Soluția Propusă pentru Rezolvarea Task-urilor de Procesare a Imaginilor Jocul Mathable

Autor: Podeanu Mate-Alexandru Grupa 462

3 decembrie 2024

Cuprins

1 Introducere

În această lucrare, prezentăm o soluție completă pentru rezolvarea a trei task-uri esențiale în analiza poziției și identificării pieselor pe o tablă de joc, precum și pentru calcularea scorului. Soluția este implementată utilizând Python și biblioteca OpenCV pentru procesarea imaginilor. Scopul este de a oferi o explicație detaliată, împreună cu secvențe de cod relevante și ilustrații conceptuale, pentru a facilita reimplementarea de către studenții de nivel mediu.

2 Task 1: Depistarea Poziției Piesei pe Tablă

2.1 Obiectiv

Detectarea poziției exacte a unei piese plasate pe tablă, determinând rândul și coloana acesteia.

2.2 Abordare

1. **Generarea Grilei de Linie:** Creăm grile orizontale și verticale care delimitează celulele tablei de joc.
2. **Extracția și Transformarea Perspectivei Tablei:** Utilizăm detecția contururilor și transformarea perspectivei pentru a obține o imagine corectă a tablei.
3. **Procesarea și Compararea Imaginilor:** Comparam imaginea actuală cu o imagine de bază pentru a identifica diferențele.
4. **Determinarea Celulei cu Diferență Maximă:** Identificăm celula care prezintă cea mai mare diferență, indicând poziția noii piese.

2.3 Pași Detaliați

2.3.1 1. Generarea Grilei de Linie

Se generează coordonatele liniilor orizontale și verticale care formează grila tablei. Acest lucru facilitează segmentarea tablei în celule distincte de dimensiuni egale.

Listing 1: Generarea Grilei de Linie

```
def generate_grid_lines():
    horizontal_lines = [
        [(350, y), (2440, y)] for y in range(370, 2452, 148)
    ]
    vertical_lines = [
        [(x, 370), (x, 2440)] for x in range(350, 2600, 150)
    ]
    return horizontal_lines, vertical_lines
```

2.3.2 2. Extracția și Transformarea Perspectivei Tablei

Se aplică un filtru de eroziune pentru reducerea zgomotului, se detectează marginile folosind Canny Edge Detection, se identifică contururile și se selectează cel mai mare contur care ar trebui să reprezinte tabla. Apoi, se aplică o transformare de perspectivă pentru a obține o imagine "aerisită" a tablei.

Listing 2: Extracția și Transformarea Perspectivei Tablei

```
def extract_and_warp_board(image, mask, kernel_size=(3, 4), canny_thresh
    kernel = np.ones(kernel_size, np.uint8)
    mask = cv.erode(mask, kernel)
    edges = cv.Canny(mask, canny_threshold1, canny_threshold2)
    contours, _ = cv.findContours(edges, cv.RETR_EXTERNAL, cv.CHAIN_APPI
    if not contours:
        return image
    largest_contour = max(contours, key=cv.contourArea)
    perimeter = cv.arcLength(largest_contour, True)
    approx_poly = cv.approxPolyDP(largest_contour, 0.02 * perimeter, Tru
    if len(approx_poly) != 4:
        return image
    corners = order_corner_points(approx_poly.reshape(4, 2))
    destination_corners = np.array([
        [0, 0],
        [width - 1, 0],
        [width - 1, height - 1],
        [0, height - 1]
    ], dtype="float32")
    perspective_transform = cv.getPerspectiveTransform(corners, destinat
```

```
    warped = cv.warpPerspective(image, perspective_transform, (width, h
    return warped
```

2.3.3 3. Procesarea și Compararea Imaginilor

Funcția `process_and_compare_images` este responsabilă pentru compararea a două imagini (imaginea de bază, cu tabla goală și imaginea curentă) pentru a elimina fundalul din imaginea cu tabla.

Listing 3: Procesarea și Compararea Imaginilor

```
def process_and_compare_images(image1, image2, low_hsv=(14, 0, 0), high_
    def create_hsv_mask(image):
        hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
        return cv.inRange(hsv, low_hsv, high_hsv)

    # Create masks and extract boards from both images
    mask1 = create_hsv_mask(image1)
    board1 = extract_and_warp_board(image1, mask1)

    mask2 = create_hsv_mask(image2)
    board2 = extract_and_warp_board(image2, mask2)

    # Compute the absolute difference between the two boards
    difference_image = cv.absdiff(board2, board1)
    return difference_image
```

2.3.4 4. Determinarea Celulei cu Diferență Maximă

Se compară imaginea actuală a tablei cu o imagine de bază pentru a identifica diferențele, indicând poziția noii piese. Se extrage celula cu intensitatea maximă a diferenței, care reprezintă poziția piesei plasate.

