



**WYDZIAŁ MATEMATYKI  
i INFORMATYKI**  
Uniwersytet Łódzki

Gabriel Ozeg

Nr albumu: 395263

System antykolizyjny na mikroprocesorze  
Raspberry Pi

Praca magisterska  
na kierunku Informatyka

Praca wykonana pod kierunkiem  
dr Paweł Zajaczkowski  
Katedra Informatyki Stosowanej

Łódź, 2025

**Słowa kluczowe:** Przetwarzanie obrazu, Głębia obrazu, Metody pomiaru odległości w czasie rzeczywistym, Zastosowania w robotyce

**Title in English:** Collision avoidance system on Raspberry Pi microprocessor

**Keywords:** Image Processing, Image Depth, Real-Time Distance Measurement Methods, Applications in Robotics

# Spis treści

<b>1. Wstęp</b>	5
<b>2. Natura kamery</b>	9
2.1. Ogniskowa obiektywu	10
2.2. Dystorsja obrazu	12
2.3. Dane z obrazu kamery	14
<b>3. Funkcjonalność projektu</b>	15
3.1. Kalibracja	16
3.2. Rektyfikacja stereo	19
3.2.1. Geometria epipolarna	20
3.2.2. Macierz fundamentalna i esencjalna	20
3.2.3. Macierz obrotu i wektor translacji	21
3.2.4. Algorytmy rektyfikacji w OpenCV	21
3.3. Mapa dysparycji	23
3.4. Filtr WLS	28
3.5. Główna pętla	29
3.6. Triangulacja	31
<b>4. Wnioski i możliwe ulepszenia</b>	33
<b>Spis rysunków</b>	35
<b>Bibliografia</b>	39



# Rozdział 1

## Wstęp

Przetwarzanie obrazu cyfrowego stanowi jedną z kluczowych dziedzin współczesnej informatyki oraz inżynierii komputerowej, znajdującą zastosowanie w wielu obszarach życia codziennego, przemysłu i nauki. Głównym celem tej dziedziny jest analiza, przekształcanie oraz interpretacja danych wizualnych przy użyciu metod matematycznych, algorytmów komputerowych oraz technik sztucznej inteligencji. Przetwarzanie obrazu umożliwia nie tylko poprawę jakości obrazów, ale także automatyczną ekstrakcję informacji, segmentację obiektów, rozpoznawanie kształtów czy estymację głębi sceny.

Systemy wizyjne znajdują zastosowanie m.in. w diagnostyce medycznej (np. analiza obrazów RTG i MRI), przemyśle (automatyczna kontrola jakości), systemach bezpieczeństwa (rozpoznawanie twarzy i analiza zachowań) oraz w autonomicznych pojazdach i robotyce, gdzie odpowiadają za detekcję przeszkód i wspomaganie decyzji nawigacyjnych w czasie rzeczywistym. Szczególne znaczenie zyskują implementacje systemów wizyjnych na platformach o ograniczonych zasobach sprzętowych, takich jak mikrokontrolery czy komputery jednoplatformowe.

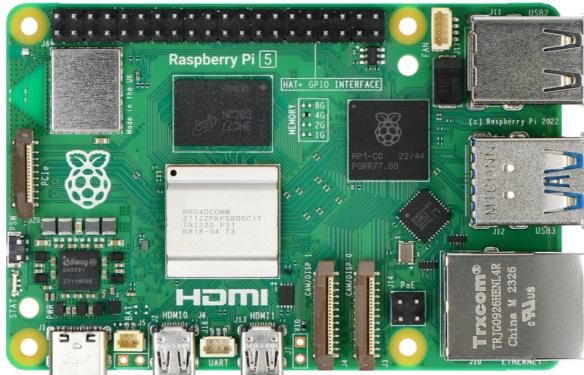
Celem niniejszego projektu jest implementacja oraz ocena działania **systemu antykolizyjnego** opartego na przetwarzaniu obrazu stereoskopowego w czasie rzeczywistym. Główną platformą obliczeniową jest **Raspberry Pi 5** – komputer jednoplatformowy nowej generacji, oferujący około 50% wyższą wydajność w porównaniu do swojego poprzednika, co czyni go obiecującą jednostką dla zastosowań typu *edge computing*.

