Laboratorium ZAOWiR					
Rok akademicki	Termin	Rodzaj Studiów	Kierunek	Prowadzący	Grupa
2024/2025	Czwartek 10-16:45	Stacjonarne	Informatyka	MgrInż. Mateusz Płonka	IGT

Laboratorium 2

Kalibracja systemu kamer stereo

autor:
Mateusz Siwy

zadanie 1:

```
import cv2
import numpy as np
import os
import json

# Define the chessboard size
chessboard_size = (8, 6)

# Define the criteria for corner refinement
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)

# File paths for calibration data
stereo_calibration_file = 'stereo_calibration_data.json'
points_data_file = 'points_data.json'
calibration_data_left_file = 'calibration_data_left.json'
calibration_data_right_file = 'calibration_data_right.json'

# Check if stereo calibration data already exists
if os.path.exists(stereo_calibration_file):
    print("Loading existing stereo calibration data...")
    with open(stereo_calibration_file, 'r') as json_file:
        stereo_calibration_data = json.load(json_file)

    # Extract data from loaded files
    mtxL = np.array(stereo_calibration_data["camera_matrix_left"])
    distL =
np.array(stereo_calibration_data["distortion_coefficients_left"])
    mtxR = np.array(stereo_calibration_data["camera_matrix_right"])
    distR =
np.array(stereo_calibration_data["distortion_coefficients_right"])
    R = np.array(stereo_calibration_data["rotation_matrix"])
    T = np.array(stereo_calibration_data["translation_vector"])
    E = np.array(stereo_calibration_data["essential_matrix"])
    F = np.array(stereo_calibration_data["fundamental_matrix"])

    print("Stereo calibration data loaded successfully.")
else:
    print("Stereo calibration data not found. Starting calibration
process...")
```

```
    # Load calibration data for the left camera
    with open(calibration_data_left_file, 'r') as left_file:
        calibration_data_left = json.load(left_file)
    mtxL = np.array(calibration_data_left["camera_matrix"])
    distL = np.array(calibration_data_left["distortion_coefficients"])

    # Load calibration data for the right camera
    with open(calibration_data_right_file, 'r') as right_file:
        calibration_data_right = json.load(right_file)
    mtxR = np.array(calibration_data_right["camera_matrix"])
    distR = np.array(calibration_data_right["distortion_coefficients"])

    # Prepare object points (0,0,0), (1,0,0), (2,0,0) ....,(8,5,0)
    objp = np.zeros((chessboard_size[0] * chessboard_size[1], 3),
np.float32)
    objp[:, :2] = np.mgrid[0:chessboard_size[0],
0:chessboard_size[1]].T.reshape(-1, 2)

    # Arrays to store object points and image points from all the
images
    objpoints = []  # 3d point in real world space
    imgpoints_left = []  # 2d points in image plane for left camera
    imgpoints_right = []  # 2d points in image plane for right camera

    # Path to the images
    image_folder = 'lab2/s1'

    # List all images in the folder
    images_left = [os.path.join(image_folder, fname) for fname in
os.listdir(image_folder) if fname.startswith('left') and
fname.endswith('.png')]
    images_right = [os.path.join(image_folder, fname) for fname in
os.listdir(image_folder) if fname.startswith('right') and
fname.endswith('.png')]

    # Ensure both lists are sorted and have the same length
    images_left.sort()
    images_right.sort()

