Politechnika Śląska

6 stycznia 2025,
Gliwice

Laboratorium ZAOWR

Rok akademicki	Termin	Rodzaj studiów	Kierunek	Prowadzący	Grupa
2024/2025	Czwartek, 10:00-13:15	Stacjonarne	Informatyka	Dr inż. Marcin Paszkuta Mgr inż. Mateusz Płonka	IGT

Ćwiczenie nr 3

Temat: Odtwarzanie trójwymiarowej sceny na podstawie wielu perspektyw

Informatyka, IGT

Maksymilian Kisiel

Spis treści

1 Zadanie 1 - Wyznaczanie mapy dysparycji z wykorzystaniem StereoBM

Aby wykonać zadania od 1 do 3, przygotowano funkcję, która pozwala na wybór metody wyznaczania mapy dysparycji. Każda z metod współdzieli ten sam początek i koniec kodu, ale w zależności od wybranej metody wykonywany jest odpowiedni kod w bloku `if-else`. Zarówno w tej sekcji, jak i kolejnych, zamieszczono jedynie fragment kodu specyficzny dla danej metody (Kod 1, 2 i 3). Pełny kod funkcji znajduje się w pliku `calculate_disparity_map.py`, który dostępny jest w serwisie GitHub (pliki źródłowe zostały również dołączone do sprawozdania - w takim przypadku wspomniany plik znajduje się w katalogu `image_processing`). Przygotowana funkcja pozwala również na zapis wyznaczonej mapy dysparycji do pliku.

```
if disparityCalculationMethod == "bm":
    calculationMethod = "StereoBM"
    print(Fore.GREEN + f"\nComputing disparity map using '{calculationMethod}'...")
    # Create StereoBM object
    stereoBM = cv.StereoBM.create(numDisparities=numDisparities, blockSize=blockSize)
    # Compute disparity map
    disparityMap = stereoBM.compute(img_left, img_right)
```

Kod 1: Fragment kodu odpowiedzialny za wyznaczanie mapy dysparycji metodą **StereoBM**

2 Zadanie 2 - Wyznaczanie mapy dysparycji z wykorzystaniem StereoSGBM

```
elif disparityCalculationMethod == "sgbm":
    calculationMethod = "StereoSGBM"
    print(Fore.GREEN + f"\nComputing disparity map using '{calculationMethod}'...")
    # Create StereoSGBM object with initial/default parameters
    stereoSGBM = cv.StereoSGBM.create(minDisparity=minDisparity,
    numDisparities=numDisparities, blockSize=blockSize,
        P1=(8 * 3 * blockSize ** 2),  # Penalty on the disparity change
        P2=(32 * 3 * blockSize ** 2),  # Stronger penalty for larger changes in
    disparity
        disp12MaxDiff=2,  # Maximum allowed disparity difference
        preFilterCap=63,  # Truncation value for prefiltered image pixels
        uniquenessRatio=15,  # Margin by which the best (minimum) computed cost
    function value
        speckleWindowSize=100,  # Maximum size of smooth disparity regions to
    consider noise
        speckleRange=1,  # Maximum disparity variation within smooth disparity regions
    )
    disparityMap = stereoSGBM.compute(img_left, img_right)
```

Kod 2: Fragment kodu odpowiedzialny za wyznaczanie mapy dysparycji metodą **StereoSGBM**

