Jurnal Komputer dan Teknologi Informasi
Vol. 4, No. 1, Januari 2026, pp. 67~77
E-ISSN: 2986-7592, DOI: 10.26714/jkti.v4i1.18612.pp67-77

Penerapan pengenalan pola pada warna puzzle menggunakan metode *hybrid computer vision, edge matching,* dan *greedy heuristic*

Rizky Pratama Firdauz H.P[1], Riski Arya Putra [2], Syihab Akbar[3], Muhammad Yusuf Rizal[4], Muhammad Munsyarif[5]

[1]Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Semarang, Semarang, Indonesia
[2]Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Semarang, Semarang, Indonesia
[3]Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Semarang, Semarang, Indonesia
[4]Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Semarang, Semarang, Indonesia
[5]Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Semarang, Semarang, Indonesia

Info Artikel

Riwayat Artikel:

Diterima 15, Oktober, 2025
Perbaikan 22, Desember, 2025
Disetujui 13, Januari, 2026

Keywords:

Puzzle
Tepi
Hybridasi
Computer Vision
Edge Matching
Greedy Heuristic

ABSTRAK

Pengenalan pola dalam penyelesaian puzzle merupakan tantangan yang kompleks, terutama ketika melibatkan variasi warna dan bentuk potongan. Penelitian ini mengusulkan pendekatan hybridasi metode *computer vision*, *edge matching*, dan *greedy heuristic* untuk meningkatkan akurasi dan efisiensi dalam proses identifikasi dan penyusunan potongan puzzle. Metode *computer vision* digunakan untuk mendeteksi dan mengenali warna dominan pada setiap potongan, *edge matching* berperan dalam mencocokkan sisi potongan berdasarkan gambar. Algoritma *greedy heuristic* untuk menyusun potongan secara bertahap berdasarkan kecocokan terbaik yang tersedia pada setiap iterasi. Hasil eksperimen menunjukkan bahwa pendekatan hybrid ini mampu mempercepat proses penyusunan puzzle serta meningkatkan tingkat keberhasilan dalam pengenalan dan penempatan potongan yang tepat dengan akurasi tinggi. Penelitian ini memberikan kontribusi terhadap pengembangan sistem otomatisasi dalam pemrosesan visual dan pengenalan pola berbasis warna.

ABSTRACT

Pattern recognition in puzzle solving is a complex challenge, especially when it involves variations in color and shape of pieces. This research proposes a hybrid approach using computer vision, edge matching, and greedy heuristic methods to improve accuracy and efficiency in the process of identifying and assembling puzzle pieces. Computer vision methods are used to detect and recognize the dominant color of each piece, while edge matching plays a role in matching the edges of the pieces based on the image. The greedy heuristic algorithm gradually assembles the pieces based on the best match available at each iteration. Experimental results show that this hybrid approach can accelerate the puzzle assembly process and increase the success rate in recognizing and placing pieces correctly with high accuracy. This research contributes to the development of automated systems in visual processing and color-based pattern recognition.

` ***Penulis Korespondensi:***

Rizky Pratama Firdauz Heryawan Putra
Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Semarang
Alamat: Gedung GKB 2Lt. 7, Ruang 707, Jl.Kedungmundu Raya No.18, Semarang 50273, Indonesia
Email: c2c022024@student.unimus.ac.id

1. PENDAHULUAN

Puzzle adalah bentuk permainan atau tantangan yang dirancang untuk merangsang kemampuan berpikir seseorang, baik secara logis, analitis, maupun kreatif. Secara umum, puzzle melibatkan proses pemecahan masalah di mana pemain harus menemukan solusi yang tepat berdasarkan petunjuk, pola, atau potongan-potongan informasi yang tersedia. Tujuan utama dari puzzle bukan hanya untuk hiburan, tetapi juga untuk melatih otak agar lebih tajam dan responsif terhadap berbagai situasi. Jenis-jenis puzzle sangat beragam, mulai dari puzzle fisik yang mengharuskan pemain menyusun potongan gambar, puzzle logika, maupun puzzle dalam bentuk digital, seperti game interaktif berbasis aplikasi atau website yang menggabungkan elemen visual untuk menciptakan pengalaman bermain yang lebih kompleks. Pada penelitian ini menggunakan puzzle berbasis penyusunan warna dan gambar.

Pengenalan pola merupakan salah satu bidang penting dalam kecerdasan buatan dan pengolahan citra digital, yang memiliki berbagai aplikasi mulai dari pengenalan wajah hingga sistem navigasi otomatis. Dalam konteks penyusunan *puzzle*, pengenalan pola menjadi tantangan tersendiri karena melibatkan identifikasi bentuk, warna, dan kecocokan antar potongan yang kompleks. Puzzle tidak hanya menguji kemampuan spasial, tetapi juga menuntut sistem untuk memahami hubungan visual antar elemen yang tidak beraturan.

Dengan berkembangnya teknologi *computer vision*, pendekatan berbasis citra telah menjadi solusi potensial untuk menyelesaikan *puzzle* secara otomatis. Deteksi warna dominan pada potongan *puzzle* dapat membantu dalam klasifikasi awal, sementara analisis kontur dan tekstur memungkinkan pencocokan sisi yang lebih akurat. Namun, tantangan tetap ada dalam memilih urutan penyusunan yang efisien dan minim kesalahan. Untuk mengatasi hal tersebut, diperlukan strategi yang tidak hanya mengandalkan satu metode, tetapi menggabungkan beberapa pendekatan secara sinergis. Pendekatan *hybrid* yang mengintegrasikan pengolahan visual, pencocokan sisi, dan strategi heuristik menawarkan solusi yang lebih adaptif dan efisien dalam menyusun puzzle secara otomatis. Penelitian ini berfokus pada penerapan pendekatan tersebut untuk meningkatkan akurasi dan kecepatan dalam proses penyusunan puzzle berbasis warna dan bentuk.