System bazuje na wykorzystaniu **stereowizji**, czyli przetwarzania obrazu z dwóch zsynchronizowanych kamer w celu uzyskania mapy dysparacji i wyznaczenia odległości do obiektów w scenie. W przypadku wykrycia przeszkody znajdującej się zbyt blisko, system podejmuje decyzję o zatrzymaniu pojazdu poprzez fizyczne odłączenie zasilania jego silnika napędowego.

go, co ma na celu uniknięcie kolizji.

Projekt został zrealizowany przy użyciu następujących komponentów sprzętowych:

- **Raspberry Pi 5** – odpowiada za przetwarzanie obrazu stereoskopowego, analizę głębi oraz sterowanie logiką decyzyjną systemu.



Rysunek 1: Płytnka Raspberry Pi 5.

- **Kamera stereo** – dostarcza zsynchronizowane obrazy z dwóch perspektyw, umożliwiające wyznaczenie mapy głębi.



Rysunek 2: Kamera stereowizyjna

- **Moduł UPS HAT (B) firmy Waveshare** – zapewnia nieprzerwane zasilanie, umożliwiając pracę systemu w warunkach mobilnych oraz zwiększając jego niezawodność.



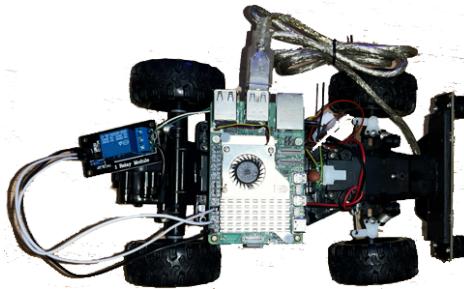
Rysunek 3: Moduł zasilający.

- **Moduł przekaźnikowy** – odpowiada za fizyczne odłączenie zasilania jednostki napędowej w przypadku wykrycia zagrożenia kolizją.



Rysunek 4: Moduł przekaźnika.

- **Zabawkowy samochód elektryczny** – służy jako platforma testowa dla systemu, umożliwiając przeprowadzanie eksperymentów w warunkach laboratoryjnych oraz polowych.



Rysunek 5: Pojazd projektu.

W ramach realizacji projektu przeprowadzona została analiza skuteczności oraz wydajności systemu w warunkach rzeczywistych. Zakres badań obejmował:

- ocenę dokładności estymacji głębi oraz detekcji przeszkód,
- pomiar obciążenia obliczeniowego Raspberry Pi 5 i ocena jego zdolności do pracy w czasie rzeczywistym,
- analizę responsywności systemu w kontekście szybkości reakcji na przeszkody,
- testy stabilności zasilania przy wykorzystaniu UPS HAT w środowisku mobilnym,
- ocenę możliwości zastosowania projektu w celach komercyjnych, m.in. w robotyce mobilnej, pojazdach magazynowych czy inteligentnych systemach bezpieczeństwa.

Implementacja systemu antykolizyjnego na bazie Raspberry Pi 5 oraz kamery stereoskopowej wykazała, że nawet przy ograniczonych zasobach sprzętowych możliwe jest skuteczne

przeprowadzanie przetwarzania obrazu w czasie rzeczywistym. Pomimo istotnego obciążenia obliczeniowego, stereowizja dostarcza bogatych informacji o głębi sceny, co zwiększa niezawodność systemów unikania przeszkód.

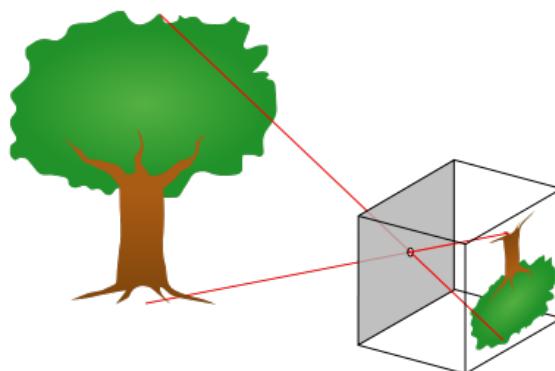
Dzięki kompaktowym rozmiarom, niskiej cenie oraz energooszczędności, Raspberry Pi 5 okazuje się atrakcyjną platformą dla zastosowań edukacyjnych, prototypowych oraz potencjalnie komercyjnych.

## Rozdział 2

### Natura kamery

Kamery rejestrują promienie świetlne z otoczenia. Zasadniczo kamera działa podobnie jak ludzkie oko — odbite promienie światła trafiają do oka i są skupiane na siatkówce.