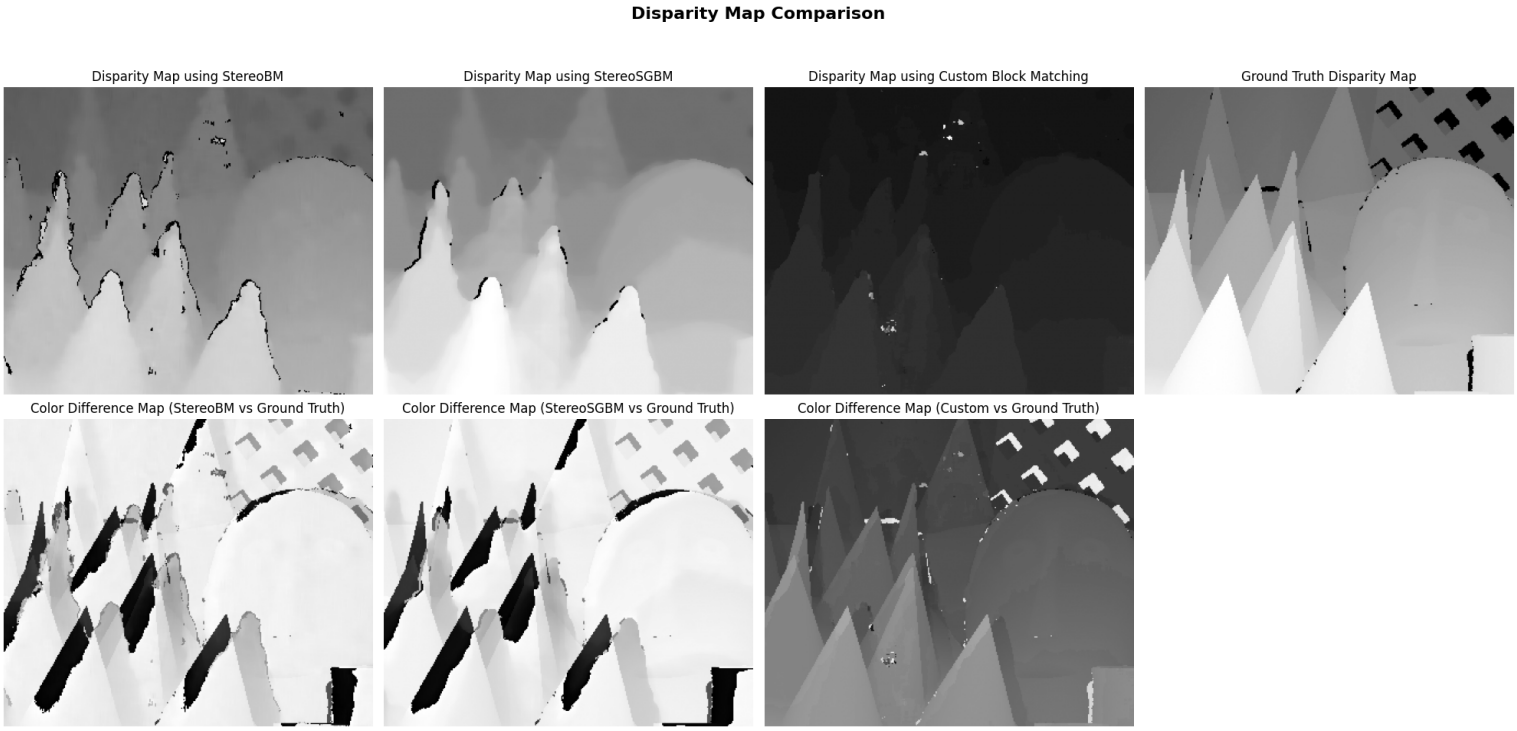
3 Zadanie 3 - Implementacja własnej funkcji dopasowywania stereo

```
elif disparityCalculationMethod == "custom": # SSD
    calculationMethod = "Custom Method (SSD, left to right)"
    print(Fore.GREEN + f"\nComputing disparity map using '{calculationMethod}'...")
    windowHeight, windowWidth = windowSize
    height, width = img_left.shape
    halfWindowHeight = windowHeight // 2
    halfWindowWidth = windowWidth // 2
    # Initialize the disparity map
    disparityMap = np.zeros((height, width), dtype=np.float32)
    for dy in tqdm(range(halfWindowHeight, height - halfWindowHeight),
    desc=Style.RESET_ALL + "Searching for the best SSD match...", dynamic_ncols=True,
    bar_format="{l_bar}{bar}{r_bar}", colour="green", file=stdout, position=0):
        for dx in range(halfWindowWidth, width - halfWindowWidth):
            # Extract block from the right image
            template = img_right[dy - halfWindowHeight : dy + halfWindowHeight + 1,
    dx - halfWindowWidth : dx + halfWindowWidth + 1]
            # Initialize the best SSD and disparity
            minSsd = float('inf')
            bestDisparity = 0
            # Search over the disparity range
            for offset in range(min(maxDisparity, width - dx - halfWindowWidth)):
                # Extract block from the left image
                roi = img_left[dy - halfWindowHeight : dy + halfWindowHeight + 1, dx
    - halfWindowWidth + offset : dx + halfWindowWidth + offset + 1]
                # Compute SSD
                ssd = np.sum((template - roi) ** 2)
                # Update the best match
                if ssd < minSsd:
                    minSsd = ssd
                    bestDisparity = offset
            # Store the best disparity
            disparityMap[dy, dx] = bestDisparity
elif disparityCalculationMethod == "custom2":
    calculationMethod = "Custom Method 2 (SSD, stereo block matching)"