Edge matching banyak digunakan dalam implementasi pengenalan pola, salah satunya penelitian mengenai pengaplikasian pengenalan objek dan registrasi citra [1]. Penelitian lain mengenai multilevel pada thresholding [2]. Penelitian tersebut mempunyai beberapa kelebihan seperti tahan dengan perubahan intensitas dan pencahayaan karena *edge matching* fokus pada informasi tepi, sehingga tahan dari perubahan pencahayaan atau warna. Cocok untuk penanganan data yang bising, tepi tidak sempurna karena mendukung berbagai transformasi geometris. Namun mempunyai kekurangan dalam membedakan objek yang tidak ada atau objek yang tepiannya hilang karena berpengaruh dalam algoritma ekstraksi tepi sehingga metode ini sensitif dalam segmemtasi awal. Kelemahan lainnya adalah kesusahan dalam membedakan tepi yang memiliki kesamaan tinggi yang dapat menurunkan tingkat keakurasiannya, serta kebutuhan ruang yang besar tergantung transformasi yang dilakukan yang mengharuskan edge berubah setiap frame, statistik jumlah jauh edge berubah dan representasi secara segmen [3].

Untuk mempertegas pondasi model, ditambahkanlah model lain untuk menutupi kelemahan yang ada dalam *edge matching*, metode yang memungkinkan untuk dilakukannya hybridasi dengan metode greedy heuristic. *Greedy heuristic* adalah metode konstruktif yang membangun solusi secara bertahap dengan memilih keputusan terbaik dalam setiap langkah awal tanpa pertimbangan selanjutnya, namun untuk mendapatkan efektivitas dalam metode ini adalah struktur yang sederhana dan dimana keputusan lokal cenderung mengarah dalam solusi global serta dibutuhkan kecepatan dalam penyelesaiannya [4]. *Greedy heuristic* dalam hybridasi dalam penelitian ini diperlukan untuk mempercepat tahap awal dalam penyusunan dimana banyaknya pilihan dan ingin cepat dalam membangun struktur awal dan pemilihan potongan yang paling mirip secara warna dari sisi yang berdekatan. Sedangkan *edge matching* digunakan dalam validasi kecocokan bentuk dan warna pada tepi, serta dapat mengurangi kelemahan greedy heuristic dalam pengambilan keputusan sesaat.

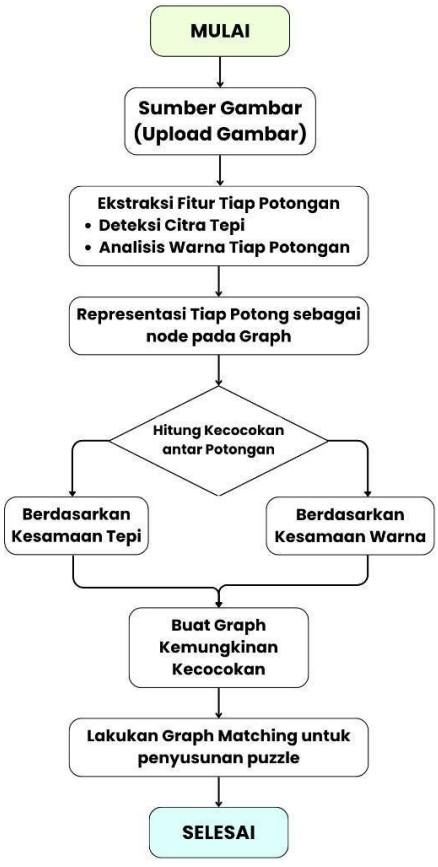
Dengan adanya hybridasi metode menggunakan edge matching dan greedy heuristic, membuat kinerja dari metode computer vision lebih meningkat. Pada tahapan matching dan penyusunan puzzle, metode heuristik dibutuhkan untuk menyaring dan memilih pasangan yang paling cocok yang setelah itu akan dilakukan. Setelah itu sistem melakukan rotasi dan translasi potongan untuk menyusun puzzle secara visual. Lalu pada tahap evaluasi kecocokan, metode computer vision dapat bekerja dengan lebih baik dengan menghitung nilai GAP

pada potongan yang dicocokkan, yaitu area yang tidak sesuai atau tidak saling berpasangan. GAP digunakan sebagai matriks untuk menilai kualitas pencocokan[5].

Aplikasi penyusunan *puzzle* secara otomatis ditujukan untuk diuji coba pada beberapa industri yang berkaitan. Pada industri game dan edukasi interaktif dengan menargetkan pengembang game edukatif, aplikasi pembelajaran anak, dan *platform* pembelajaran berbasis visual [6]. Aplikasi ini dapat digunakan sebagai sarana permainan *puzzle* yang adaptif dengan dukungan pembelajaran kognitif dengan meningkatkan interaksi pengguna melalui sistem yang dapat memberikan penenyelesaiannya secara otomatis. Pada industri kesehatan dan rehabilitasi kognitif dapat ditargetkan kepada klinik terapi okupasi dan pusat rehabilitasi sebagai upaya membantu pasien dengan gangguan kognitif agar dapat melatih kognitif dan motorik [7]. Dalam industri lainnya, konsep dari aplikasi ini dapat diimplementasikan pada berbagai pendekatan dibidang lain seperti otomotif dan robotika, dengan pendekatan penyusunan komponen berdasarkan bentuk dan warna dalam proses produksi dan perakitan. Sasaran lainnya terdapat dalam industri kreatif dan percetakan untuk membantu pengujian desain *puzzle* secara otomatis sebelum proses pencetakan serta menciptakan pengalaman interaktif berbasis *augmented reality*.

