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

Oleh Bagas Aryo Seto - 13521081

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 BoyerMoore. 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.

Import Modules

```
# import modules
import cv2
import numpy as np
import matplotlib.pyplot as plt
import copy
import pandas as pd
import random
import seaborn as sns
import time
```

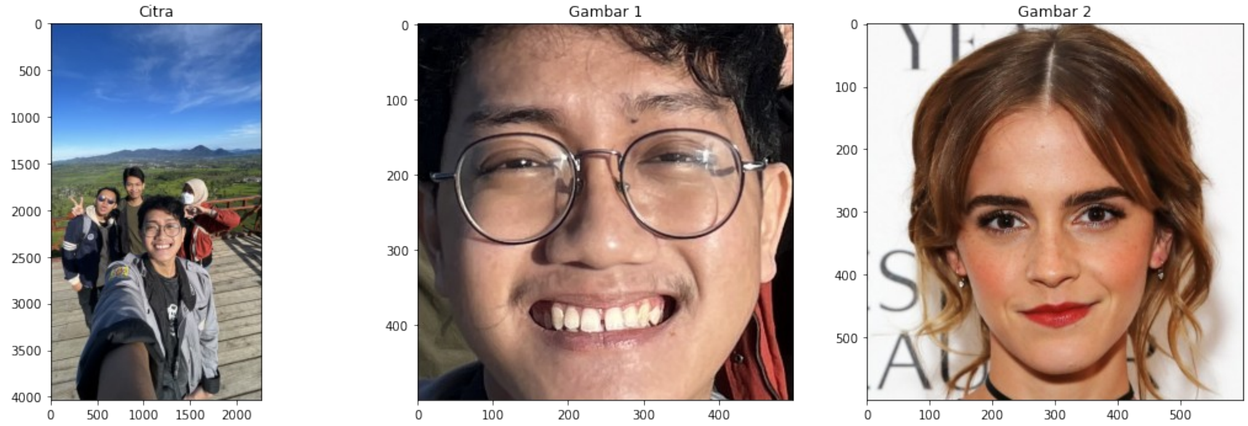
Definisikan Knuth-Morris-Pratt Search, Boyer-Moore Search, dan Brute Force Search

```
def KMPSearch(pattern, txt):
    patternLen = len(pattern)
    textLen = len(txt)

    compareCounter = 0
    results = []

    # create lps[] that will hold the longest prefix suffix
    # values for pattern
    lps = [0]*patternLen
    patternIdx = 0 # index for pat[]

    # Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pattern, patternLen, lps)

    textIdx = 0 # index for txt[]
    while (textLen - textIdx) >= (patternLen - patternIdx):
        compareCounter += 1

        if pattern[patternIdx] == txt[textIdx]:
            textIdx += 1
            patternIdx += 1

        if patternIdx == patternLen:
            results += [textIdx - patternIdx]
            patternIdx = lps[patternIdx - 1]

        # mismatch after j matches
        elif textIdx < textLen and pattern[patternIdx] !=
txt[textIdx]:

            if patternIdx != 0:
                patternIdx = lps[patternIdx-1]
            else:
                textIdx += 1

    return results, compareCounter


# Function to compute LPS array
def computeLPSArray(pat, plen, lps):
    len = 0 # length of the previous longest prefix suffix
    lps[0] = 0
    i = 1

    while i < plen:
        if pat[i] == pat[len]:
            len += 1
            lps[i] = len
            i += 1
        else:
            if len != 0:
                len = lps[len-1]
            else:
                lps[i] = 0
                i += 1

def BMSearch(pattern, text):
    # Preprocessing
    patternLen = len(pattern)
    textLen = len(text)

    compareCounter = 0

    last = {}
    for i in range(patternLen):
        last[pattern[i]] = i

    # Searching
    i = patternLen - 1  # Index of the last character in the pattern
    j = patternLen - 1  # Index of the last character in the text
being examined
    results = []

    while j < textLen:
        compareCounter += 1
        if pattern[i] == text[j]:
            if i == 0:
                results.append(j)
                i = patternLen - 1
                j += patternLen
            else:
                i -= 1
                j -= 1
        else:
            if text[j] in last:
                j += patternLen - min(i, last[text[j]] + 1)
            else:
                j += patternLen
            i = patternLen - 1

    return results, compareCounter


def BFSearch(pattern, text):
    # Preprocessing
    patternLen = len(pattern)
    textLen = len(text)

    compareCounter = 0

    # Searching
    results = []

    for i in range(textLen - patternLen + 1):
        j = 0
        compareCounter += 1
        while j < patternLen and pattern[j] == text[i + j]:
            compareCounter += 1
            j += 1

        if j == patternLen:
            results.append(i)

    return results, compareCounter



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 yang sesuai
    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 pada citra
        for j in found:
            if (np.array_equal(template, image[i:i+tempRow,
j:j+tempCol])):
                return i, j, compareCounter

    return -1, -1, compareCounter
```