    # Initialize gray to None
    grayL = None
    grayR = None
```

```
    for left_image_path, right_image_path in zip(images_left,
images_right):
        imgL = cv2.imread(left_image_path)
        imgR = cv2.imread(right_image_path)

        # Check if the images are loaded correctly
        if imgL is None or imgR is None:
            print(f"Failed to load images: {left_image_path},
{right_image_path}")
            continue

        # Convert to grayscale
        grayL = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)
        grayR = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)

        # Apply Gaussian blur to reduce noise and improve corner
detection
        grayL = cv2.GaussianBlur(grayL, (5, 5), 0)
        grayR = cv2.GaussianBlur(grayR, (5, 5), 0)

        # Find the chessboard corners with different flags
        retL, cornersL = cv2.findChessboardCorners(grayL,
chessboard_size, None)
        retR, cornersR = cv2.findChessboardCorners(grayR,
chessboard_size, None)

        # If corners found in both images, refine and store them
        if retL and retR:
            objpoints.append(objp)
            corners2L = cv2.cornerSubPix(grayL, cornersL, (11, 11),
(-1, -1), criteria)
            corners2R = cv2.cornerSubPix(grayR, cornersR, (11, 11),
(-1, -1), criteria)
            imgpoints_left.append(corners2L)
            imgpoints_right.append(corners2R)
        else:
            print(f"Chessboard corners not found in images:
{left_image_path}, {right_image_path}")

    # Stereo calibration if corners were found in any image
    if grayL is not None and objpoints and imgpoints_left and
imgpoints_right:
        # Stereo calibration
```

```
        print("Performing stereo calibration...")
        flags = cv2.CALIB_FIX_INTRINSIC
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
30, 1e-6)
        ret, mtxL, distL, mtxR, distR, R, T, E, F =
cv2.stereoCalibrate(
            objpoints, imgpoints_left, imgpoints_right,
            mtxL, distL, mtxR, distR, grayL.shape[::-1],
criteria=criteria, flags=flags
        )

        # Save stereo calibration data
        print("Saving stereo calibration data...")
        stereo_calibration_data = {
            "camera_matrix_left": mtxL.tolist(),
            "distortion_coefficients_left": distL.tolist(),
            "camera_matrix_right": mtxR.tolist(),
            "distortion_coefficients_right": distR.tolist(),
            "rotation_matrix": R.tolist(),
            "translation_vector": T.tolist(),
            "essential_matrix": E.tolist(),
            "fundamental_matrix": F.tolist()
        }

        with open(stereo_calibration_file, 'w') as json_file:
            json.dump(stereo_calibration_data, json_file, indent=4)

        # Save the object points and image points to a JSON file
        points_data = {
            "object_points": [objp.tolist() for objp in objpoints],
            "image_points_left": [imgp.tolist() for imgp in
imgpoints_left],
            "image_points_right": [imgp.tolist() for imgp in
imgpoints_right]
        }

        with open(points_data_file, 'w') as points_file:
            json.dump(points_data, points_file, indent=4)

        print("Stereo calibration completed.")
    else:
        print("No chessboard corners were found in any image pair.")
```