    print(Fore.GREEN + f"\nComputing disparity map using '{calculationMethod}'...")
    # Initialize disparity map
    height, width = img_left.shape
    disparityMap = np.zeros((height, width), dtype=np.float32)
    halfBlock = blockSize // 2
    # Pad images to handle borders
    padded_left = cv.copyMakeBorder(img_left, halfBlock, halfBlock, halfBlock,
    halfBlock, cv.BORDER_CONSTANT, value=0)
    padded_right = cv.copyMakeBorder(img_right, halfBlock, halfBlock, halfBlock,
    halfBlock, cv.BORDER_CONSTANT, value=0)
    # Compute disparity map
    for y in tqdm(range(halfBlock, height + halfBlock), desc=Style.RESET_ALL +
    "Searching for the best SSD match...", dynamic_ncols=True,
    bar_format="{l_bar}{bar}{r_bar}", colour="green", file=stdout, position=0):
        for x in range(halfBlock, width + halfBlock):
            # Extract block from left image
            block_left = padded_left[y - halfBlock:y + halfBlock + 1, x - halfBlock:x
    + halfBlock + 1]
            # Initialize variables to find the best match
            minSsd = float('inf')
            bestDisparity = 0
            # Search for best match in disparity range
            for d in range(maxDisparity):
                # Compute the position in the right image
                x_shifted = x - d
                if x_shifted - halfBlock < 0:
                    continue
                # Extract block from right image
                right_block = padded_right[y - halfBlock:y + halfBlock + 1, x_shifted
    - halfBlock:x_shifted + halfBlock + 1]
                # Compute the sum of squared differences (SSD)
                ssd = np.sum((block_left - right_block) ** 2)
                # Update best match
                if ssd < minSsd:
                    minSsd = ssd
                    bestDisparity = d
            # Store best disparity
            disparityMap[y - halfBlock, x - halfBlock] = bestDisparity
```