Listing 4: Determinarea Celulei cu Diferență Maximă

```
def find_max_intensity_cell(thresholded_image, horizontal_lines, vertica
    row_coords = [line[0][1] for line in horizontal_lines]
    col_coords = [line[0][0] for line in vertical_lines]
    patches = [
        thresholded_image[row_coords[i]:row_coords[i+1], col_coords[j]:
```

```
    for i in range(len(row_coords) - 1)
    for j in range(len(col_coords) - 1)
]
mean_intensities = np.array([patch.mean() for patch in patches])
max_index = mean_intensities.argmax()
num_cols = len(col_coords) - 1
max_row = max_index // num_cols
max_col = max_index % num_cols
return max_row, max_col, row_coords, col_coords
```

3 Task 2: Template Matching pentru Identificarea Piesei

3.1 Obiectiv

Identificarea tipului de piesă plasată pe tablă prin compararea imaginii celulei cu șabloanele (template-urile) disponibile.

3.2 Abordare

1. **Pregătirea Șabloanelor:** Se preprocesează șabloanele prin aplicarea unui filtru HSV și redimensionarea pentru a se potrivi dimensiunilor celulelor.
2. **Compararea Șabloanelor cu Imaginea Celulei:** Se folosește tehnica de Template Matching pentru a determina cea mai bună potrivire.
3. **Identificarea Numeului Șablonului Cel Mai Potrivit:** Se selectează șablonul cu cel mai mare scor de potrivire.

3.3 Pași Detaliați

3.3.1 1. Pregătirea Șabloanelor

Se aplică un filtru HSV pentru a evidenția caracteristicile relevante ale șabloanelor. Se convertesc șabloanele la scală de gri și se redimensionează pentru a se potrivi dimensiunii celulei.

Listing 5: Aplicarea Filtrului HSV

```
def apply_hsv_mask(image, low_hsv=(0, 0, 45), high_hsv=(255, 255, 255)):
    hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv, low_hsv, high_hsv)
    masked_image = cv.bitwise_and(image, image, mask=mask)
    return masked_image
```

Listing 6: Pregătirea Șabloanelor

```
templates = []
template_names = []
template_dict = data.get('templates')
```

```
if not template_dict:
    logging.error("No templates found in images.pkl. Exiting.")
    return
for template_name, template_image in template_dict.items():
    if template_image is not None:
        masked_template = apply_hsv_mask(template_image)
        masked_template_gray = cv.cvtColor(masked_template, cv.COLOR_BGR
        resized_template = cv.resize(masked_template_gray, cell_size)
        templates.append(resized_template)
        template_names.append(template_name)
```

3.3.2 2. Compararea Șabloanelor cu Imaginea Celulei

Pentru fiecare șablon, se calculează scorul de potrivire utilizând `cv.matchTemplate`.
Se colectează scorurile și se identifică șablonul cu cel mai mare scor.

Listing 7: Compararea Șabloanelor cu Imaginea Celulei

```
match_scores = []
for template_img in templates:
    res = cv.matchTemplate(cell_image_gray, template_img, cv.TM_CCOEFF_N
    match_scores.append(res[0][0])  # Extract the matching score
best_match_index = np.argmax(match_scores)
best_match_name = template_names[best_match_index]
cleaned_name = best_match_name.split()[0]  # Clean up the template name
```

4 Task 3: Calcularea Scorului

4.1 Obiectiv

Calcularea scorului în funcție de plasarea pieselor și aplicarea multiplicatorilor specifici pentru anumite celule.

4.2 Abordare

1. **Definirea Celulelor cu Multiplicatori:** Identificăm celulele care oferă multiplicatori dublu sau triplu.

2. **Actualizarea Gridului de Joc:** Pe baza mutărilor, actualizăm gridul pentru a reflecta pozițiile pieselor.

3. **Calcularea Scorului:** Verificăm dacă plasarea unei piese formează o ecuație validă cu piesele adiacente și aplicăm multiplicatorii corespunzători.

4.3 Pași Detaliați

4.3.1 1. Definirea Celulelor cu Multiplicatori

Se definesc coordonatele celulelor care oferă multiplicatori dublu sau triplu.