Najprostszym modelem kamery jest tzw. „kamera otworkowa” [1]. To dobre uproszczenie pozwalające zrozumieć podstawy działania kamery. W tym modelu wszystkie promienie świetlne są blokowane przez ścianki, a tylko te przechodzące przez mały otwór trafiają na powierzchnię światłoczułą wewnętrz kamery, tworząc odwrócony obraz. Poniższa ilustracja przedstawia tę zasadę.



Rysunek 6: Odwrócenie obrazu przez soczewkę

Choć ten model jest prosty, nie nadaje się dobrze do rejestrowania wystarczającej ilości światła przy krótkim czasie naświetlania. Dlatego w kamerach stosuje się soczewki, które skupiają promienie świetlne w jednym punkcie. Niestety, powoduje to powstawanie zniekształceń.

Istnieją dwa główne rodzaje zniekształceń:

- Zniekształcenie promieniowe(radialne) — spowodowane kształtem soczewki, występujące symetrycznie względem środka obrazu.
- Zniekształcenie styczne(tangencjalne) — wynikające z niedoskonałości montażu lub geometrii kamery.

Obrazy zniekształcone w ten sposób można skorygować za pomocą metod matematycznych. Proces kalibracji pozwala stworzyć model geometrii kamery oraz model zniekształceń obiektywu. Te modele określają tzw. parametry wewnętrzne kamery.

## 2.1. Ogniskowa obiektywu

Względny rozmiar obrazu rzutowanego na powierzchnię w kamerze zależy od ogniskowej [2].

W modelu otworkowym ogniskowa to odległość między otworem, przez który przechodzi światło, a obszarem, na który rzutowany jest obraz.

Aby wyznaczyć, jak duży będzie obraz obiektu na płaszczyźnie projekcji, korzystamy z **twierdzenia Talesa**. Dlaczego?

Obiekt znajdujący się w przestrzeni oraz jego rzut w kamerze tworzą dwa **podobne trójkąty**:

- Jeden utworzony przez obiekt o wysokości  $X$ , znajdujący się w odległości  $Z$  od kamery.
- Drugi utworzony przez obraz tego obiektu na płaszczyźnie znajdującej się w odległości  $f$  od otworu kamery, którego wysokość to  $x$ .

Ponieważ kąty tych trójkątów są identyczne (wierzchołek kąta w otworze kamery) i odpowiadające sobie boki są proporcjonalne, możemy zastosować twierdzenie Talesa:

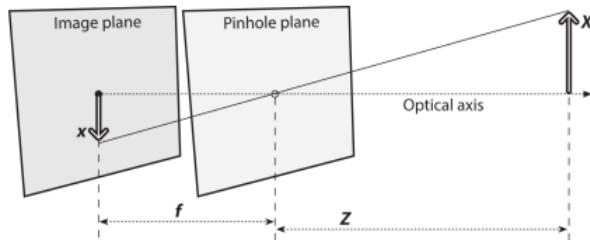
$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = f \cdot \frac{X}{Z}$$

Obraz na matrycy jest odwrócony, dlatego często w literaturze pojawia się wersja ze znakiem minus:

$$-x = f \cdot \frac{X}{Z}$$

- $x$ : obraz obiektu (znak minus wynika z tego, że obraz jest odwrócony)

- $X$ : rozmiar obiektu
- $Z$ : odległość od otworu do obiektu
- $f$ : ogniskowa, odległość od otworu do obrazu



Rysunek 7: Model kamery otworkowej

Ponieważ soczewka nie jest idealnie wyśrodkowana, wprowadzono dwa parametry,  $C_x$  i  $C_y$ , oznaczające odpowiednio poziome i pionowe przemieszczenie soczewki. Ogniskowa na osiach  $X$  i  $Y$  są również różna, ponieważ obszar obrazu jest prostokątny. Daje to następujący wzór na położenie obiektu na powierzchni.

$$x_{\text{screen}} = f_x \left( \frac{X}{Z} \right) + c_x, \quad y_{\text{screen}} = f_y \left( \frac{Y}{Z} \right) + c_y$$

Rzutowane punkty świata rzeczywistego na powierzchnię obrazu można zatem modelować w następujący sposób.  $M$  jest tutaj macierzą wewnętrzną.