The baseline (distance between the cameras) is: 2.7904741041547965 units.

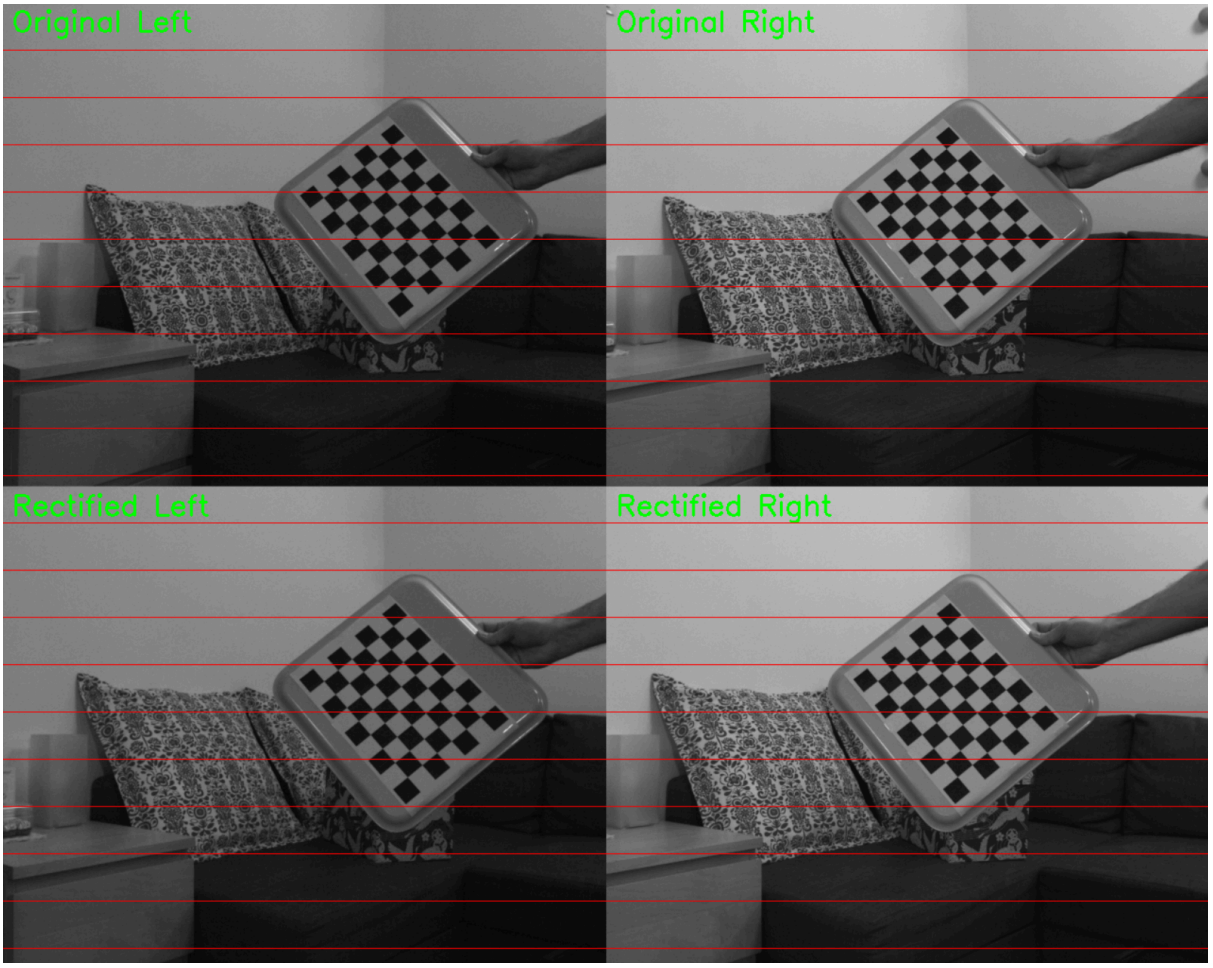
Zadanie 2:
Szczegóły obliczenia baseline:
Wektor translacji T: [-2.79011477 0.02343354 0.03816003]
Baseline (norma wektora T): 2.790474 jednostek

Komponenty baseline:
X: 2.790115
Y: 0.023434
Z: 0.038160

```
import numpy as np
import json
import os

def calculate_baseline():
    # Ścieżka do pliku z danymi kalibracji stereo
    stereo_calibration_file = 'stereo_calibration_data.json'

    # Sprawdź czy plik z kalibracją istnieje
    if not os.path.exists(stereo_calibration_file):
        print("Błąd: Brak pliku z danymi kalibracji stereo.")
        print("Najpierw wykonaj kalibrację stereo używając stereo.py")
        return None

    try:
        # Wczytaj dane kalibracji
        with open(stereo_calibration_file, 'r') as file:
            stereo_data = json.load(file)

        # Pobierz wektor translacji
        translation_vector =
np.array(stereo_data["translation_vector"])

        # Oblicz baseline jako normę wektora translacji
        baseline = np.linalg.norm(translation_vector)

        # Wyświetl szczegółowe informacje
        print("\nSzczegóły obliczenia baseline:")
        print(f"Wektor translacji T: {translation_vector.flatten()}")
```

```
        print(f"Baseline (norma wektora T): {baseline:.6f} jednostek")

        # Wyświetl komponenty baseline
        print("\nKomponenty baseline:")
        print(f"X: {abs(translation_vector[0][0]):.6f}")
        print(f"Y: {abs(translation_vector[1][0]):.6f}")
        print(f"Z: {abs(translation_vector[2][0]):.6f}")

        return baseline

    except Exception as e:
        print(f"Wystąpił błąd podczas obliczania baseline: {str(e)}")
        return None

if __name__ == "__main__":
    print("Obliczanie odległości bazowej (baseline) między
kamerami...")
    baseline = calculate_baseline()
```