Membaca dan Menampilkan Citra dan Template

```
# Membaca citra dan template
image = cv2.imread(r'images\image.png')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

template1 = cv2.imread(r'images\template.png')
template1 = cv2.cvtColor(template1, cv2.COLOR_BGR2RGB)
```

```python
template2 = cv2.imread(r'images\emma_watson.png')
template2 = cv2.cvtColor(template2, cv2.COLOR_BGR2RGB)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Display the first image on the first subplot
axes[0].imshow(image)
axes[0].set_title('Citra')

# Display the second image on the second subplot
axes[1].imshow(template1)
axes[1].set_title('Gambar 1')

# Display the third image on the third subplot
axes[2].imshow(template2)
axes[2].set_title('Gambar 2')

# Adjust the spacing between subplots
plt.tight_layout()

# Show the figure
plt.show()
```



Citra

Gambar 1

Gambar 2

Mendapatkan Hasil Template Matching Menggunakan KMP, BM, dan Brute Force Pada Gambar 1

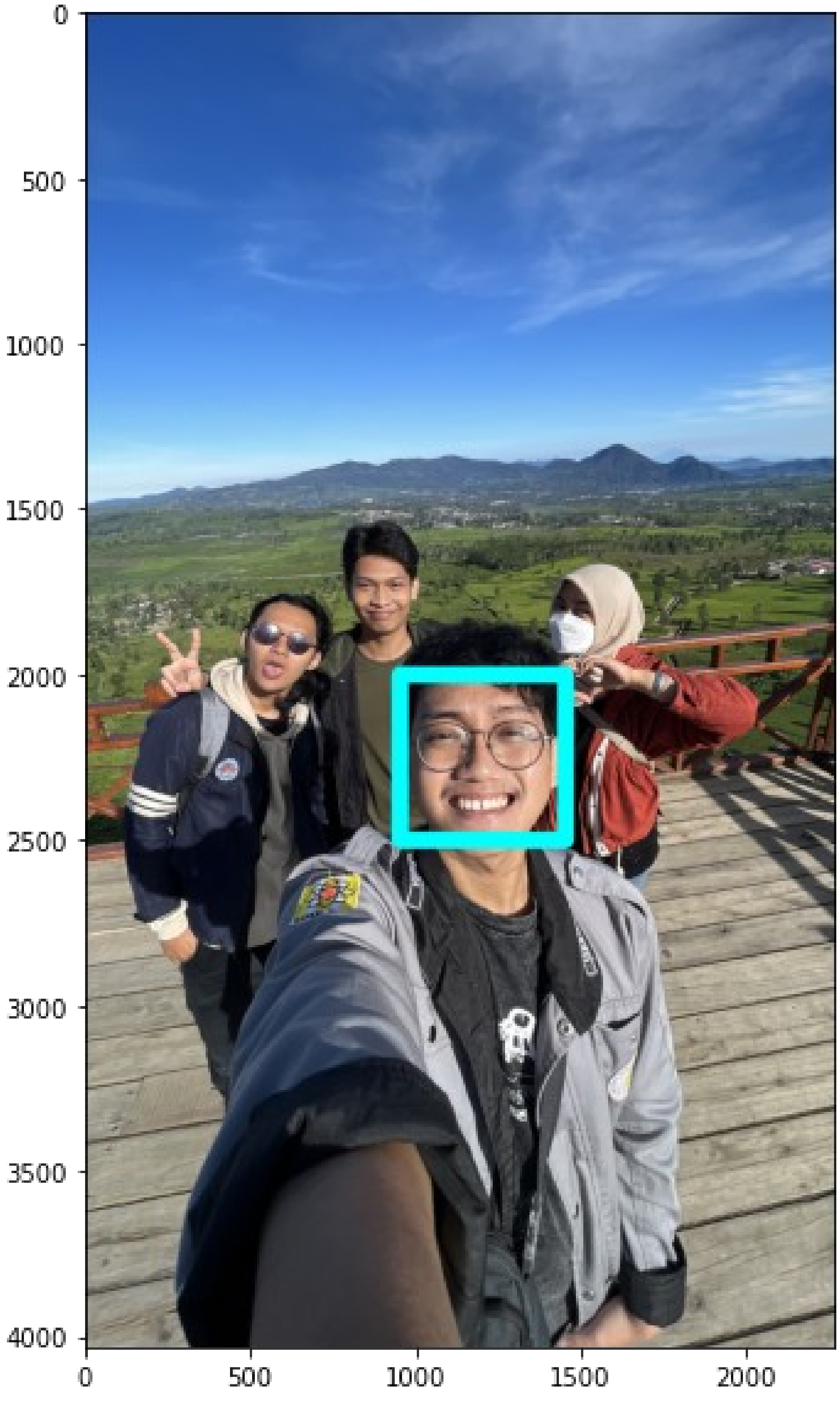
```python
t1 = time.time()
i3, j3, bfCount  = templateMatch(template1, image, 'bf')
t2 = time.time()
i1, j1, kmpCount = templateMatch(template1, image, 'kmp')
t3 = time.time()
i2, j2, bmCount  = templateMatch(template1, image, 'bm')
t4 = time.time()

bfTime  = t2 - t1
kmpTime = t3 - t2
```

```
bmTime  = t4 - t3

print(f"Ketiga hasil cocok : {(i1,j1) == (i2,j2) == (i3,j3)}")
print(f"Koordinat Gambar 1 : ({i1}, {j1})\n")
print(f"Waktu Brute Force        : {bfTime}\ts")
print(f"Waktu Knuth-Morris-Pratt : {kmpTime}\ts")
print(f"Waktu Boyer-Moore        : {bmTime}\ts\n")
print(f"Banyak Perbandingan Brute Force        : {bfCount}")
print(f"Banyak Perbandingan Knuth-Morris-Pratt : {kmpCount}")
print(f"Banyak Perbandingan Boyer-Moore        : {bmCount}")
```

```
Ketiga hasil cocok : True
Koordinat Gambar 1 : (2000, 950)

Waktu Brute Force        : 1.9515295028686523   s
Waktu Knuth-Morris-Pratt : 2.637633800506592    s
Waktu Boyer-Moore        : 0.21509647369384766  s

Banyak Perbandingan Brute Force        : 3543720
Banyak Perbandingan Knuth-Morris-Pratt : 3539769
Banyak Perbandingan Boyer-Moore        : 12310
```