Kod 3: Fragment kodu odpowiedzialny za wyznaczanie mapy dysparycji metodą **Custom1** i **Custom2**

4 Zadanie 4 - Analiza wyników i wizualizacja błędów

Otrzymane wyniki z zadań od 1 do 3 zostały zgrupowane jako jeden plot, który zawiera zarówno mapy desparycji wyznaczone za pomocą różnych metod (StereoBM, StereoSGBM, Custom), jak i mapy różnic (wizualizacja błędów). Te mapy zostały zestawione z rzeczywistą mapą dysparycji (*ang. ground truth*) i widoczne są poniżej (Rys. 1).



Rys. 1: Porównanie różnych metod wyznaczania mapy dysparycji

Poniżej znajduje się tabela (Tab. 1) porównująca wyniki **MSE** (*Mean Square Error* - błąd średnio-kwadratowy) i **SSIM** (*Structural similarity index measure* - wskaźniki podobieństwa strukturalnego) dla różnych metod wyznaczania mapy dysparycji (otrzymana mapa w porównaniu do **ground truth**).

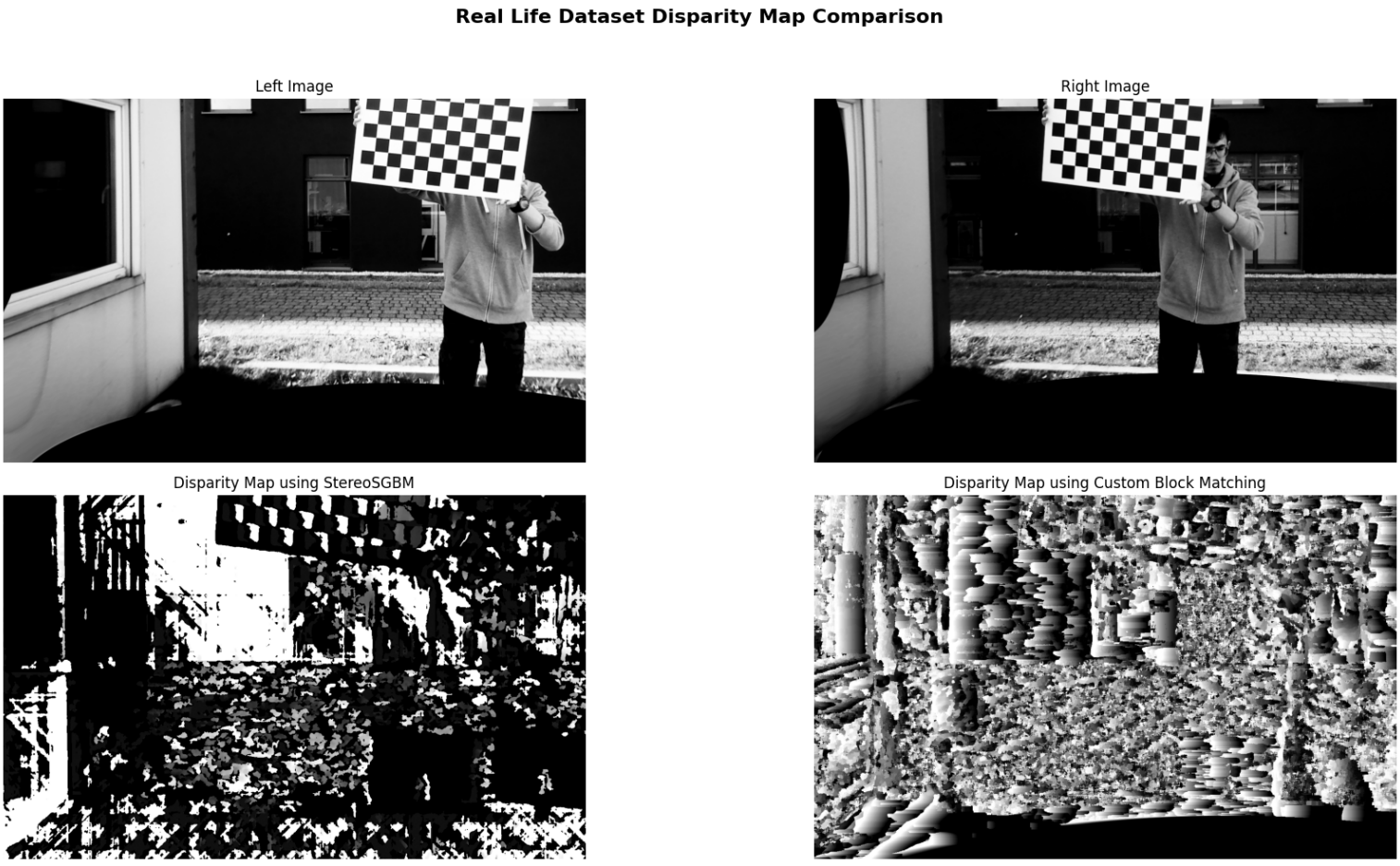
Metoda / Wskaźnik	**StereoBM**	**StereoSGBM**	**Custom**
MSE	101.38	113.26	102.74
SSIM	0.71	0.76	0.38