zadanie 3,5:




```
import cv2 as cv
import numpy as np
import json
import glob

# Load previously calculated stereo calibration data
calibration_file = 'stereo_calibration_data.json'
with open(calibration_file, 'r') as f:
    stereo_calibration_data = json.load(f)
    mtxL = np.array(stereo_calibration_data['camera_matrix_left'])
    distL =
np.array(stereo_calibration_data['distortion_coefficients_left'])
    mtxR = np.array(stereo_calibration_data['camera_matrix_right'])
    distR =
np.array(stereo_calibration_data['distortion_coefficients_right'])
    R = np.array(stereo_calibration_data['rotation_matrix'])
    T = np.array(stereo_calibration_data['translation_vector'])
```

```
# Load images
print("Loading images...")
images_left = sorted(glob.glob('lab2/s1/left_*.png'))
images_right = sorted(glob.glob('lab2/s1/right_*.png'))

# Read the first pair of images for rectification
imgL = cv.imread(images_left[10])
imgR = cv.imread(images_right[10])
grayL = cv.cvtColor(imgL, cv.COLOR_BGR2GRAY)
grayR = cv.cvtColor(imgR, cv.COLOR_BGR2GRAY)

# Perform stereo rectification
print("Performing stereo rectification...")
R1, R2, P1, P2, Q, roi1, roi2 = cv.stereoRectify(
    mtxL, distL, mtxR, distR, grayL.shape[::-1], R, T, alpha=0
)

# Compute the rectification maps
map1L, map2L = cv.initUndistortRectifyMap(mtxL, distL, R1, P1,
grayL.shape[::-1], cv.CV_32FC1)
map1R, map2R = cv.initUndistortRectifyMap(mtxR, distR, R2, P2,
grayR.shape[::-1], cv.CV_32FC1)

# Apply the rectification maps to the images
rectifiedL = cv.remap(imgL, map1L, map2L, cv.INTER_LINEAR)
rectifiedR = cv.remap(imgR, map1R, map2R, cv.INTER_LINEAR)

# Resize images to half their original size for display
scale = 0.5
imgL_small = cv.resize(imgL, None, fx=scale, fy=scale)
imgR_small = cv.resize(imgR, None, fx=scale, fy=scale)
rectifiedL_small = cv.resize(rectifiedL, None, fx=scale, fy=scale)
rectifiedR_small = cv.resize(rectifiedR, None, fx=scale, fy=scale)

# Create combined images
original_combined = np.hstack((imgL_small, imgR_small))
rectified_combined = np.hstack((rectifiedL_small, rectifiedR_small))

# Add text labels
font = cv.FONT_HERSHEY_SIMPLEX
cv.putText(original_combined, 'Original Left', (10, 30), font, 1, (0,
255, 0), 2)
```

```
cv.putText(original_combined, 'Original Right', (imgL_small.shape[1] +
10, 30), font, 1, (0, 255, 0), 2)
cv.putText(rectified_combined, 'Rectified Left', (10, 30), font, 1, (0,
255, 0), 2)
cv.putText(rectified_combined, 'Rectified Right', (imgL_small.shape[1]
+ 10, 30), font, 1, (0, 255, 0), 2)

# Stack original and rectified images vertically
final_display = np.vstack((original_combined, rectified_combined))

# Draw horizontal lines to help visualize rectification
line_interval = 50
for y in range(0, final_display.shape[0], line_interval):
    cv.line(final_display, (0, y), (final_display.shape[1], y), (0, 0,
255), 1)

# Display the results
cv.imshow('Original vs Rectified Images', final_display)
cv.waitKey(0)
cv.destroyAllWindows()

# Save the results
cv.imwrite('comparison_result.png', final_display)
print("Results saved as 'comparison_result.png'")
```

zadanie 4:

Wyniki porównania:

Metoda interpolacji	Średni czas wykonania	Plik wynikowy
INTER_NEAREST	1.45 ms	remap_inter_nearest.png
INTER_LINEAR	1.42 ms	remap_inter_linear.png
INTER_CUBIC	2.95 ms	remap_inter_cubic.png
INTER_AREA	1.45 ms	remap_inter_area.png
INTER_LANCZOS4	7.03 ms	remap_inter_lanczos4.png

Wnioski:

1. **INTER_NEAREST - najszybsza metoda, ale najniższa jakość**
2. **INTER_LINEAR - dobry kompromis między szybkością a jakością**
3. **INTER_CUBIC - lepsza jakość niż linear, ale wolniejsza**