Menampilkan Hasil Template Matching Pada Gambar 1

```
result = copy.copy(image)
w, h = template1.shape[:-1]

if ((i1, j1) != (-1, -1)):
    cv2.rectangle(result, (j1, i1), (j1 + h, i1 + w), (0, 255, 255), 50)

plt.figure(figsize=(10, 10))
plt.imshow(result)
plt.show()
```


0
500
1000
1500
2000
2500
3000
3500
4000
0
500
1000
1500
2000

Mendapatkan Hasil Template Matching Menggunakan KMP, BM, dan Brute Force Pada Gambar 2

```
t5 = time.time()
i4, j4, bfCount2  = templateMatch(template2, image, 'bf')
t6 = time.time()
i5, j5, kmpCount2 = templateMatch(template2, image, 'kmp')
t7 = time.time()
i6, j6, bmCount2  = templateMatch(template2, image, 'bm')
t8 = time.time()

bfTime2  = t6 - t5
kmpTime2 = t7 - t6
bmTime2  = t8 - t7

print(f"Ketiga hasil cocok : {(i4,j4) == (i5,j5) == (i6,j6)}")
print(f"Koordinat Gambar 2 : ({i4}, {j4})\n")
print(f"Waktu Brute Force        : {bfTime2}\ts")
print(f"Waktu Knuth-Morris-Pratt : {kmpTime2}\ts")
print(f"Waktu Boyer-Moore        : {bmTime2}\ts\n")
print(f"Banyak Perbandingan Brute Force        : {bfCount2}")
print(f"Banyak Perbandingan Knuth-Morris-Pratt : {kmpCount2}")
print(f"Banyak Perbandingan Boyer-Moore        : {bmCount2}")
```

```
Ketiga hasil cocok : True
Koordinat Gambar 2 : (-1, -1)

Waktu Brute Force        : 2.214761972427368     s
Waktu Knuth-Morris-Pratt : 5.476287364959717     s
Waktu Boyer-Moore        : 0.8662610054016113    s

Banyak Perbandingan Brute Force        : 6731432
Banyak Perbandingan Knuth-Morris-Pratt : 6729497
Banyak Perbandingan Boyer-Moore        : 15729
```

Menampilkan Hasil Template Matching Pada Gambar 2

```
result = copy.copy(image)
w, h = template2.shape[:-1]

if ((i4, j4) != (-1, -1)):
    cv2.rectangle(result, (j4, i4), (j4 + h, i4 + w), (0, 255, 255), 50)

plt.figure(figsize=(10, 10))
plt.imshow(result)
plt.show()
```

