture element*, *pel*, atau *pixel*. Citra dengan resolusi  $m \times n$  memiliki  $mn$  buah *pixel*. Setiap *pixel* direpresentasikan dengan bilangan biner  $n$ -bit. Citra digital diklasifikasikan menjadi beberapa kategori berdasarkan jumlah bit representasi *pixel*-nya:

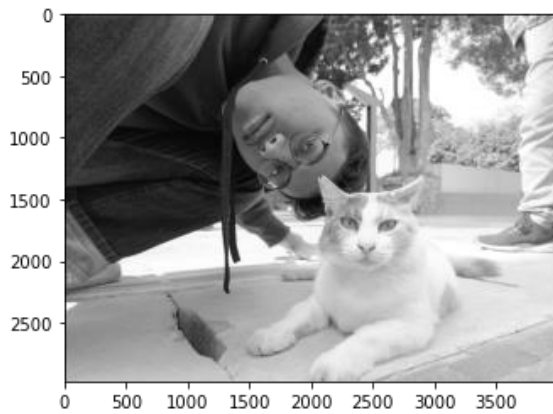
- Citra 1-bit  
Citra 1-bit biasa disebut citra biner. Pada kategori ini, citra digital direpresentasikan oleh satu bit saja sehingga citra biner hanya memiliki dua kemungkinan nilai *pixel*. *Pixel* bernilai 0 mempresentasikan warna hitam sedangkan *pixel* bernilai 1 atau 255 mempresentasikan warna putih.
- Citra 8-bit  
Citra 8-bit disebut juga dengan citra *grayscale*. Setiap *pixel* pada citra *grayscale* direpresentasikan oleh sebuah kanal bilangan 0 sampai 255 yang menyatakan nilai keabuan. *Pixel* dengan nilai 0 memiliki nilai keabuan paling gelap sedangkan *pixel* dengan nilai 255 memiliki nilai keabuan paling terang.  
Citra *grayscale* dapat dikonversi menjadi citra biner dengan cara mengganti nilai *pixel* yang bernilai kurang dari 127 menjadi 0 dan mengganti nilai *pixel* yang bernilai lebih dari 127 menjadi 255.
- Citra 24-bit  
Dibandingkan dengan citra 1-bit dan 8-bit yang hanya berwarna hitam, putih, atau abu-abu, citra 24-bit memiliki warna yang beragam. Oleh karena itu, citra 24-bit disebut juga citra berwarna. Warna pada citra 24-bit dimungkinkan karena terdapat 3 kanal bernilai 8-bit pada setiap *pixel*-nya. Ketiga kanal tersebut masing-masing mempresentasikan konsentrasi warna *Red*, *Green*, dan *Blue*. Besar konsentrasi warna untuk setiap kanal *pixel* adalah 0 sampai 255.  
Citra berwarna dapat dikonversikan menjadi citra *grayscale* dengan rumus pada persamaan 1:

$$x = 0.299R + 0.587G + 0.114B \quad (1)$$

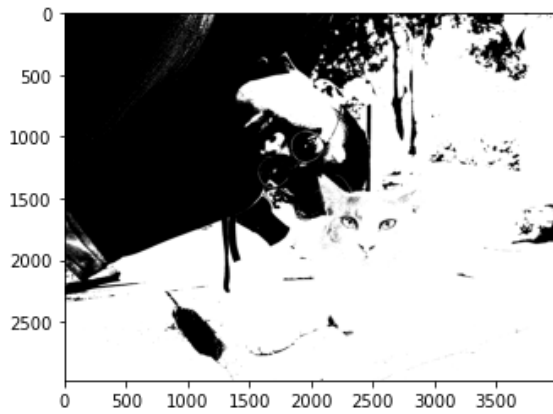
Dengan  $x$  merupakan nilai *pixel* citra 8-bit hasil konversi dan  $R$ ,  $G$ , dan  $B$  masing-masing merupakan konsentrasi warna *Red*, *Green*, dan *Blue* citra 24-bit. Citra pada gambar 2.2. merupakan konversi citra *grayscale* 8-bit dari citra berwarna 24-bit pada gambar 2.1. sedangkan gambar 2.3. merupakan konversi citra biner 1-bit dari citra *grayscale* 8-bit pada gambar 2.2.