Listing 8: Definirea Celulelor cu Multiplicatori

```
double_multiplier_cells = [
    (1, 1), (2, 2), (3, 3), (4, 4), (9, 9),
    (10, 10), (11, 11), (12, 12),
    (1, 12), (2, 11), (3, 10), (4, 9),
    (9, 4), (10, 3), (11, 2), (12, 1)
]

triple_multiplier_cells = [
    (0, 0), (0, 13), (13, 0), (13, 13),
    (6, 0), (7, 0), (0, 6), (0, 7),
    (13, 6), (13, 7), (6, 13), (7, 13)
]
```

4.3.2 2. Actualizarea Gridului de Joc

Gridul este reprezentat ca o listă bidimensională, inițializat cu spații libere. Pozitiile din centrul grid-ului, în jocul de mathable, au valorile următoare:

Listing 9: Inițializarea Gridului de Joc

```
grid_size = 14
game_grid = [[" " for _ in range(grid_size)] for _ in range(grid_size)]

game_grid[6][6] = "1"
game_grid[7][7] = "2"
game_grid[6][7] = "3"
```

```
game_grid[7][6] = "4"
```

4.3.3 3. Calcularea Scorului

Pentru fiecare mutare, se verifică dacă plasarea piesei formează o ecuație validă cu piesele adiacente. Se aplică multiplicatorii specifici celulei unde a fost plasată piesa. Scorul local este adăugat la scorul total al jocului.

Listing 10: Calcularea Scorului

```
total_score = 0  # Initialize total score

def calculate_equations_and_score(game_grid, row, col, piece_value_str):
    nonlocal total_score
    piece_value = int(piece_value_str)  # Convert piece value to integer
    local_score = 0  # Initialize local score
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Directions: up, d

    for dr, dc in directions:
        r1, c1 = row + dr, col + dc
        r2, c2 = row + 2 * dr, col + 2 * dc
        if 0 <= r1 < grid_size and 0 <= c1 < grid_size and 0 <= r2 < gri
            try:
                val1 = int(game_grid[r1][c1])
                val2 = int(game_grid[r2][c2])
            except ValueError:
                continue  # Skip if cells do not contain integers

            # Check if placing the piece creates a valid equation
            if val1 + val2 == piece_value or val1 - val2 == piece_value
               val1 * val2 == piece_value or (val2 != 0 and val1 // val2
                # Valid equation found
                local_score += piece_value

    # Apply multipliers based on cell position
    if (row, col) in double_multiplier_cells:
        local_score *= 2
    elif (row, col) in triple_multiplier_cells:
        local_score *= 3
```

```
    # Ensure the minimum score is at least the piece value
    if local_score == 0:
        local_score = piece_value

    # Update the total score
    total_score += local_score
    return local_score
```

4.3.4 4. Procesarea Mutărilor și Calcularea Scorului

Se parcurg fișierele de mutări și se actualizează gridul în consecință. Pentru fiecare mutare, se calculează scorul și se actualizează fișierul de scoruri.

Listing 11: Procesarea Mutărilor și Calcularea Scorului

```
with open(turns_file_path, 'r') as file:
    turns = file.readlines()

turn_numbers = [int(line.split()[1]) for line in turns]
differences = [turn_numbers[i+1] - turn_numbers[i] for i in range(len(tu
differences.append(51 - turn_numbers[-1])  # Add the remaining turns

file_list = differences
folder_path = output_path  # 'annotations' folder

def position_to_indices(position):
    row = int(position[:-1]) - 1  # Convert row number to zero-based inc
    col = ord(position[-1].upper()) - ord('A')  # Convert column letter
    return row, col

with open(output_file, 'w') as outfile:
    file_counter = 1  # Counter for the file names
    for turn, count in zip(turns, file_list):
        turn_score = 0  # Score for the current turn
        for _ in range(count):
            file_name = f"1_{file_counter:02}.txt"  # Format as 1_01.txt
            file_path = folder_path / file_name
```

```
            if file_path.exists():
                with open(file_path, 'r') as file:
                    content = file.read().strip()
                    position, piece_value_str = content.split()
# Split into position and piece value
                    row, col = position_to_indices(position)
# Convert position to indices

                    # Place the piece on the grid
                    game_grid[row][col] = piece_value_str

                    # Calculate score for this move
                    step_score = calculate_equations_and_score(game_gri
            else:
                logging.warning(f"File {file_path} does not exist. Skipp
                step_score = 0  # Assign zero score if file does not ex

            file_counter += 1  # Increment the file counter
            turn_score += step_score  # Add to the turn score

        # Write the turn information and score to the output file
        player, turn_number = turn.strip().split()
        outfile.write(f"{player} {turn_number} {turn_score}\n")
```

5 Concluzie

Această soluție oferă o abordare comprehensivă pentru detectarea poziției pieselor pe o tablă de joc, identificarea tipului de piesă prin template matching și calcularea scorului în funcție de regulile jocului.