0
500
1000
1500
2000
2500
3000
3500
4000
0
500
1000
1500
2000

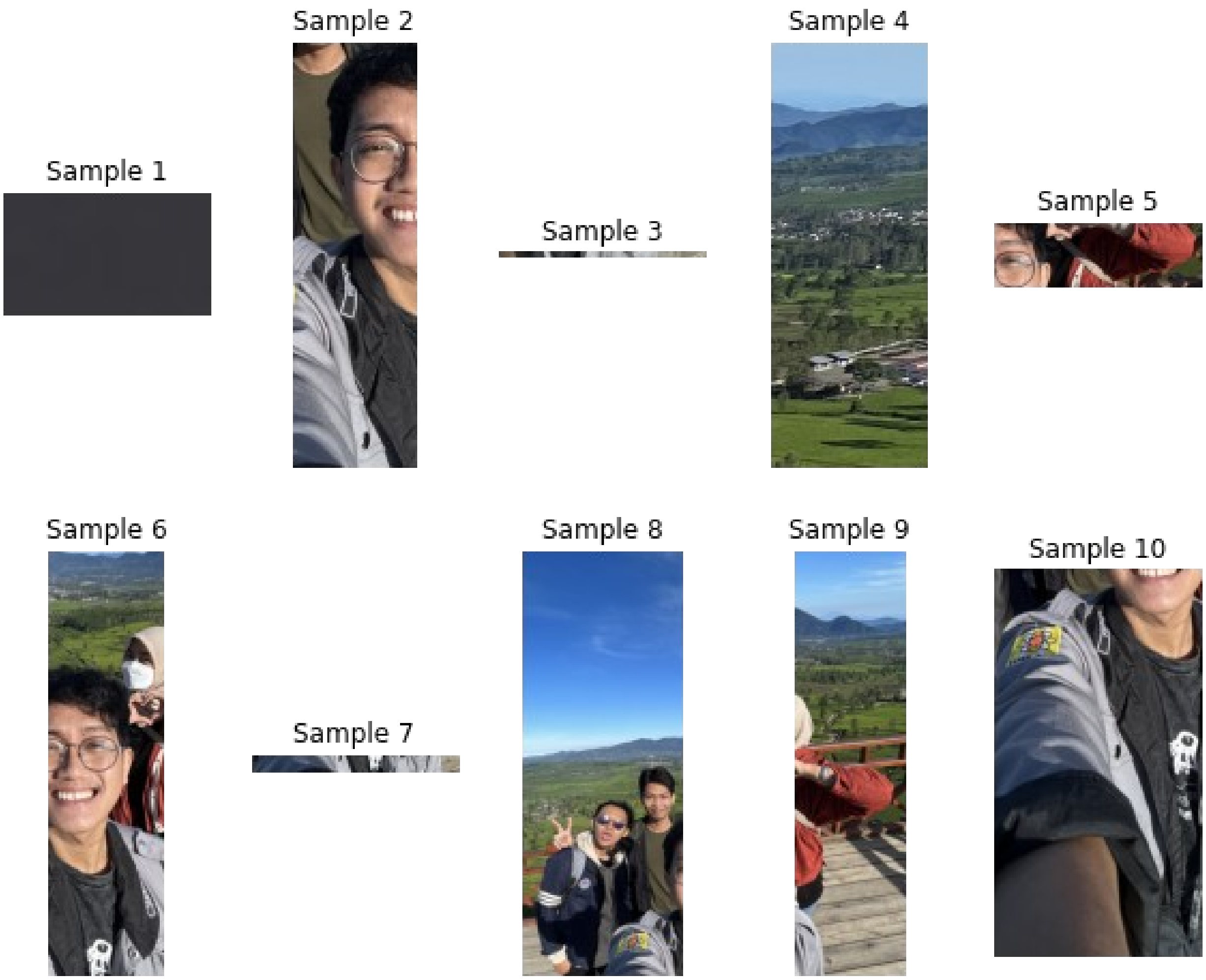
Generate 10 Random Sampel Lalu Ambil Hasil Template Matching Pada Sampel

```
data = {
    'size': [],
    'BF': [],
    'KMP': [],
    'BM': [],
    'BF time (ms)': [],
    'KMP time (ms)': [],
    'BM time (ms)': [],
    'accurate': [],
    'row': [],
    'col': [],
    'width': [],
    'height': []
}

w, h = image.shape[:-1]


for i in range(10):
    r1 = random.randint(0, w-10)
    r2 = random.randint(r1+1, w-1)
    c1 = random.randint(0, h-10)
    c2 = random.randint(c1+1, h-1)

    data['size'].append((r2-r1) * (c2-c1))
    temp = image[r1:r2, c1:c2]

    t1 = time.time()
    i1, j1, bfCount  = templateMatch(temp, image, 'bf')
    t2 = time.time()
    i2, j2, kmpCount = templateMatch(temp, image, 'kmp')
    t3 = time.time()
    i3, j3, bmCount  = templateMatch(temp, image, 'bm')
    t4 = time.time()

    data['BF'].append(bfCount)
    data['BF time (ms)'].append((t2 - t1)*1000)
    data['KMP'].append(kmpCount)
    data['KMP time (ms)'].append((t3 - t2)*1000)
    data['BM'].append(bmCount)
    data['BM time (ms)'].append((t4 - t3)*1000)
    data['accurate'].append(r1 == i1 == i2 == i3 and c1 == j1 == j2 ==
j3)
    data['row'].append(r1)
    data['col'].append(c1)
    data['width'].append(r2-r1)
    data['height'].append(c2-c1)
```

```
df = pd.DataFrame(data)
df

      size       BF      KMP      BM  BF time (ms)  KMP time (ms)  \
0      756  8572875  8495186  254418   3410.575390    5075.389385
1   498892  3518453  3517447   13933   1132.727146    2257.345438
2    34380  3876302  3864823   31865   1296.786070    4106.243134
3   101904  2671851  2645565   28832   1046.411037    1615.803242
4   222865  2923123  2885193   17654   1275.613546    2228.004456
5   898586  2418976  2389150   53548    819.306374    1633.229733
6   205226  2366499  2356548   13676    907.504559    2839.982986
7  2805720   219528   197499    1813    103.229284     317.312241
8   881973  1915169  1899815    5141   1106.501102    2198.771715
9  1204800  3517851  3484468   28418   1454.519033    2907.816172