Gambar 2.1. Citra 24-bit "Icad Sama Eva"  
(Sumber: Dokumen Pribadi)



Gambar 2.2. Citra 8-bit



Gambar 2.3. Citra 1-bit

#### F. Template Matching

*Template matching* adalah metode pengolahan citra yang digunakan untuk mencocokkan atau mengidentifikasi kemunculan sebuah template atau pola yang telah ditentukan di dalam citra yang lebih besar. Tujuan utama dari template matching adalah untuk menemukan posisi dan/atau keberadaan suatu objek dalam citra berdasarkan kesamaannya dengan template yang diinginkan.

Proses template matching dimulai dengan memiliki sebuah template, yang biasanya berupa citra kecil yang mewakili objek

yang ingin ditemukan. Kemudian, template tersebut akan dicocokkan dengan citra yang lebih besar secara berulang-ulang untuk mencari lokasi di mana kemunculan template tersebut paling cocok atau paling mirip.

Template matching dapat digunakan untuk berbagai aplikasi, seperti deteksi wajah, pengenalan karakter tulisan tangan, identifikasi objek dalam citra medis, dan banyak lagi. Meskipun template matching sederhana dan mudah diimplementasikan, namun kemampuannya terbatas jika terdapat perbedaan skala, rotasi, atau deformasi yang signifikan pada objek yang ingin ditemukan.

### III. DESKRIPSI MASALAH

Pada makalah ini, penulis melakukan pemrosesan citra metode *template matching* dengan mengimplementasikan proses *pattern matching* algoritma *Brute Force*, *Knuth-Morris-Pratt*, dan *Boyer-Moore*. Proses pencocokan pola menggunakan *pixel* pada citra untuk membandingkan citra yang digunakan sebagai teks (gambar 3.1) dengan potongan citra yang digunakan sebagai pola (gambar 3.2).

Untuk setiap algoritma *pattern matching* yang dilakukan, kemudian dihitung banyaknya perbandingan *pixel* yang dilakukan serta waktu yang dibutuhkan untuk mencocokkan citra dengan potongan citra. Dihitung juga banyak perbandingan *pixel* dan waktu yang dibutuhkan pada kemungkinan terburuk, yaitu untuk mencocokkan citra dengan potongan citra yang tidak cocok dengan citra utama (gambar 3.3).



Gambar 3.1. Citra Utama "H-1 UAS"  
(Sumber: Dokumentasi Pribadi)



Gambar 3.2.  
Potongan Citra  
"Indra"



Gambar 3.3.  
Potongan Citra  
"Pacar Penulis"

Setelah itu, dibangkitkan 10 potongan citra acak dari citra utama (gambar 3.1) untuk dilakukan analisis performa algoritma *Brute Force*, *Knuth-Morris-Part*, dan *Boyer-Moore* dengan



membandingkan lamanya waktu eksekusi serta banyaknya perbandingan *pixel* yang dilakukan pada setiap algoritma.

#### IV. IMPLEMENTASI DAN PENGUJIAN

Kode dalam bahasa Python berikut ini merupakan algoritma utama dalam proses *template matching* yang telah dideskripsikan diatas:

```
def templateMatch(template, image, algo):
    # preprocessing
    # ubah citra dan template menjadi 8-bit
    image = cv2.cvtColor(image,
                          cv2.COLOR_RGB2GRAY)
    template = cv2.cvtColor(template,
                              cv2.COLOR_RGB2GRAY)
    tempRow = len(template)
    tempCol = len(template[0])
    pattern = template[0].flatten().tolist()
    n = len(image)
    compareCounter = 0

    # memanggil algoritma pattern matching
    for i in range(n):
        text = image[i].flatten().tolist()
        if algo == 'kmp':
            found, cnt =
                KMPSearch(pattern, text)
        elif algo == 'bm':
            found, cnt =
                BMSearch(pattern, text)
        else:
            found, cnt =
                BFSearch(pattern, text)
        compareCounter += cnt

    # mengecek validitas template
    for j in found:
        if (np.array_equal(template,
                            image[i:i+tempRow,
                                j:j+tempCol])):
            return i, j, compareCounter

    return -1, -1, compareCounter
```

Fungsi *templateMatch* memiliki parameter *template* yaitu potongan citra yang ingin dicocokkan, *image* yaitu citra utama, serta *algo* yaitu algoritma yang hendak digunakan. Fungsi tersebut mengembalikan tiga nilai dengan dua nilai pertama adalah indeks dimana potongan citra ditemukan dan banyaknya perbandingan *pixel* yang dilakukan pada pemanggilan fungsi.