2. METODE

Metode yang digunakan dalam penelitian ini adalah hybridasi antara metode computer vision, edge matching, dan greedy heuristic dengan masing-masing tujuan untuk memaksimalkan model yang akan dijalankan. Dataset didapatkan dari beberapa contoh gambar yang didapatkan dari link berikut https://www.cct.lsu.edu/~cliu/PuzzleDataset/GVCPuzzles.html. Setelah itu dimulailah untuk memproses untuk membuat website untuk penyusunan puzzle dengan algoritma sebagai berikut



Gambar 1. Algoritma penyusunan puzzle

Tahapan pertama dalam proses ini dimulai dengan pengambilan fitur visual dari gambar utuh yang diunggah, di mana gambar tersebut dibagi menjadi beberapa segmen (dalam penelitian ini, menggunakan grid 3x3). Setiap segmen dianalisis menggunakan teknik *Computer Vision* untuk menemukan karakteristik visual seperti deteksi tepi dan distribusi warna.

```python
def extract_pieces(image_path, output_folder, session_id):
    img = cv2.imread(image_path)
    if img is None:
        return []

    h, w = img.shape[:2]
    rows, cols = 3, 3
    ph, pw = h // rows, w // cols
    extracted_paths = []

    for i in range(rows):
        for j in range(cols):
            piece = img[i*ph:(i+1)*ph, j*pw:(j+1)*pw]
            filename = f"{session_id}_{i}_{j}.png"
            save_path = os.path.join(output_folder, filename)
            cv2.imwrite(save_path, piece)
            extracted_paths.append(os.path.join("static", "extracted", filename))

    return extracted_paths
```

Gambar 2. Potongan Kode *Computer Vision*

Setelah proses ekstraksi fitur menggunakan *Computer Vision*, dilakukan penilaian kesesuaian antar bagian puzzle dengan menggunakan metode *Edge Matching*.

```python
def compute_edge_similarity(img1, img2):
    right = img1[:, -5:, :].astype("int32")
    left = img2[:, :5, :].astype("int32")
    diff = np.abs(right - left)
    score = 1 - (np.mean(diff) / 255.0)
    return score

def total_edge_score(order, pieces, rows=3, cols=3):
    score = 0.0
    h, w = pieces[0].shape[:2]
    for r in range(rows):
        for c in range(cols):
            idx = r * cols + c
            current = pieces[order[idx]]

            # cek kanan
            if c (variable) right_idx: int
                right_idx = r * cols + (c + 1)
                neighbor = pieces[order[right_idx]]
                score += compute_edge_similarity(current, neighbor)

            # cek bawah
            if r < rows - 1:
                down_idx = (r + 1) * cols + c
                neighbor = pieces[order[down_idx]]
                down_score = compute_edge_similarity(current.transpose(1, 0, 2), neighbor.transpose(1, 0, 2))
                score += down_score
    return score
```

Gambar 3. Potongan Kode *Edge Matching*

Di tahap ini, sistem membandingkan tepi-tepi dari bagian puzzle berdasarkan kesamaan nilai piksel yang terdapat pada bagian tersebut. Skor kesesuaian dihitung dengan menemukan selisih absolut dari rata-rata piksel di tepi kiri dan kanan dari dua bagian yang dibandingkan. Hasil dari penghitungan kesesuaian tersebut selanjutnya ditampilkan dalam bentuk graf, di mana setiap simpul melambangkan satu bagian puzzle, dan bobot antar simpul menunjukkan tingkat kesesuaian di antara bagian-bagian tersebut.

```python
def match_puzzle(pieces_paths, result_folder, session_id):
    pieces = [cv2.imread(p) for p in pieces_paths]
    if any(p is None for p in pieces):
        return ""

    # Penyusunan berdasarkan indeks skor
    n = len(pieces)
    rows, cols = 3, 3
    h, w = pieces[0].shape[:2]
    result_img = np.zeros((h*rows, w*cols, 3), dtype=np.uint8)

    for idx in range(n):
        r, c = divmod(idx, cols)
        result_img[r*h:(r+1)*h, c*w:(c+1)*w] = pieces[idx]
```

Gambar 4. Potongan Kode *Greedy Heuristic*

Untuk menyusun kembali puzzle, sistem menerapkan pendekatan *Greedy Heuristic*, yaitu memilih pasangan bagian dengan skor kesesuaian tertinggi secara bertahap tanpa perlu mengecek setiap kombinasi secara menyeluruh. Pendekatan ini diambil karena lebih efisien dalam hal perhitungan dan cukup efektif bila jumlah bagian puzzle terbatas. Dengan penggabungan ketiga metode tersebut, diharapkan sistem mampu menyusun kembali puzzle secara otomatis dengan hasil yang tepat dan mirip dengan gambar aslinya.

3. HASIL DAN PEMBAHASAN

```python
def greedy_assignment(cost_matrix):
    n = len(cost_matrix)
    used_rows = set()
    used_cols = set()
    assignments = []

    for _ in range(n):
        min_cost = float('inf')
        min_row, min_col = -1, -1
        for i in range(n):
            if i in used_rows: continue
            for j in range(n):
                if j in used_cols: continue
                if cost_matrix[i][j] < min_cost:
                    min_cost = cost_matrix[i][j]
                    min_row, min_col = i, j
        if min_row != -1 and min_col != -1:
            assignments.append((min_row, min_col))
            used_rows.add(min_row)
            used_cols.add(min_col)
    return assignments
```