   BM time (ms)  accurate   row   col  width  height
0    236.806154      True  3799   735     21      36
1    167.150259      True  1863   823   1306     382
2   1072.239637      True  3439   920     30    1146
3     96.963167      True  1272  2013    528     193
4    299.212217      True  2019  1220    265     841
5    167.706013      True  1345  1086   1819     494
6    702.826023      True  3054   415    137    1498
7     92.001915      True   156    29   2724    1030
8    191.809654      True  1059  1610   1849     477
9    433.033943      True  2371   574   1506     800

df.describe()

               size            BF           KMP             BM  BF
time (ms)  \
count  1.000000e+01  1.000000e+01  1.000000e+01      10.000000
10.000000
mean   6.855102e+05  3.200063e+06  3.173569e+06   44929.800000
1255.317354
std    8.538765e+05  2.153084e+06  2.137927e+06   75119.035131
843.679298
min    7.560000e+02  2.195280e+05  1.974990e+05    1813.000000
103.229284
25%    1.277345e+05  2.379618e+06  2.364698e+06   13740.250000
942.231178
50%    3.608785e+05  2.797487e+06  2.765379e+06   23036.000000
1119.614124
75%    8.944328e+05  3.518302e+06  3.509202e+06   31106.750000
1291.492939
max    2.805720e+06  8.572875e+06  8.495186e+06  254418.000000
3410.575390

       KMP time (ms)  BM time (ms)          row          col
width  \
count      10.000000     10.000000    10.000000    10.00000
10.000000
mean     2517.989850    345.974898  2037.700000   942.50000
1018.500000
std      1333.838877    314.558089  1145.300645   576.91695
949.684538
min       317.312241     92.001915   156.000000    29.00000
21.000000
25%      1774.615228    167.289197  1290.250000   614.25000
169.000000
50%      2242.674947    214.307904  1941.000000   871.50000
917.000000
75%      2890.857875    399.578512  2883.250000  1186.50000
1740.750000
max      5075.389385   1072.239637  3799.000000  2013.00000
2724.000000

            height
count    10.000000
mean    689.700000
std     455.176901
min      36.000000
25%     405.750000
50%     647.000000
75%     982.750000
max    1498.000000
```