Tab. 1: Tabela przedstawiająca wartości wskaźników **MSE** i **SSIM** dla różnych metod

5 Zadanie 5 - Wyznaczanie mapy dysparycji z rektyfikacją obrazu

Aby wykonać to zadanie, wykorzystano obrazy ze zbioru kalibracyjnego, który został dostarczony przez prowadzącego. Dla wybranych zdjęć stereo (zrektyfikowanych) wyznaczono mapy dysparyzji za pomocą metod **StereoSGBM** oraz **Custom** (implementacja własna). Następnie zestawiono obie mapy na jednym wykresie (Rys. 2), aby łatwiej można było porównać otrzymane wyniki.

Poniżej znajduje się również fragment kodu opowiedzialny za wywołanie odpowiednich funkcji (wyznaczanie mapy i tworzenie zestawienia, Kod 4), żeby przetestować przygotowane implementacje na rzeczywistych obrazach.



Rys. 2: Porównanie różnych metod wyznaczania mapy dysparycji

```
# Calculate the disparity map using SGBM
pbar.set_description(tasks[9])
disparityMapSGBM = zw.calculate_disparity_map(
    leftImagePath=img_left_real_life,
    rightImagePath=img_right_real_life,
    blockSize=7,
    numDisparities=16,
    minDisparity=8,
    disparityCalculationMethod="sgbm",
    saveDisparityMap=saveDisparityMap,
    saveDisparityMapPath=os.path.join(saveDisparityMapPath,
    "disparity_map_SGBM_RL.png"),
    showDisparityMap=showMaps
)
pbar.update(1)

# Calculate the disparity map using custom block matching
pbar.set_description(tasks[10])
disparityMapCustom = zw.calculate_disparity_map(
    leftImagePath=img_left_real_life,
    rightImagePath=img_right_real_life,
    maxDisparity=16,
    windowSize=(9, 9),
    disparityCalculationMethod="custom",
    saveDisparityMap=saveDisparityMap,
    saveDisparityMapPath=os.path.join(saveDisparityMapPath,
    "disparity_map_custom_RL.png"),
    showDisparityMap=showMaps
)
pbar.update(1)

# Crop the images
pbar.set_description(tasks[11])
croppingPercentage = 0.75

disparityMapSGBM = zw.crop_image(disparityMapSGBM, cropPercentage=croppingPercentage)
disparityMapCustom = zw.crop_image(disparityMapCustom,
    cropPercentage=croppingPercentage)
pbar.update(1)

# Plot the comparison
pbar.set_description(tasks[12])
zw.plot_disparity_map_comparison(
    disparityMapBM=cv2.imread(img_left_real_life, cv2.IMREAD_GRAYSCALE), # top left
    disparityMapSGBM=cv2.imread(img_right_real_life, cv2.IMREAD_GRAYSCALE), # top
    right
    disparityMapCustom=disparityMapSGBM, # bottom left
    groundTruth=disparityMapCustom, # bottom right
    saveComparison=saveComparison,
    savePath=saveComparisonPath_RL,
    titleMain="Real Life Dataset Disparity Map Comparison",
    title1="Left Image",
    title2="Right Image",
    title3="Disparity Map using StereoSGBM",
    title4="Disparity Map using Custom Block Matching",
)
pbar.update(1)
```

Kod 4: Fragment kodu odpowiedzialny za wyznaczanie mapy dysparycji i tworzenie zestawienia

6 Kod

Przygotowana na potrzeby tego laboratorium paczka, została zaktualizowana, aby wykonywać wszystkie wymagane zadania z tego laboratorium:

- wyznaczanie mapy dysparycji z wykorzystaniem StereoBM,
- wyznaczanie mapy dysparycji z wykorzystaniem StereoSGBM,
- implementacja własnej funkcji dopasowywania stereo,
- wizualizacja błędów,
- wyznaczanie mapy dysparycji z rektyfikacją obrazu.

Kod paczki jest dostępny w serwisie GitHub, jednak kopia kodu źródłowego została załączona do sprawozdania w celu łatwiejszej weryfikacji wyników.