Berikut merupakan hasil *template matching* untuk potongan citra pada gambar 3.2 dan gambar 3.3 dengan citra utama 3.1:

Tabel 4.1. Hasil Template Matching Gambar 3.2

Algoritma	Indeks	Waktu Eksekusi (ms)	Banyak Perbandingan
Brute Force	2000, 950	1159.99	3543720
Knuth-Morris-Pratt	2000, 950	2486.06	3539769
Boyer-Moore	2000, 950	183.98	12310

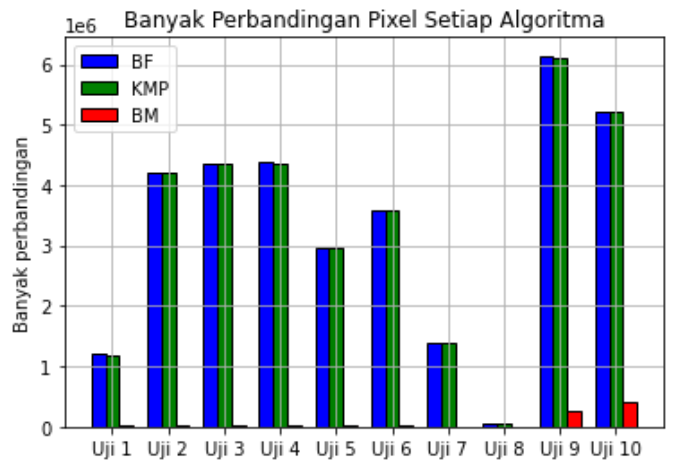


Gambar 4.1. Hasil Template Matching Gambar 3.2

Tabel 4.2. Hasil Template Matching Gambar 3.3

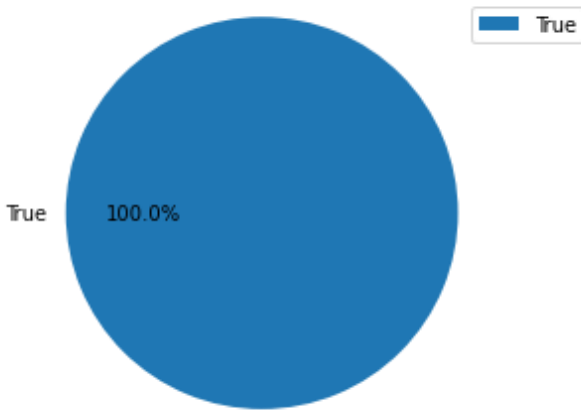
Algoritma	Indeks	Waktu Eksekusi (ms)	Banyak Perbandingan
Brute Force	-1, -1	2255.98	7161446
Knuth-Morris-Pratt	-1, -1	5940.01	7132608
Boyer-Moore	-1, -1	504.99	27437

Selanjutnya dibangkitkan 10 potongan citra secara acak untuk dilakukan analisis perbandingan kinerja setiap algoritma *pattern matching*. Berikut merupakan hasil perbandingan kinerja masing-masing algoritma tersebut:



Gambar 4.3. Grafik Banyak Perbandingan Pixel Setiap Algoritma

Akurasi Template Matching



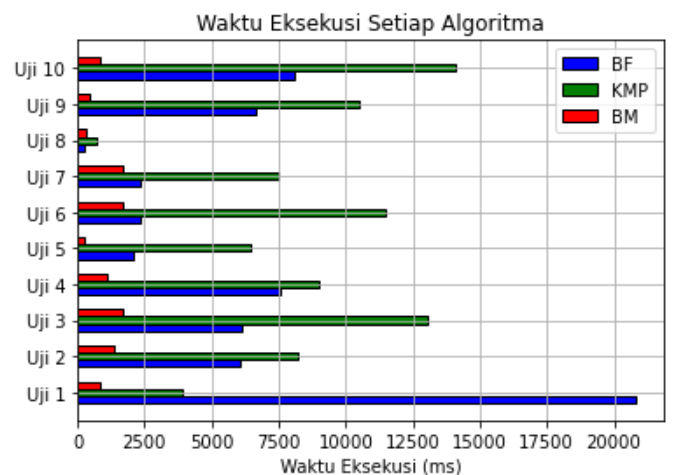
Gambar 4.2. Grafik Akurasi Template Matching Seluruh Algoritma