Punkt w przestrzeni 3D:

$$Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Po rzutowaniu za pomocą macierzy kamery  $M$ :

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

otrzymujemy punkt obrazu w jednorodnych współrzędnych:

$$q = M \cdot \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x \cdot \frac{X}{Z} + c_x \\ f_y \cdot \frac{Y}{Z} + c_y \\ 1 \end{bmatrix}$$

Po normalizacji współrzędnych jednorodnych:

$$x = \frac{q_x}{q_w} = f_x \cdot \frac{X}{Z} + c_x \quad , \quad y = \frac{q_y}{q_w} = f_y \cdot \frac{Y}{Z} + c_y$$

Dla ogólnej macierzy projekcji:

$$P = M \cdot [R \mid t]$$

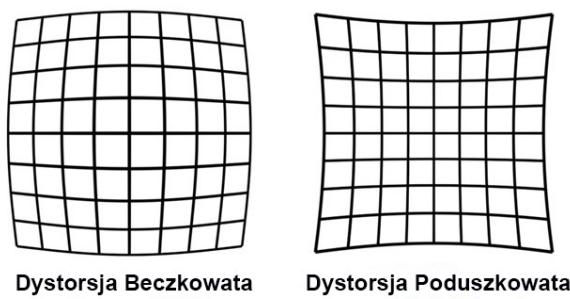
Macierz  $P = M[R|t]$  nazywana jest macierzą projekcji kamery i zawiera zarówno informacje o parametrach wewnętrznych kamery (macierz  $M$ ), jak i jej położeniu i orientacji w przestrzeni (macierze  $R$  i  $t$ ).

Rzut punktu  $Q_h = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$  przy pomocy tej macierzy daje nam współrzędne punktu  $q = \begin{bmatrix} x' \\ y' \\ w \end{bmatrix}$  w jednorodnych współrzędnych, które po znormalizowaniu ( $x = \frac{x'}{w}$ ) dają końcowe położenie punktu na obrazie.

$$x = \frac{x'}{w}, \quad y = \frac{y'}{w}$$

Współrzędne  $x, y$  po normalizacji są współrzędnymi piksela na płaszczyźnie obrazu, czyli miejscem, gdzie dany punkt 3D zostanie odwzorowany na zdjęciu lub klatce z kamery. Uwzględniają one zarówno parametry geometryczne obiektywu (ogniskowe  $f_x, f_y$ ) jak i przesunięcia optycznego środka obrazu ( $c_x, c_y$ ).

## 2.2. Dystorsja obrazu



Rysunek 8: Rodzaje dystorsji

Teoretycznie możliwe jest zbudowanie obiektywu, który nie powoduje zniekszałceń, np. przy użyciu soczewki parabolicznej. W praktyce jednak znacznie łatwiej i taniej jest wytwarzanie soczewki sferyczne, które niestety powodują różne typy zniekszałceń geometrycznych obrazu [9].

Aby móc opisać i skorygować te zniekszałcenia, punkt obrazu wyrażony w pikselach  $(u, v)$  przekształca się najpierw do znormalizowanego układu współrzędnych kamery:

$$x = \frac{u - c_x}{f_x}, \quad y = \frac{v - c_y}{f_y}$$

Gdzie:

- $(c_x, c_y)$  — współrzędne głównego punktu optycznego (środka obrazu),
- $(f_x, f_y)$  — ogniskowe w poziomie i pionie wyrażone w pikselach,
- $(x, y)$  — znormalizowane współrzędne względem osi optycznej kamery.

## Zniekszałcenia promieniowe

Zniekszałcenia promieniowe (ang. *radial distortion*) [3] są symetryczne względem środka obrazu i ich wpływ rośnie wraz z odległością od środka. Dla punktów znormalizowanych odległość ta dana jest przez:

$$r^2 = x^2 + y^2$$

Efekt zniekszałcenia można modelować jako nieliniową funkcję  $D(r)$ , która modyfikuje współrzędne punktu zależnie od  $r$ . Ponieważ funkcja  $D(r)$  nie jest znana analitycznie, stosujemy jej rozwinięcie Taylora w punkcie  $r = 0$ :

$$D(r) = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots$$

Ograniczamy się zwykle do trzeciego rzędu ( $r^6$ ), ponieważ kolejne składniki mają marginalny wpływ, a zwiększały złożoność obliczeń. Po uwzględnieniu tej funkcji korekta zniekszałceń promieniowego ma postać:

$$\begin{aligned} x_{\text{radial}} &= x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{\text{radial}} &= y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned}$$