Fungsi greedy_assignment merupakan implementasi dari algoritma Greedy Heuristic yang digunakan untuk menyelesaikan assignment problem atau masalah penugasan, di mana tujuannya adalah mencocokkan setiap baris ke kolom dengan biaya seminimal mungkin berdasarkan matriks biaya (cost_matrix). Fungsi ini bekerja dengan cara memilih pasangan baris dan kolom yang belum digunakan dengan nilai biaya terkecil secara iteratif. Pertama-tama, fungsi mencatat ukuran dari matriks biaya dan menyiapkan dua himpunan used_rows dan used_cols untuk mencatat baris dan kolom yang telah dialokasikan. Lalu dalam setiap iterasi sebanyak n kali, fungsi mencari kombinasi baris dan kolom dengan nilai biaya terkecil yang belum digunakan, lalu mencatatnya ke dalam daftar assignments. Setelah sebuah pasangan dipilih, indeks baris dan kolom tersebut ditandai sebagai telah digunakan agar tidak dipilih kembali di iterasi berikutnya. Proses ini dilakukan hingga semua elemen telah dipasangkan. Pada akhirnya, fungsi akan mengembalikan daftar pasangan baris dan kolom yang merepresentasikan hasil penugasan dengan pendekatan greedy.

```
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Function to extract edge features (example: average color along edges)
def extract_edge_features(tile):
    h, w, _ = tile.shape
    top_edge = np.mean(tile[0, :, :], axis=0)
    bottom_edge = np.mean(tile[h-1, :, :], axis=0)
    left_edge = np.mean(tile[:, 0, :], axis=0)
    right_edge = np.mean(tile[:, w-1, :], axis=0)
    return {'top': top_edge, 'bottom': bottom_edge, 'left': left_edge, 'right': right_edge}

# Function to calculate compatibility cost between two edges
def edge_compatibility_cost(edge1, edge2):
    return np.sum((edge1 - edge2)**2)

# Function to build the cost matrix for matching
def build_cost_matrix(shuffled_tiles, original_tiles_features, rows=3, cols=3):
    n = len(shuffled_tiles)
    cost_matrix = np.zeros((n, n))
    shuffled_tiles_features = [extract_edge_features(tile) for tile in shuffled_tiles]

    for i in range(n):
        for j in range(n):
            # Calculate cost based on compatibility with surrounding tiles in the original position
            original_row, original_col = divmod(j, cols)
            shuffled_row, shuffled_col = divmod(i, cols) # Assume shuffled tile i goes to position j
```

Kode ini bertujuan untuk menyusun kembali gambar puzzle yang telah diacak berdasarkan kecocokan antar sisi (edge) dari setiap potongan gambar (tile). Proses dimulai dengan menyembunyikan sumbu tampilan dan merapikan layout visual untuk menampilkan gambar yang akan diolah. Kemudian, dari setiap tile diekstrak fitur sisi-sisinya—atas, bawah, kiri, dan kanan—dengan menghitung rata-rata nilai warna (RGB) dari masing-masing sisi. Fitur-fitur ini disimpan dalam dictionary yang merepresentasikan setiap sisi tile.

Selanjutnya, dilakukan perhitungan *edge compatibility cost* atau tingkat ketidakcocokan antara dua sisi menggunakan metode *Mean Squared Error* (MSE), yaitu dengan mengkuadratkan selisih nilai warna dan menjumlahkannya. Nilai ini menunjukkan seberapa mirip atau berbedanya dua sisi yang dibandingkan.

Berdasarkan hasil perhitungan tersebut, dibentuklah *cost matrix* (matriks biaya) berukuran *n x n*, yang digunakan untuk mencocokkan tile acak ke posisi grid aslinya. Matriks ini dibangun dengan membandingkan setiap tile dengan semua kemungkinan posisi di grid menggunakan dua perulangan, sementara fungsi divmod digunakan untuk mengubah indeks satu dimensi menjadi koordinat baris dan kolom.

Meski bagian akhir perhitungan cost matrix belum terlihat secara lengkap dalam potongan kode, keseluruhan proses ini membentuk dasar untuk penyusunan ulang puzzle. Proses penyusunan optimalnya akan dilanjutkan menggunakan algoritma seperti pendekatan heuristik lainnya seperti *Greedy*, untuk menentukan urutan tile terbaik berdasarkan nilai kecocokan sisi.

```
            cost = 0
            # Check compatibility with potential neighbors in the original grid
            if original_row > 0: # Check top neighbor
                cost += edge_compatibility_cost(shuffled_tiles_features[i]['top'], original_tiles_features[j - cols]['bottom'])
            if original_row < rows - 1: # Check bottom neighbor
                cost += edge_compatibility_cost(shuffled_tiles_features[i]['bottom'], original_tiles_features[j + cols]['top'])
            if original_col > 0: # Check left neighbor
                cost += edge_compatibility_cost(shuffled_tiles_features[i]['left'], original_tiles_features[j - 1]['right'])
            if original_col < cols - 1: # Check right neighbor
                cost += edge_compatibility_cost(shuffled_tiles_features[i]['right'], original_tiles_features[j + 1]['left'])

            cost_matrix[i, j] = cost
    return cost_matrix
```

```
def reconstruct_with_greedy(shuffled_tiles, rows=3, cols=3):
    n = len(shuffled_tiles)
    original_tiles = split_image_into_tiles(img, rows, cols)
    original_tiles_features = [extract_edge_features(tile) for tile in original_tiles]

    print("Membuat matriks biaya (Greedy)...")
    cost_matrix = build_cost_matrix(shuffled_tiles, original_tiles_features, rows, cols)

    print("Menjalankan algoritma Greedy...")
    assigned_rows = set()
    reconstructed_tiles = [None] * n
    tile_indices = [None] * n

    for col in range(n):  # Setiap posisi asli
        # Cari tile teracak terbaik (cost paling kecil) yang belum dipakai
        min_cost = float('inf')
        best_row = None
        for row in range(n):
            if row not in assigned_rows and cost_matrix[row, col] < min_cost:
                min_cost = cost_matrix[row, col]
                best_row = row
```

```
                best_row = row

        if best_row is not None:
            reconstructed_tiles[col] = shuffled_tiles[best_row]
            tile_indices[col] = best_row
            assigned_rows.add(best_row)

    return reconstructed_tiles, tile_indices
```