Tabel 4.4. Waktu Eksekusi Setiap Algoritma

No Uji	Ukuran Potongan Citra	Waktu Eksekusi (ms)		
		Brute Force	Knuth-Morris-Pratt	Boyer-Moore
1	136940	20816.65	3906.04	803.13
2	835031	6036.04	8196.79	1339.53
3	24030	6141.88	13012.82	1690.73
4	40922	6141.88	13012.82	1690.73
5	421872	2089.15	6442.80	235.99
6	331128	2324.00	11499.64	1668.34
7	1485306	2342.96	7475.67	1683.20
8	1340874	265.04	711.57	310.00
9	918	6672.87	10528.43	469.00
10	10361	8113.01	14073.02	806.15

Tabel 4.3. Banyak Perbandingan Pixel Setiap Algoritma

No Uji	Ukuran Potongan Citra	Banyak Perbandingan Pixel		
		Brute Force	Knuth-Morris-Pratt	Boyer-Moore
1	136940	1199472	1177416	9427
2	835031	4216102	4208024	24010
3	24030	4352372	4344819	32659
4	40922	4387948	4361979	29081
5	421872	2971693	2950695	22838
6	331128	3578741	3570299	12609
7	1485306	1395389	1393167	6279
8	1340874	56048	53004	1118
9	918	6142190	6104966	260758
10	10361	5221855	5211360	403929



Gambar 4.4. Grafik Waktu Eksekusi Setiap Algoritma

## V. ANALISIS

Berdasarkan implementasi dan pengujian *template matching* menggunakan proses *pattern matching* algoritma *Brute Force*, algoritma Knuth-Morris-Pratt, dan algoritma Boyer-Moore yang telah dibuat dengan bahasa Python, terdapat beberapa poin yang hendak dianalisis, diantaranya:

### 1. Akurasi Hasil Pencocokan.

Berdasarkan tabel 4.1. dan tabel 4.2. serta gambar 4.2 terlihat bahwa proses *pattern matching* algoritma *Brute Force*, algoritma Knuth-Morris-Pratt, dan algoritma Boyer-Moore ketiganya memberikan hasil pencocokan yang 100% akurat dalam pengolahan citra *template matching*. Ketiga algoritma tersebut juga dapat mendeteksi apabila sebuah pola tidak ditemukan di dalam citra.

### 2. Banyak Perbandingan *Pixel*.

Berdasarkan tabel 4.3. dapat dihitung rata-rata banyak proses perbandingan *pixel* sebagai berikut:

Tabel 5.1. Rata-Rata Banyak Perbandingan

Algoritma	Rata-Rata Banyak Perbandingan
<i>Brute Force</i>	3352181.0
Knuth-Morris-Pratt	3337572.9
Boyer-Moore	80270.8

Terlihat bahwa algoritma Boyer-Moore jauh lebih efektif dalam melakukan perbandingan *pixel* dibandingkan algoritma Knuth-Morris-Pratt dan *Brute Force*. Algoritma BM cenderung efektif pada *template matching* pada citra dikarenakan pola yang digunakan untuk *pattern matching* berukuran panjang serta variasi pembandingan *pixel* yang besar yaitu pada jangkauan 0 hingga 256. Sehingga algoritma Boyer-Moore dapat melakukan pergeseran yang besar serta mengurangi perbandingan yang tidak diperlukan.

Sebaliknya, algoritma Knuth-Morris-Pratt terlihat kurang efektif dan cenderung melakukan perbandingan sebanyak algoritma *Brute Force*. Seperti yang telah disebutkan, variasi pembandingan *pixel* yang besar serta ukuran pola yang panjang justru menyebabkan tabel pergeseran algoritma KMP kurang berguna.

### 3. Waktu Eksekusi.

Berdasarkan tabel 4.4. dapat dihitung rata-rata waktu eksekusi sebagai berikut:

Tabel 5.2. Rata-Rata Waktu Eksekusi

Algoritma	Rata-Rata Waktu Eksekusi (ms)
<i>Brute Force</i>	6234.6284
Knuth-Morris-Pratt	8485.9668

Boyer-Moore	1010.9879
-------------	-----------

Seperti pada aspek banyak perbandingan, algoritma Boyer-Moore terlihat lebih efektif daripada algoritma *Brute Force* dan Knuth-Morris-Pratt dalam aspek waktu eksekusi. Menariknya, Algoritma Knuth-Morris-Pratt yang lebih efektif daripada algoritma *Brute Force* dalam melakukan perbandingan, malah memakan waktu yang cenderung lebih lama dalam melakukan eksekusi program. Lamanya waktu eksekusi algoritma KMP bisa dijelaskan oleh preprocessing algoritma KMP itu sendiri yang perlu menghitung tabel pergeseran setiap melakukan *pattern matching*. Panjangnya pola pada citra dan pemrosesan tabel pergeseran menyebabkan algoritma KMP memiliki rata-rata waktu eksekusi yang lebih lama daripada algoritma *Brute Force*.