## Zniekształcenia styczne (tangencjalne)

Zniekształcenia styczne pojawiają się w wyniku niewspółosiowości soczewek (np. przesunięcia lub nachylenia względem osi optycznej). Ich model opiera się na dwóch parametrach  $p_1$  i  $p_2$ :

$$\begin{aligned}x_{\text{tangential}} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\y_{\text{tangential}} &= y + [p_1(r^2 + 2y^2) + 2p_2xy]\end{aligned}$$

## Pełna korekta punktu znormalizowanego

Sumując oba typy zniekształceń, uzyskujemy skorygowane współrzędne punktu w układzie znormalizowanym:

$$\begin{aligned}x_{\text{corrected}} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1xy + p_2(r^2 + 2x^2) \\y_{\text{corrected}} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y^2) + 2p_2xy\end{aligned}$$

## Powrót do współrzędnych obrazu

Na końcu przekształcamy punkt z powrotem do układu pikselowego:

$$u_{\text{corrected}} = f_x \cdot x_{\text{corrected}} + c_x, \quad v_{\text{corrected}} = f_y \cdot y_{\text{corrected}} + c_y$$

W ten sposób otrzymujemy ostateczną, skorygowaną pozycję punktu obrazu, która uwzględnia wpływ obu typów zniekształceń optycznych.

## 2.3. Dane z obrazu kamery

Znormalizowane współrzędne  $(x, y)$  oraz współrzędne pikselowe  $(u, v)$  służą do różnych celów, ale są ze sobąściem powiązane. Współrzędne znormalizowane są używane wszędzie tam, gdzie istotna jest struktura geometryczna sceny i relacje przestrzenne – np. w algorytmach rekonstrukcji 3D, lokalizacji kamery czy kalibracji. Z kolei współrzędne pikselowe są wykorzystywane do interakcji z obrazem: lokalizacji punktów na zdjęciu, wizualizacji, wykrywania cech czy ekstrakcji informacji wizualnych.

## Rozdział 3

### Funkcjonalność projektu

Stereowizja, nazywana również wizją stereoskopową, jest techniką pozyskiwania informacji przestrzennych na podstawie dwóch lub więcej obrazów tej samej sceny, zarejestrowanych z różnych punktów obserwacyjnych. Metoda ta, inspirowana sposobem postrzegania przestrzeni przez ludzkie oczy, stanowi jedno z najstarszych i najbardziej rozwiniętych podejść do estymacji głębi w przetwarzaniu obrazu.

W przeciwieństwie do wizji monokularnej, stereowizja pozwala na geometrycznie uzasadnione, bezpośrednie obliczanie odległości do obiektów. Dzięki temu znajduje zastosowanie w systemach wymagających wysokiej precyzji pomiarów, takich jak robotyka, pojazdy autonomiczne czy systemy pomiarowe 3D.

W systemie stereowizyjnym wykorzystuje się dwa obrazy uzyskane przez kamery rozmieszczone w znanej odległości od siebie, określonej jako *baza stereo*. Kluczowym pojęciem jest *paralaksa* – przesunięcie położenia obrazu tego samego punktu sceny między lewym a prawym obrazem.

Przy założeniu idealnej konfiguracji (kamery wyrównane, płaszczyzny obrazów równolegle), głębokość  $Z$  punktu sceny można obliczyć z zależności:

$$Z = \frac{f \cdot B}{d}$$

gdzie:

- $f$  – ogniskowa kamery,
- $B$  – odległość między kamerami (baza stereo),
- $d$  – przesunięcie (paralaksa) danego punktu między obrazem lewym i prawym.

Większa wartość paralaksy oznacza mniejszą odległość obiektu od kamery.

Typowy proces obliczania głębi metodą stereowizyjną obejmuje następujące kroki:

1. **Korekcja zniekształceń optycznych** – eliminacja zniekształceń promieniowych i stycznych w obrazach na podstawie parametrów kalibracyjnych kamery.
2. **Rektyfikacja obrazów** – transformacja obrazów w taki sposób, aby linie epipolarne były równoległe i współłaszczyznowe względem osi  $Y$ , co umożliwia wyszukiwanie korespondencji wyłącznie wzdłuż osi  $X$ .
3. **Wyszukiwanie korespondencji** – identyfikacja odpowiadających sobie punktów na obrazach lewym i prawym, prowadząca do wygenerowania *mapy dysparcji* przedstawiającej różnice położenia na osi  $X$ .
4. **Triangulacja** – przekształcenie mapy dysparcji na mapę głębi przy wykorzystaniu parametrów geometrycznych układu kamer.