Fungsi reconstruct_with_greedy bertujuan untuk menyusun kembali gambar yang telah diacak (puzzle gambar) menggunakan pendekatan Greedy Heuristic. Fungsi ini pertama-tama menerima parameter berupa kumpulan tile (bagian-bagian gambar yang telah diacak), serta jumlah baris dan kolom dari gambar tersebut (default-nya 3x3). Langkah pertama dalam fungsi ini adalah memanggil split_image_into_tiles untuk membagi gambar asli (img) menjadi potongan-potongan kecil sesuai dengan ukuran baris dan kolom, dan kemudian setiap tile-nya diekstrak fitur tepinya dengan extract_edge_features, menghasilkan data ciri tepi dari setiap potongan gambar asli.

Setelah fitur tepi dikumpulkan, fungsi membuat matriks biaya (cost matrix) menggunakan build_cost_matrix, yang mengukur seberapa mirip antara setiap tile acak dan tile asli berdasarkan fitur tepinya. Proses ini penting karena akan menentukan seberapa cocok suatu tile ditempatkan di posisi tertentu.

Kemudian, fungsi menjalankan algoritma greedy dengan cara memilih secara iteratif tile acak terbaik (yaitu yang memiliki biaya terkecil di kolom tertentu dan belum digunakan sebelumnya). Ini dilakukan dengan perulangan untuk setiap kolom (posisi asli tile), di mana fungsi mencari baris (tile acak) dengan nilai terkecil di cost matrix dan belum dipakai. Jika ditemukan, tile tersebut ditambahkan ke daftar reconstructed_tiles di posisi kolom tersebut dan indeksnya disimpan di tile_indices.

Hasil akhirnya adalah dua buah list: reconstructed_tiles, yaitu daftar potongan gambar yang sudah tersusun ulang berdasarkan heuristik greedy, dan tile_indices, yaitu urutan indeks dari tile acak yang digunakan untuk menyusun gambar tersebut.

```python
# Function to calculate matching percentage between two tiles
def calculate_tile_matching(tile1, tile2):
    # Convert tiles to grayscale for simpler comparison
    gray1 = cv2.cvtColor(tile1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(tile2, cv2.COLOR_BGR2GRAY)

    # Calculate Mean Squared Error
    mse = np.mean((gray1 - gray2) ** 2)
    # Convert MSE to similarity percentage (inverse relationship)
    max_mse = 255 ** 2  # Maximum possible MSE for 8-bit images
    similarity = 100 * (1 - mse / max_mse)
    return similarity
```

Fungsi calculate_tile_matching digunakan untuk mengukur tingkat kemiripan antara dua potongan gambar (tile) dalam bentuk persen. Langkah pertama dalam fungsi ini adalah mengubah kedua tile, yaitu tile1 dan tile2, ke dalam format grayscale menggunakan OpenCV (cv2.cvtColor) agar perbandingan lebih sederhana karena hanya melibatkan satu kanal warna. Setelah itu, fungsi menghitung nilai Mean Squared Error (MSE), yaitu rata-rata kuadrat selisih piksel antar kedua gambar grayscale. Nilai MSE ini menunjukkan seberapa besar perbedaan antara dua tile: semakin kecil MSE, semakin mirip kedua gambar tersebut. Kemudian, nilai MSE ini dikonversi menjadi persentase kemiripan (similarity) dengan rumus similarity = 100 * (1 - mse / max_mse), di mana max_mse adalah nilai MSE maksimum untuk gambar 8-bit (yaitu 255^2). Hasil akhir dari fungsi ini adalah angka kemiripan dalam persentase, yang bisa digunakan untuk menentukan seberapa cocok dua tile dalam puzzle gambar.

```python
# Function to run the puzzle process
def run_puzzle(img, rows=3, cols=3):
    if img is None:
        return

    print("Memulai proses puzzle...")
    print(f"Memotong gambar menjadi {rows*cols} bagian...")
    original_tiles = split_image_into_tiles(img, rows, cols)

    print("Mengacak potongan puzzle...")
    shuffled_tiles = shuffle_tiles(original_tiles)

    print("\nMemulai proses rekonstruksi puzzle...")
    print("Mohon tunggu, ini mungkin memerlukan waktu beberapa saat...")

    reconstructed_tiles, tile_indices = reconstruct_with_greedy(shuffled_tiles, rows, cols)

    if reconstructed_tiles is None:
        print("Rekonstruksi gagal.")
        return

    # Display all puzzles side by side
    print("\nMenampilkan hasil puzzle:")
    tile_sets = [original_tiles, shuffled_tiles, reconstructed_tiles]
    titles = ["Puzzle Asli", "Puzzle Teracak", "Hasil Rekonstruksi"]
    display_tiles_side_by_side(tile_sets, titles, rows, cols)
```

Fungsi run_puzzle didefinisikan untuk menjalankan seluruh proses rekonstruksi puzzle gambar. Baris pertama dari fungsi ini melakukan validasi awal dengan memeriksa apakah parameter image bernilai None. Jika ya, maka fungsi akan langsung mengembalikan nilai dan tidak melanjutkan proses. Hal ini dilakukan untuk

memastikan bahwa input gambar tersedia sebelum proses dilanjutkan. Setelah itu, gambar akan dibagi menjadi beberapa potongan kecil (tiles) berdasarkan jumlah baris dan kolom menggunakan fungsi split_image_into_tiles. Potongan-potongan gambar yang dihasilkan akan disimpan dalam variabel original_tiles.