### 4. Batasan *Pattern Matching* dalam *Template Matching*.

Dalam proses implementasi, penulis menemukan beberapa batasan dalam penggunaan proses *pattern matching* untuk pengolahan citra *template matching*. Proses *pattern matching* dapat dengan mudah melakukan pencarian *exact match* (sama persis) sebuah pola pada data. Masalah terjadi ketika pola tidak benar-benar sama persis dengan potongan dari data utama.

Dalam implementasi *pattern matching* pada gambar, sering terjadi kesalahan ketika citra yang hendak digunakan sebagai pattern tidak memiliki informasi *pixel* yang sama dengan citra utama, menyebabkan *template matching* tidak menemukan solusi karena algoritma hanya mencari potongan data yang sama persis. Hal ini kadang disebabkan oleh format penyimpanan ketika memotong citra merupakan penyimpanan *lossy* atau bersifat mengurangi kualitas seperti format gambar JPG.

## VI. SIMPULAN

*Pattern matching* adalah proses mencari kemunculan atau pencocokan dari suatu pola (pattern) dalam suatu teks atau data. Beberapa algoritma yang dapat digunakan dalam *pattern matching* yaitu algoritma *Brute Force*, algoritma Knuth-Morris-Pratt, dan algoritma Boyer-Moore. Proses *pattern matching* dapat diimplementasikan dalam pengolahan citra *template matching*. *Template matching* merupakan metode pengolahan citra yang digunakan untuk mencocokkan atau mengidentifikasi kemunculan sebuah template atau pola yang telah ditentukan di dalam citra yang lebih besar.

Dalam pengolahan citra *template matching* menggunakan proses *pattern matching*, algoritma *Brute Force*, algoritma Knuth-Morris-Pratt, dan algoritma Boyer-Moore ketiganya memberikan solusi yang akurat. Selain itu, dapat disimpulkan algoritma Boyer-Moore merupakan algoritma yang paling efisien dalam aspek banyaknya proses perbandingan *pixel* maupun lama waktu eksekusi dibandingkan algoritma *Brute Force* dan algoritma Knuth-Morris-Pratt. Algoritma KMP dirasa kurang efektif dalam proses ini dikarenakan panjangnya pola dan banyaknya variasi data.

Terdapat batasan *template matching* menggunakan proses *pattern matching* yaitu hanya dapat melakukan pencocokkan pada pola yang sama persis dengan potongan citra utama. Masalah kadang terjadi ketika *template* disimpan dengan format penyimpanan yang *lossy*.

#### LINK REPOSITORY GITHUB

Kode program yang digunakan dalam makalah ini dapat dilihat di link berikut:

<https://github.com/bagas003/Stima-TemplateMatching>

#### UCAPAN TERIMA KASIH

Puji syukur terhadap Tuhan Yang Maha Esa karena atas rahmat-Nya, makalah berjudul “Analisis Perbandingan Algoritma Pattern Matching Knuth-Morris-Pratt dan Boyer-Moore Untuk Metode Pemrosesan Citra Template Matching” ini dapat terselesaikan dengan baik. Terimakasih juga disampaikan kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma, yakni Bapak Rinaldi Munir atas bimbingannya selama perkuliahan. Penulis juga berterimakasih kepada teman-teman penulis yaitu Indra, Topa, Ridha, dan Icad yang bersedia fotonya penulis masukkan dalam makalah ini.

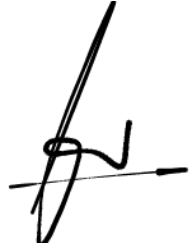
#### REFERENCES

- [1] Munir, Rinaldi. 2009. Diktat IF2211 Strategi Algoritma
- [2] Hidayatullah, Priyanto. 2017. Pengolahan Citra Digital Teori dan Aplikasi Nyata. Bandung: Penerbit Informatika.

#### PERNYATAAN

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

Bandung, 22 Mei 2023



Bagas Aryo Seto 13521081