Menampilkan Sampel

```python
image_samples = []

for i in range(10):
    r, c = df.at[i, 'row'], df.at[i, 'col']
    w, h = df.at[i, 'width'], df.at[i, 'height']

    image_samples.append(image[r:r+w+1, c:c+h+1])

fig, axes = plt.subplots(2, 5, figsize=(10, 8))
axes = axes.flatten()

for i, (sample, ax) in enumerate(zip(image_samples, axes)):
    ax.imshow(sample, cmap='gray')
    ax.set_title(f"Sample {i+1}")
    ax.axis('off')
```



Menampilkan Hasil Template Matching Sampel

```
result = copy.copy(image)

for i in range(10):
    r, c = df.at[i, 'row'], df.at[i, 'col']
    w, h = df.at[i, 'width'], df.at[i, 'height']
    if ((r, c) != (-1, -1)):
        cv2.rectangle(result, (c, r), (c + h, r + w), (0, 255, 255),
25)

plt.figure(figsize=(10, 10))
plt.imshow(result)
plt.show()
```

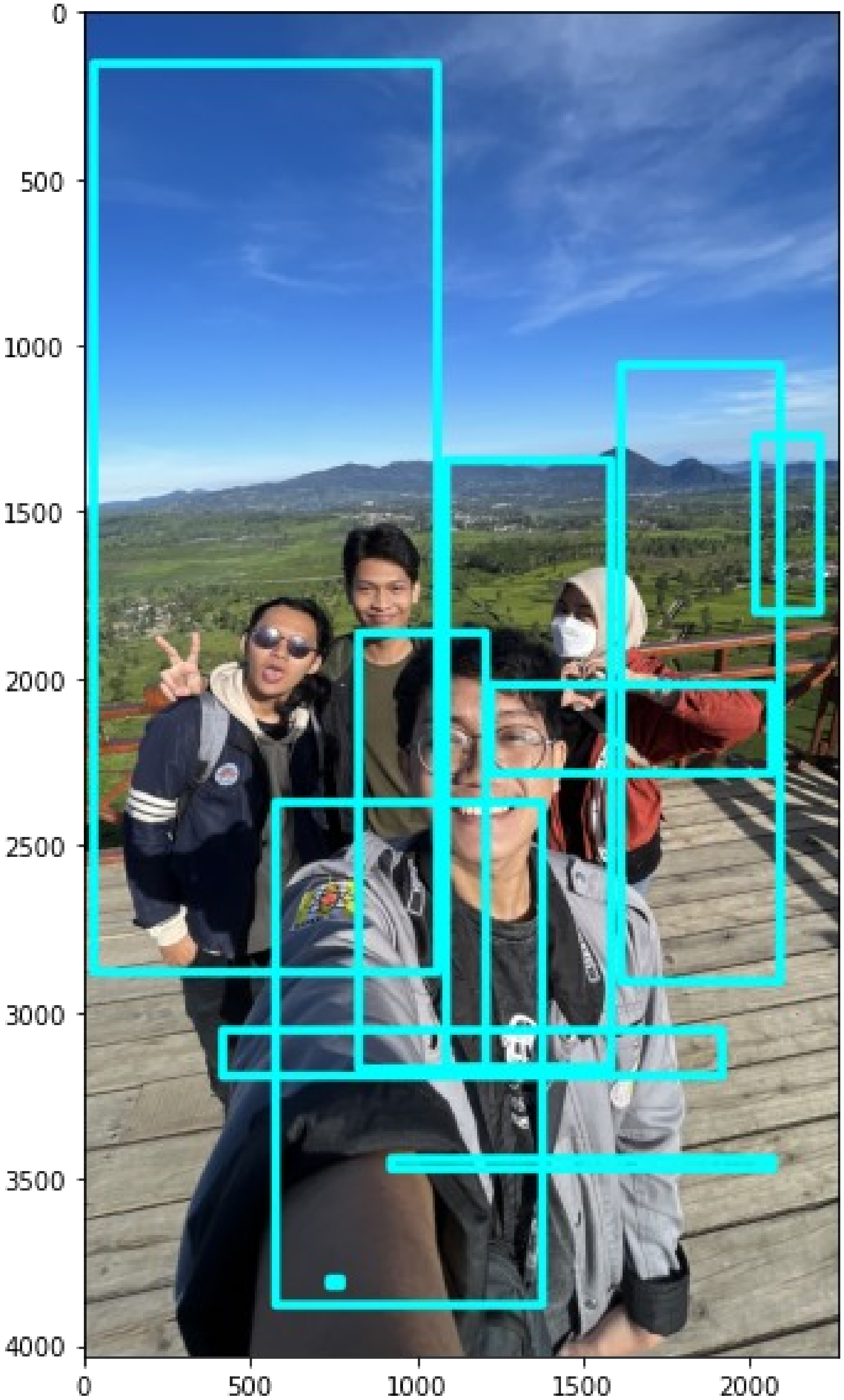


Diagram Venn Akurasi Template Matching Ketiga Algoritma

```
value_counts = df['accurate'].value_counts()
value_counts.plot(kind='pie', autopct='%1.1f%%')
```

```
plt.axis('equal')
plt.legend()
plt.title('Akurasi Template Matching')
plt.show()
```