Selanjutnya, fungsi shuffle_tiles digunakan untuk mengacak urutan potongan gambar sehingga menciptakan kondisi puzzle yang acak. Hasil pengacakan ini disimpan dalam shuffled_tiles. Setelah itu, sistem mencetak bahwa proses rekonstruksi akan dimulai. Sebagai gantinya, gambar akan direkonstruksi menggunakan pendekatan Greedy Heuristik, yang berusaha mencocokkan potongan-potongan gambar berdasarkan kesamaan (similarity) piksel secara berurutan dan memilih kecocokan terbaik di setiap langkah.

Fungsi reconstruct_with_greedy(shuffled_tiles, rows, cols) dijalankan untuk menyusun kembali gambar yang teracak tersebut. Jika proses rekonstruksi gagal, akan dicetak pesan bahwa rekonstruksi gagal dan fungsi berhenti. Namun jika berhasil, sistem akan mencetak bahwa hasil puzzle akan ditampilkan. Variabel tile_sets disiapkan sebagai tuple yang berisi tiga versi gambar: versi asli (original_tiles), versi acak (shuffled_tiles), dan hasil rekonstruksi (reconstructed_tiles). Masing-masing dari ketiga gambar ini diberi label menggunakan titles, yaitu: “Asli,” “Telah Teracak,” dan “Telah Direkonstruksi.” Terakhir, fungsi display_tiles digunakan untuk menampilkan ketiganya secara berdampingan dalam satu jendela tampilan agar dapat dibandingkan secara visual.

```
    # Calculate and display matching percentages with colored marks
    print("\nLaporan Kecocokan Tile:")
    print("-------------------------------------")
    total_similarity = 0

    # ANSI color codes
    GREEN = '\033[92m'  # Hijau
    RED = '\033[91m'    # Merah
    RESET = '\033[0m'   # Reset warna

    for i in range(len(original_tiles)):
        similarity = calculate_tile_matching(original_tiles[i], reconstructed_tiles[i])
        total_similarity += similarity
        # Tambahkan centang hijau (✓) jika kecocokan > 90%, silang merah (X) jika tidak
        mark = f"{GREEN}✓{RESET}" if similarity > 90 else f"{RED}X{RESET}"
        print(f"Tile {i+1}: {similarity:.2f}% cocok dengan posisi asli {mark}")

    print("-------------------------------------")
    average_similarity = total_similarity / len(original_tiles)
    mark = f"{GREEN}✓{RESET}" if average_similarity > 90 else f"{RED}X{RESET}"
    print(f"Rata-rata kecocokan seluruh puzzle: {average_similarity:.2f}% {mark}")


# --- Main Execution ---
print("Silakan unggah gambar Anda.")
img = upload_image()
```

Kode ini digunakan untuk menghitung dan menampilkan persentase kecocokan antara tile (potongan gambar) asli dengan hasil rekonstruksi. Pertama-tama, program mencetak judul "Laporan Kecocokan Tile" sebagai penanda awal laporan evaluasi. Kemudian, variabel total_similarity diinisialisasi ke 0 untuk menampung total dari nilai kemiripan seluruh tile.

Setelah itu, didefinisikan beberapa kode warna ANSI untuk menandai output terminal: hijau (GREEN) untuk hasil cocok, merah (RED) untuk tidak cocok, dan RESET untuk mengembalikan ke warna default. Ini hanya akan terlihat jika dijalankan di lingkungan yang mendukung warna terminal.

Program kemudian masuk ke dalam perulangan for untuk setiap indeks tile. Dalam loop ini, calculate_tile_similarity() digunakan untuk menghitung kemiripan (similarity) antara tile asli dan tile hasil rekonstruksi berdasarkan indeks ke-i. Nilai kemiripan ini ditambahkan ke total_similarity. Jika nilai similarity di atas 90%, output ditandai dengan warna hijau dan disebut "COCOK", jika tidak, akan ditandai merah dan disebut "TIDAK COCOK".

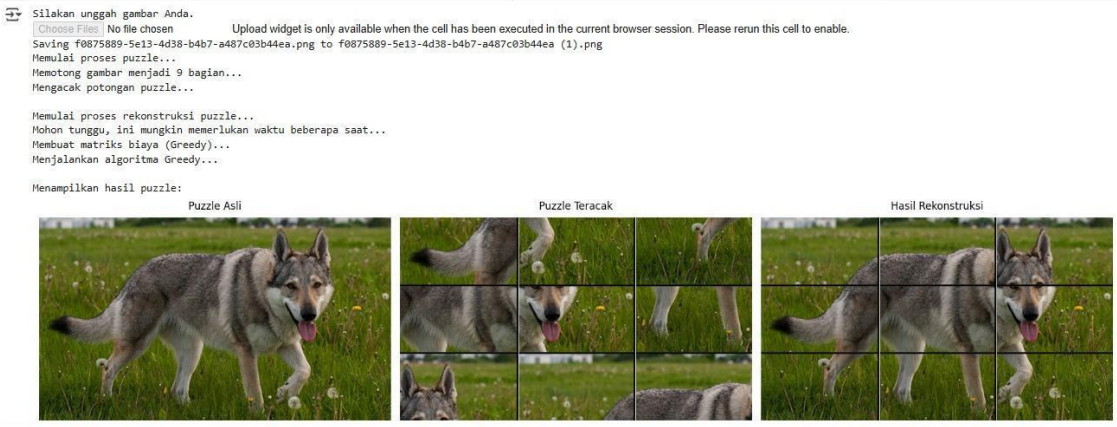
Setelah semua tile dicek, program menghitung rata-rata kecocokan dengan membagi total_similarity dengan jumlah tile. Berdasarkan nilai rata-rata ini, jika lebih dari 90%, ditampilkan dengan warna hijau, jika tidak maka berwarna merah. Output akhir mencetak persentase kecocokan total puzzle dan warna statusnya. Terakhir, bagian eksekusi utama program meminta pengguna untuk mengunggah gambar dengan upload_image(), dan menyimpannya ke dalam variabel img. Fungsi ini menjadi titik awal proses rekonstruksi puzzle secara keseluruhan.

```
# Check if image was uploaded successfully
if img is not None:
    run_puzzle(img)
else:
    print("Tidak ada gambar yang diunggah atau terjadi kesalahan saat mengunggah.")
```

Kode ini berfungsi untuk memeriksa apakah gambar berhasil diunggah sebelum menjalankan proses puzzle. Pada baris pertama, terdapat komentar # check if image was uploaded successfully sebagai penanda fungsi dari blok kode ini. Selanjutnya, kondisi if img is not None: digunakan untuk memastikan bahwa variabel img telah berisi gambar hasil unggahan.