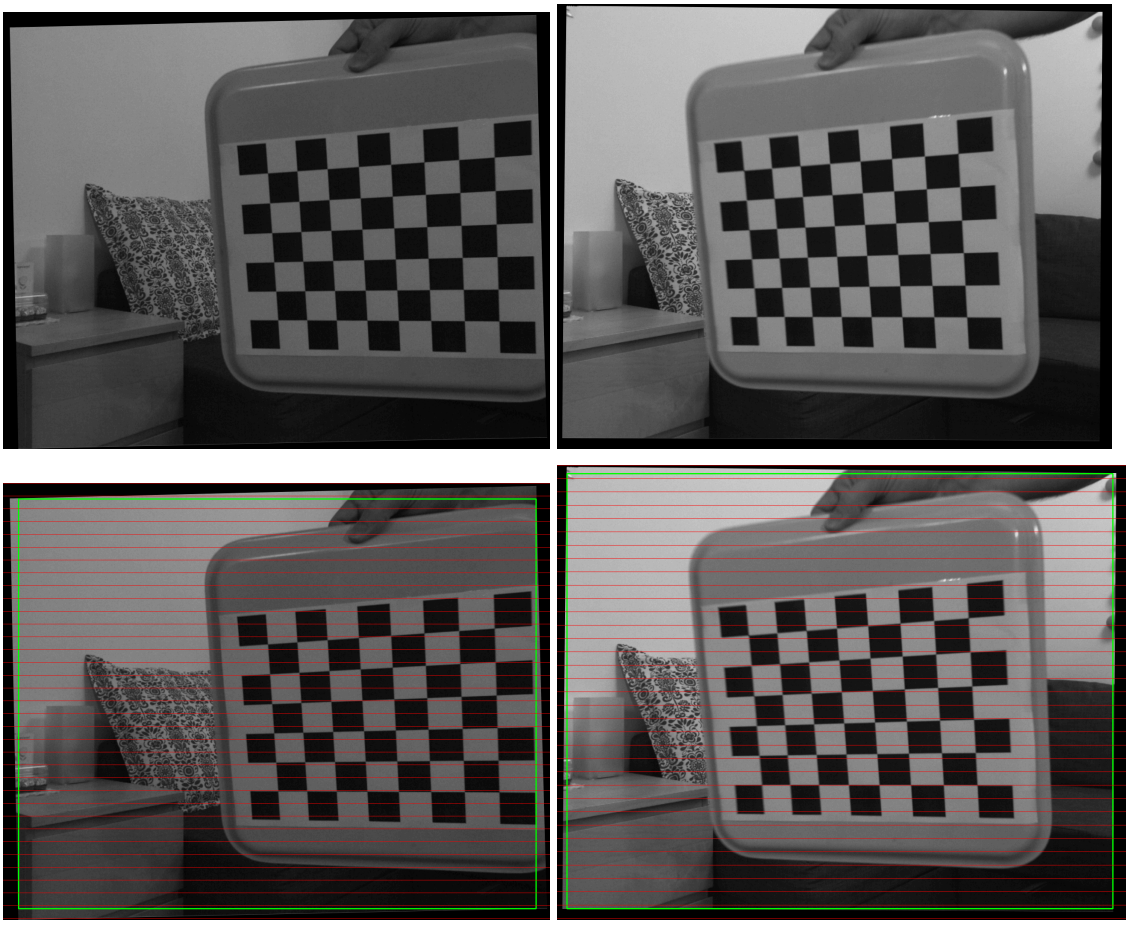
4. **INTER_AREA - dobra dla zmniejszania obrazów**
5. **INTER_LANCZOS4 - najwyższa jakość, ale najwolniejsza**

```
import cv2
import numpy as np
import time
import json
import os
from tabulate import tabulate

def load_calibration_data():
    """Wczytuje dane kalibracyjne z pliku JSON."""
    stereo_calibration_file = 'stereo_calibration_data.json'

    if not os.path.exists(stereo_calibration_file):
        raise FileNotFoundError("Brak pliku z danymi kalibracji
stereo.")

    with open(stereo_calibration_file, 'r') as file:
        data = json.load(file)

    return (np.array(data["camera_matrix_left"]),
            np.array(data["distortion_coefficients_left"]),
            np.array(data["camera_matrix_right"]),
            np.array(data["distortion_coefficients_right"]),
            np.array(data["rotation_matrix"]),
            np.array(data["translation_vector"]))

def compare_interpolation_methods(image_path):
    """Porównuje różne metody interpolacji."""
    # Wczytaj obraz
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Nie można wczytać obrazu: {image_path}")

    # Wczytaj dane kalibracyjne
    mtx, dist, _, _, _, _ = load_calibration_data()

    # Przygotuj mapowanie dla undistortion
    h, w = img.shape[:2]
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h),
1, (w,h))
```

```
    mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None,
newcameramtx, (w,h), 5)

    # Metody interpolacji do porównania