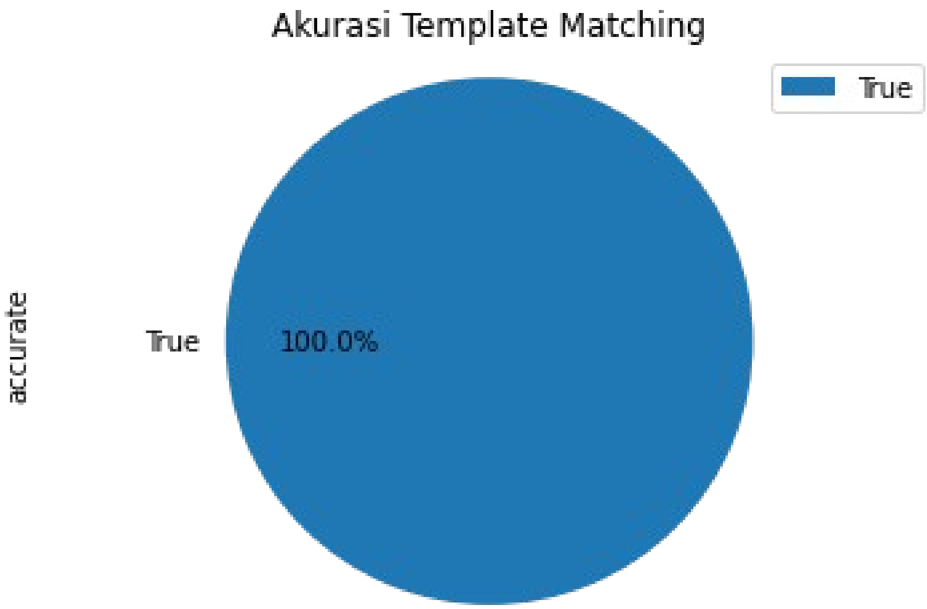


Diagram Batang Banyak Perbandingan Pixel Setiap Algoritma

```
fig, ax = plt.subplots()

# Lebar setiap bar
bar_width = 0.25

# Menghitung posisi setiap bar
r1 = range(len(df))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]

# Membuat bar chart untuk setiap kategori dan data
plt.bar(r1, df['BF'], color='b', width=bar_width, edgecolor='black',
label='BF')
plt.bar(r2, df['KMP'], color='g', width=bar_width, edgecolor='black',
label='KMP')
plt.bar(r3, df['BM'], color='r', width=bar_width, edgecolor='black',
label='BM')

# Mengatur label sumbu x
# plt.xlabel('Uji')
plt.xticks([r + bar_width for r in range(len(df))], [f'Uji {r+1}' for
r in range(len(df))])
```

```
# Mengatur label sumbu y
plt.ylabel('Banyak perbandingan')

# Menambahkan judul grafik
plt.title('Banyak Perbandingan Pixel Setiap Algoritma')

# Menambahkan legend
plt.legend()
plt.grid()

# Menampilkan grafik
plt.show()
```

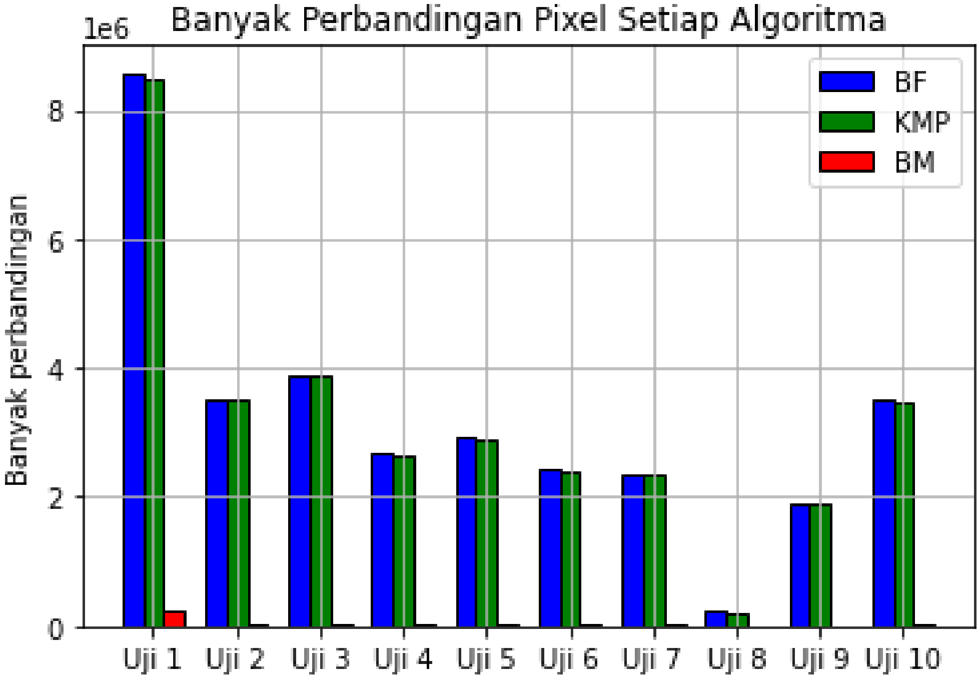


Diagram Batang Horizontal Waktu Eksekusi Setiap Algoritma

```
fig, ax = plt.subplots()

y_pos = np.arange(len(df))
bar_width = 0.2

# Membuat horizontal bar chart untuk setiap kategori dan data
plt.barh(y_pos, df['BF time (ms)'], height=bar_width, color='b',
edgecolor='black', label='BF')
plt.barh(y_pos + bar_width, df['KMP time (ms)'], height=bar_width,
color='g', edgecolor='black', label='KMP')
plt.barh(y_pos + (2 * bar_width), df['BM time (ms)'],
height=bar_width, color='r', edgecolor='black', label='BM')
```

```
# Mengatur label sumbu y
plt.yticks(y_pos + bar_width, [f'Uji {r+1}' for r in range(len(df))])

# Mengatur label sumbu y
plt.xlabel('Waktu Eksekusi (ms)')

# Menambahkan judul grafik
plt.title('Waktu Eksekusi Setiap Algoritma')

# Menambahkan legend
plt.grid()
plt.legend()

# Menampilkan grafik
plt.show()
```