Jika gambar berhasil diunggah (tidak None), maka fungsi run_puzzle(img) akan dipanggil untuk memulai seluruh proses pemecahan dan rekonstruksi puzzle berdasarkan gambar tersebut. Namun, jika img adalah None, maka artinya tidak ada gambar yang berhasil diunggah atau terjadi kesalahan saat proses upload. Dalam kasus ini, program mencetak pesan error "Tidak ada gambar yang diunggah atau terjadi kesalahan saat mengunggah."

Secara keseluruhan, bagian kode ini bertugas sebagai pengaman (error handling) agar program tidak mencoba memproses gambar yang belum tersedia, sehingga mencegah error lebih lanjut dalam eksekusi.



Gambar ini menunjukkan hasil dari proses pemecahan dan rekonstruksi puzzle gambar menggunakan algoritma pemrograman di Google Colab. Proses dimulai dengan mengunggah gambar berjudul *Dwonload.jpeg* sebagai input. Selanjutnya, gambar asli tersebut dipecah menjadi beberapa bagian atau tile. Setelah itu, tile yang telah dipotong diacak posisinya sehingga menghasilkan tampilan puzzle acak yang berbeda dari gambar aslinya. Untuk menyusun kembali puzzle tersebut, digunakan algoritma *greedy* guna mencocokkan tile berdasarkan kemiripan. Hasil visual dari proses ini ditampilkan dalam tiga bagian: gambar asli sebelum diacak, tampilan puzzle acak, dan hasil rekonstruksi oleh algoritma. Terlihat bahwa hasil rekonstruksi sangat mirip dengan gambar aslinya, yang menunjukkan bahwa algoritma berhasil menyusun ulang tile dengan akurat berdasarkan tingkat kemiripan masing-masing tile.

```
Laporan Kecocokan Tile:
----------------------------------------
Tile 1: 100.00% cocok dengan posisi asli ✓
Tile 2: 100.00% cocok dengan posisi asli ✓
Tile 3: 100.00% cocok dengan posisi asli ✓
Tile 4: 100.00% cocok dengan posisi asli ✓
Tile 5: 100.00% cocok dengan posisi asli ✓
Tile 6: 100.00% cocok dengan posisi asli ✓
Tile 7: 100.00% cocok dengan posisi asli ✓
Tile 8: 100.00% cocok dengan posisi asli ✓
Tile 9: 100.00% cocok dengan posisi asli ✓
----------------------------------------
Rata-rata kecocokan seluruh puzzle: 100.00% ✓
```

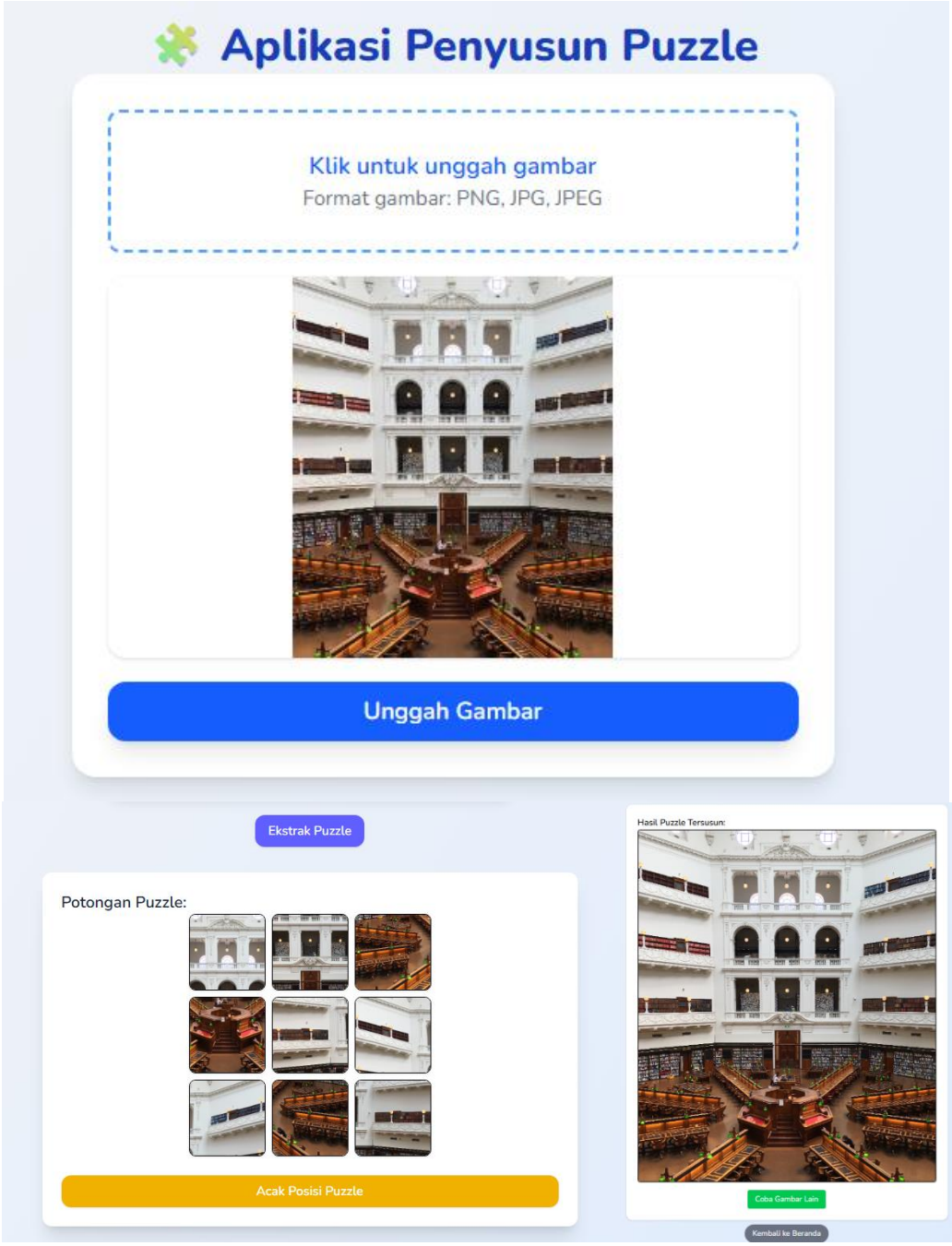
Penjelasan output:

- Setiap tile (potongan gambar) dinilai kecocokannya berdasarkan posisi asli.
- Terdapat 9 tile, masing-masing diberi persentase kecocokan:
- Semua tile memiliki nilai 100.00% kecocokan, artinya tile berhasil ditempatkan ke posisi semula
- Tanda "✓" menunjukkan bahwa tile tersebut benar dan sesuai dengan posisi semula.
- Di bagian bawah, tercantum Rata-rata kecocokan seluruh puzzle: 100.00% ✓, yang menunjukkan bahwa proses rekonstruksi berhasil sempurna.

4. KESIMPULAN

Penelitian ini berhasil menunjukkan bahwa pendekatan hybrid yang menggabungkan computer vision, edge matching, dan greedy heuristic mampu meningkatkan akurasi dan efisiensi dalam proses penyusunan puzzle berbasis warna dan bentuk. Metode computer vision digunakan untuk mendeteksi fitur visual seperti warna dominan dan tepi gambar, edge matching berperan dalam mencocokkan sisi potongan berdasarkan kontur dan tekstur, sementara greedy heuristic mempercepat proses penyusunan dengan memilih pasangan potongan terbaik secara bertahap.

Hasil eksperimen menunjukkan bahwa sistem yang dibangun mampu menyusun kembali puzzle secara otomatis dengan tingkat keberhasilan yang tinggi, bahkan mencapai 100% kecocokan dalam pengujian. Pendekatan ini juga memiliki potensi aplikasi luas di berbagai industri, seperti game edukatif, rehabilitasi kognitif, robotika, dan percetakan kreatif, serta dapat mendukung pengembangan sistem berbasis augmented reality.



UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Dr. Muhammad Munsarif, S.Kom., M.Kom. selaku dosen pengampu mata kuliah Pengenalan Pola atas bimbingan, arahan, dan kesempatan yang telah diberikan sehingga sehingga jurnal berjudul " Penerapan pengenalan pola pada warna puzzle menggunakan metode hybrid computer vision, edge matching, dan greedy heuristic" ini dapat disusun dan diselesaikan dengan baik. Ucapan terima kasih juga disampaikan kepada seluruh pihak yang telah berkontribusi, baik secara langsung maupun tidak langsung, dalam proses pelaksanaan penelitian dan

penyusunan jurnal ini, termasuk rekan-rekan yang turut membantu dalam pengumpulan data dan pengujian sistem. Semoga hasil penelitian ini dapat memberikan manfaat dan kontribusi nyata dalam pengembangan teknologi

REFERENCES

[1] G. Borgefors, "Hierarchical chamfer matching.," vol. I, no. 6, 1985.
[2] L. Hertz and R. W. Schafer, "Multilevel thresholding using edge matching," *Comput. Vision, Graph. Image Process.*, vol. 44, no. 3, pp. 279–295, 1988, doi: 10.1016/0734-189X(88)90125-9.
[3] M. J. Hossain, M. A. A. Dewan, and O. Chae, "A flexible edge matching technique for object detection in dynamic environment," *Appl. Intell.*, vol. 36, no. 3, pp. 638–648, 2012, doi: 10.1007/s10489-011-0281-4.
[4] E. A. Silver, R. Victor, V. Vidal, and D. de Werra, "A tutorial on heuristic methods," *Eur. J. Oper. Res.*, vol. 5, no. 3, pp. 153–162, 1980, doi: 10.1016/0377-2217(80)90084-3.
[5] R. W. Webster, P. S. LaFollette, and R. L. Stafford, "Isthmus Critical Points for Solving Jigsaw Puzzles in Computer Vision," *IEEE Trans. Syst. Man Cybern.*, vol. 21, no. 5, pp. 1271–1278, 1991, doi: 10.1109/21.120080.
[6] N. Aral, F. Gursoy, and M. C. Yasar, "An Investigation of the Effect of Puzzle Design on Children's Development Areas," *Procedia - Soc. Behav. Sci.*, vol. 51, pp. 228–233, 2012, doi: 10.1016/j.sbspro.2012.08.150.
[7] T. Nef *et al.*, "Development and Evaluation of Maze-Like Puzzle Games to Assess Cognitive and Motor Function in Aging and Neurodegenerative Diseases," *Front. Aging Neurosci.*, vol. 12, no. April, 2020, doi: 10.3389/fnagi.2020.00087.