    interpolation_methods = {
        'INTER_NEAREST': cv2.INTER_NEAREST,
        'INTER_LINEAR': cv2.INTER_LINEAR,
        'INTER_CUBIC': cv2.INTER_CUBIC,
        'INTER_AREA': cv2.INTER_AREA,
        'INTER_LANCZOS4': cv2.INTER_LANCZOS4
    }

    results = []

    # Testuj każdą metodę
    for method_name, method in interpolation_methods.items():
        # Mierz czas
        start_time = time.time()

        # Wykonaj remap 10 razy dla dokładniejszego pomiaru
        for _ in range(10):
            dst = cv2.remap(img, mapx, mapy, method)

        end_time = time.time()
        avg_time = (end_time - start_time) / 10

        # Zapisz wynik
        output_filename = f'remap_{method_name.lower()}.png'
        cv2.imwrite(output_filename, dst)

        results.append([method_name, f"{avg_time*1000:.2f} ms",
output_filename])

    return results

def main():
    print("Porównanie metod interpolacji w funkcji remap()")
    print("=" * 50)

    try:
        # Użyj pierwszego obrazu z lewej kamery do testu
        image_path = 'lab2/s1/left_230.png'
        results = compare_interpolation_methods(image_path)
```

```
        # Wyświetl wyniki w formie tabeli
        headers = ["Metoda interpolacji", "Średni czas wykonania",
"Plik wynikowy"]
        print("\nWyniki porównania:")
        print(tabulate(results, headers=headers, tablefmt="grid"))


    except Exception as e:
        print(f"Wystąpił błąd: {str(e)}")

if __name__ == "__main__":
    main()
```