Opracowany system został zaimplementowany w języku Python z wykorzystaniem biblioteki OpenCV do przetwarzania obrazów oraz biblioteki RPI.GPIO do obsługi pinów wejścia/wyjścia Raspberry Pi.

W procesie działania systemu wykonywana jest:

- kalibracja kamer na podstawie zestawu zdjęć wzorcowych,
- generacja mapy dysparcji,
- estymacja odległości dla każdego piksela na podstawie równania wyznaczonego eksperymentalnie,
- filtracja wyników przy użyciu filtra WLS (Weighted Least Squares), co pozwala na wyraźniejsze rozpoznawanie krawędzi obiektów.

System uruchamiany jest automatycznie wraz ze startem systemu operacyjnego *Raspberry Pi*. Moduł przekaźnikowy domyślnie pozostaje w stanie odłączenia, a po aktywacji systemu obie kamery są inicjalizowane w trybie pracy ciągłej.

### 3.1. Kalibracja

```
# Określa warunki, przy których iteracyjny algorytm się zatrzymuje
# Przerwij iteracje, gdy wykonano 30 kroków lub zmiana wyniku jest mniejsza niż 0.001
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
criteria_stereo= (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
```

```

# Przygotowanie punktów obiektu
objp = np.zeros((9*6,3), np.float32)
objp[:, :, 2] = np.mgrid[0:9, 0:6].T.reshape(-1, 2)

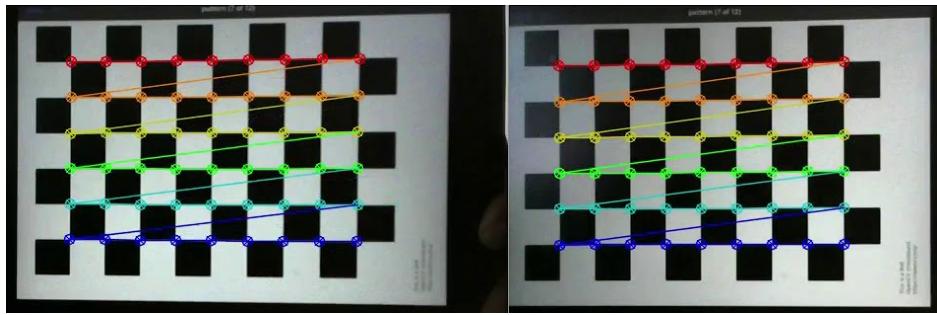
# Tablice do przechowywania punktów obiektu i punktów obrazu ze wszystkich obrazów
objpoints = []      # Punkty 3D w przestrzeni świata rzeczywistego
imgpointsR = []      # Punkty 2D na płaszczyźnie obrazu
imgpointsL = []

for i in range(0, 64):
    t = str(i)
    ChessImgR = cv2.imread('calib_images/right_chessboard-' + t + '.png', 0)
    ChessImgL = cv2.imread('calib_images/left_chessboard-' + t + '.png', 0)
    if ChessImgR is None or ChessImgL is None:
        continue # Skip this iteration if loading failed
    retR, cornersR = cv2.findChessboardCorners(ChessImgR, (9, 6), None)
    retL, cornersL = cv2.findChessboardCorners(ChessImgL, (9, 6), None)
    if retR and retL:
        objpoints.append(objp)
        cv2.cornerSubPix(ChessImgR, cornersR, (11, 11), (-1, -1), criteria)
        cv2.cornerSubPix(ChessImgL, cornersL, (11, 11), (-1, -1), criteria)
        imgpointsR.append(cornersR)
        imgpointsL.append(cornersL)

# Kalibracja
retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(objpoints, imgpointsR,
                                                       ChessImgR.shape[::-1], None, None)
retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(objpoints, imgpointsL,
                                                       ChessImgL.shape[::-1], None, None)
OmtxR, roiR = cv2.getOptimalNewCameraMatrix(mtxR, distR,
                                             ChessImgR.shape[::-1], 1, ChessImgR.shape[::-1])
OmtxL, roiL = cv2.getOptimalNewCameraMatrix(mtxL, distL,
                                             ChessImgL.shape[::-1], 1, ChessImgL.shape[::-1])