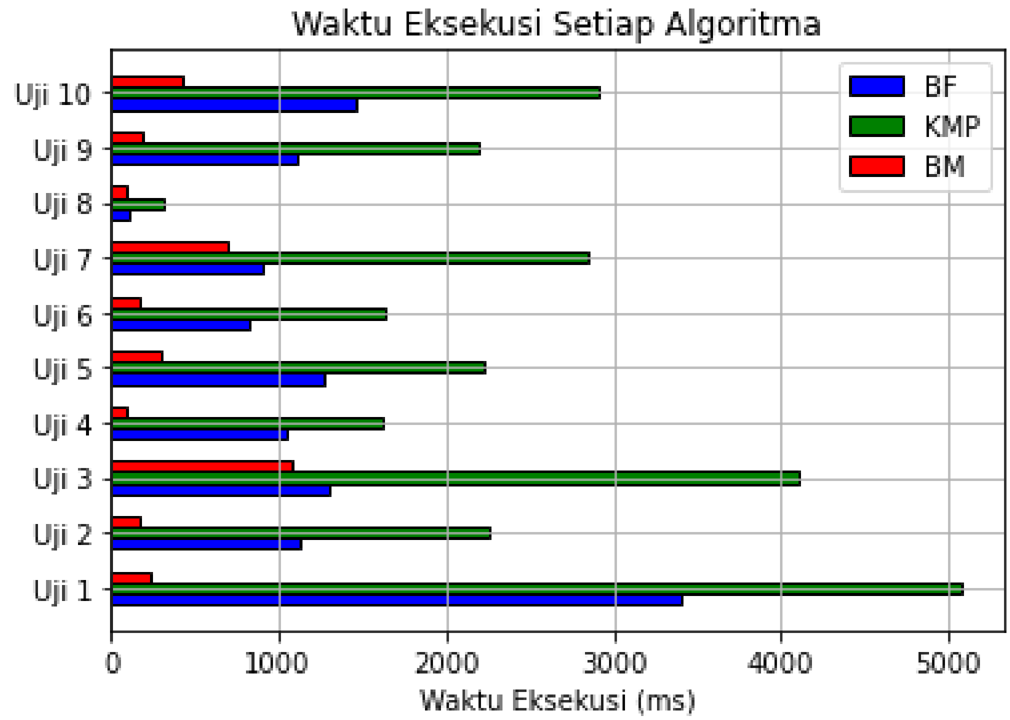
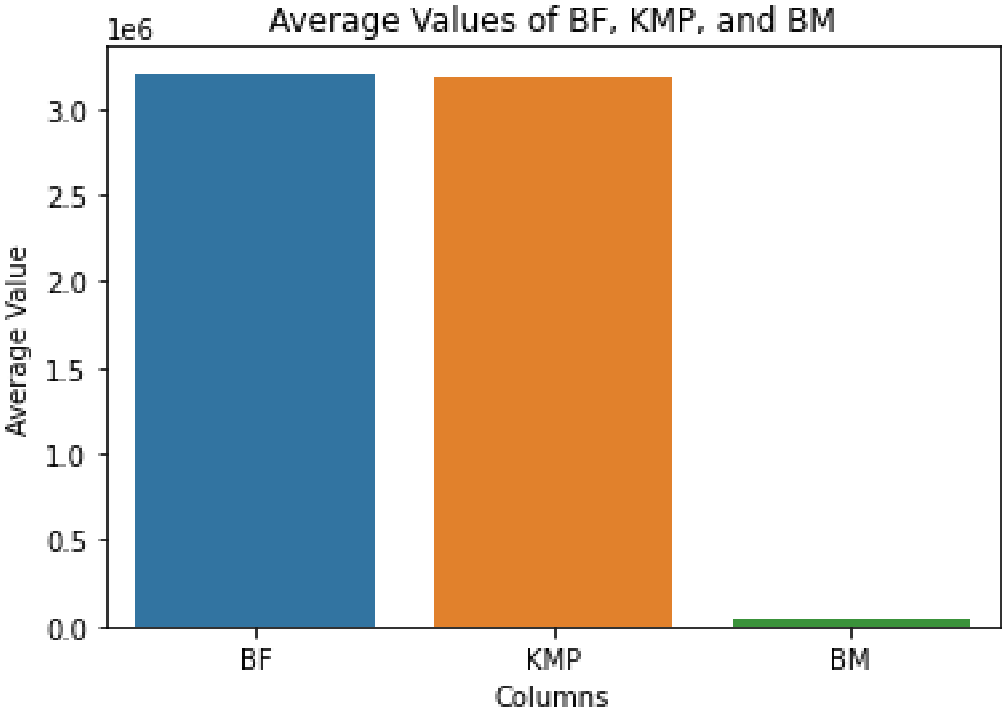


Diagram Batang Rata-Rata Perbandingan Setiap Algoritma

```
# Calculate the average values
selected_columns = ['BF', 'KMP', 'BM']
df_selected = df[selected_columns]

# Calculate the average values
avg_values = df_selected.mean()

# Create a bar plot using Seaborn
sns.barplot(x=avg_values.index, y=avg_values.values)
```

```

# Set the plot title and labels
plt.title('Average Values of BF, KMP, and BM')
plt.xlabel('Columns')
plt.ylabel('Average Value')

# Display the plot
plt.show()
```


Average Values of BF, KMP, and BM
1e6
Average Value
BF
KMP
BM
Columns