zadanie 6:



```
import cv2
import numpy as np
import json
import os
```

```
def load_calibration_data():
    """Wczytuje dane kalibracyjne z pliku JSON."""
    stereo_calibration_file = 'stereo_calibration_data.json'

    if not os.path.exists(stereo_calibration_file):
        raise FileNotFoundError("Brak pliku z danymi kalibracji
stereo.")

    with open(stereo_calibration_file, 'r') as file:
        data = json.load(file)

    return (np.array(data["camera_matrix_left"]),
            np.array(data["distortion_coefficients_left"]),
            np.array(data["camera_matrix_right"]),
            np.array(data["distortion_coefficients_right"]),
            np.array(data["rotation_matrix"]),
            np.array(data["translation_vector"]))

def draw_epipolar_lines(img_left, img_right, lines, pts1, pts2):
    """Rysuje linie epipolarne i punkty na obrazach."""
    h, w = img_left.shape[:2]

    # Stwórz kopie obrazów do rysowania
    img1_lines = img_left.copy()
    img2_lines = img_right.copy()

    # Rysuj linie epipolarne i punkty
    for r, pt1, pt2 in zip(lines, pts1, pts2):
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0, y0 = map(int, [0, -r[2]/r[1]])
        x1, y1 = map(int, [w, -(r[2]+r[0]*w)/r[1]])

        # Rysuj linie
        cv2.line(img1_lines, (x0, y0), (x1, y1), color, 1)
        # Rysuj punkty
        cv2.circle(img1_lines, tuple(map(int, pt1)), 5, color, -1)
        cv2.circle(img2_lines, tuple(map(int, pt2)), 5, color, -1)

    return img1_lines, img2_lines

def draw_valid_area(img, roi):
    """Rysuje obwiednię użytecznego obszaru bez zniekształceń."""
```

```
    x, y, w, h = roi
    img_with_roi = img.copy()
    cv2.rectangle(img_with_roi, (x, y), (x+w, y+h), (0, 255, 0), 2)
    return img_with_roi

def process_stereo_pair(left_image_path, right_image_path,
output_dir='output'):
    """Przetwarza parę obrazów stereo."""
    # Utwórz katalog wyjściowy jeśli nie istnieje
    os.makedirs(output_dir, exist_ok=True)

    # Wczytaj obrazy
    img_left = cv2.imread(left_image_path)
    img_right = cv2.imread(right_image_path)

    if img_left is None or img_right is None:
        raise ValueError("Nie można wczytać obrazów")

    # Wczytaj dane kalibracyjne
    mtx_left, dist_left, mtx_right, dist_right, R, T =
load_calibration_data()

    # Oblicz macierz fundamentalną
    F, _ = cv2.findFundamentalMat(np.array([[0, 0], [1, 1]]),
np.array([[0, 0], [1, 1]]))

    # Przygotuj parametry rektyfikacji stereo
    h, w = img_left.shape[:2]
    R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(
        mtx_left, dist_left,
        mtx_right, dist_right,
        (w, h), R, T,
        flags=cv2.CALIB_ZERO_DISPARITY,
        alpha=1
    )

    # Oblicz mapy rektyfikacji
    mapL1, mapL2 = cv2.initUndistortRectifyMap(mtx_left, dist_left, R1,
P1, (w, h), cv2.CV_32FC1)
    mapR1, mapR2 = cv2.initUndistortRectifyMap(mtx_right, dist_right,
R2, P2, (w, h), cv2.CV_32FC1)

    # Rektyfikuj obrazy
```

```
    rect_left = cv2.remap(img_left, mapL1, mapL2, cv2.INTER_LINEAR)
    rect_right = cv2.remap(img_right, mapR1, mapR2, cv2.INTER_LINEAR)

    # Narysuj obwiednie użytecznego obszaru
    rect_left_roi = draw_valid_area(rect_left, roi1)
    rect_right_roi = draw_valid_area(rect_right, roi2)

    # Rysuj linie epipolarne
    for y in range(0, h, 30):  # Rysuj co 30 pikseli
        cv2.line(rect_left_roi, (0, y), (w, y), (0, 0, 255), 1)
        cv2.line(rect_right_roi, (0, y), (w, y), (0, 0, 255), 1)

    # Zapisz wyniki
    base_name_left =
os.path.splitext(os.path.basename(left_image_path))[0]
    base_name_right =
os.path.splitext(os.path.basename(right_image_path))[0]

    cv2.imwrite(os.path.join(output_dir,
f'{base_name_left}_rect_epipolar.png'), rect_left_roi)
    cv2.imwrite(os.path.join(output_dir,
f'{base_name_right}_rect_epipolar.png'), rect_right_roi)
    cv2.imwrite(os.path.join(output_dir, f'{base_name_left}_rect.png'),
rect_left)
    cv2.imwrite(os.path.join(output_dir,
f'{base_name_right}_rect.png'), rect_right)

    return rect_left_roi, rect_right_roi

def main():
    print("Wizualizacja linii epipolarnych i eksport zrektyfikowanych
obrazów")
    print("=" * 50)

    try:
        # Przetwórz parę obrazów
        left_image = 'lab2/s1/left_230.png'
        right_image = 'lab2/s1/right_230.png'

        rect_left, rect_right = process_stereo_pair(left_image,
right_image)

        # Wyświetl wyniki
```

```
        cv2.imshow('Left Image with Epipolar Lines', rect_left)
        cv2.imshow('Right Image with Epipolar Lines', rect_right)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        print("\nWyniki zostały zapisane w katalogu 'output':")
        print("1. Obrazy z liniami epipolarnymi i ROI
(*_rect_epipolar.png)")
        print("2. Zrektyfikowane obrazy bez dodatkowych oznaczeń
(*_rect.png)")

    except Exception as e:
        print(f"Wystąpił błąd: {str(e)}")

if __name__ == "__main__":
    main()
```