```

Proces kalibracji kamery polega na wyznaczeniu jej parametrów wewnętrznych i zewnętrznych, co jest możliwe dzięki analizie serii zdjęć wzorca kalibracyjnego - najczęściej szachownicy — wykonanych pod różnymi kątami. Kluczowe jest, aby narożniki szachownicy były dobrze widoczne i możliwe do jednoznacznego zidentyfikowania przy użyciu funkcji `cv2.findChessboardCorners()`. Zgodnie z zaleceniami OpenCV, do uzyskania wiarygodnej kalibracji wymaganych jest co najmniej 10 obrazów dla każdej kamery. W niniejszym projekcie wykorzystano 32 obrazy kalibracyjne przypisane osobno do lewej i prawej kamery.



Rysunek 9: Widok wykrytych wierzchołków szachownic.

Po wykryciu narożników ich pozycje są precyzowane za pomocą funkcji `cv2.cornerSubPix()`, co pozwala uzyskać większą dokładność w procesie kalibracji. Dane te są następnie zapisywane w postaci trzech wektorów:

- `imgpointsR`: zawiera współrzędne narożników na prawym obrazie (w przestrzeni obrazu)
- `imgpointsL`: zawiera współrzędne narożników na lewym obrazie (w przestrzeni obrazu)
- `objpoints`: zawiera współrzędne narożników w przestrzeni obiektu.

Dane te wykorzystywane są przez funkcję `cv2.calibrateCamera()` do wyznaczenia macierzy kamery, współczynników dystorsji oraz wektorów rotacji i translacji dla każdej kamery. Macierz wewnętrzna kamery  $M$  opisuje rzutowanie punktów ze świata 3D na obraz 2D i przyjmuje postać:

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- $f_x = f \cdot s_x$  — ogniskowa wyrażona w pikselach w poziomie (osi X)
- $f_y = f \cdot s_y$  — ogniskowa wyrażona w pikselach w pionie (osi Y)
- $c_x, c_y$  — współrzędne punktu głównego (*principal point*), zazwyczaj w centrum obrazu
- Zera poza przekątną oznaczają brak nachylenia między osiami (czyli brak efektu trapezowego)
- Wartość 1 w prawym dolnym rogu służy do zapewnienia jednorodności w transformacjach macierzowych

Przykładowe macierze kamer lewej i prawej uzyskane po kalibracji przedstawiają się następująco:

$$\mathbf{M}_L = \begin{bmatrix} 777,7 & 0 & 345,1 \\ 0 & 780,6 & 171,3 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{M}_R = \begin{bmatrix} 792,2 & 0 & 274,9 \\ 0 & 801,4 & 212,9 \\ 0 & 0 & 1 \end{bmatrix}$$

W celu dalszego przetwarzania obrazów i uzyskania lepszej dokładności, macierze te mogą zostać zoptymalizowane przy użyciu funkcji cv2.getOptimalNewCameraMatrix(). Zoptymalizowane wersje macierzy, uwzględniające rzeczywisty obszar widzenia i dystorsje, są wykorzystywane podczas rektyfikacji obrazów za pomocą funkcji cv2.stereoRectify().

$$\mathbf{M}_L = \begin{bmatrix} 636,0 & 0 & 399,6 \\ 0 & 760,1 & 170,2 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{M}_R = \begin{bmatrix} 772,2 & 0 & 284,1 \\ 0 & 755,5 & 217,8 \\ 0 & 0 & 1 \end{bmatrix}$$

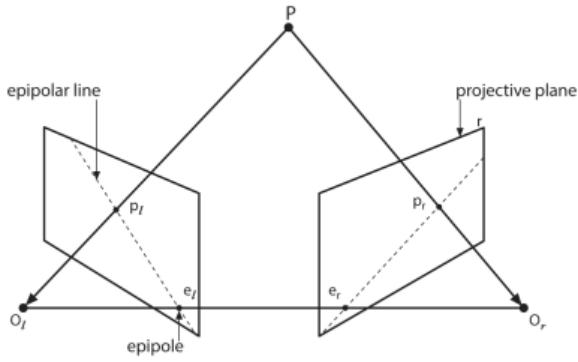
Aby wyznaczyć wzajemne położenie kamer względem siebie (rotację i translację), wykorzystywana jest funkcja cv2.stereoCalibrate(), która pozwala określić pełną konfigurację geometryczną układu stereo. Parametry te są niezbędne do poprawnego przekształcenia obrazów oraz dalszej analizy głębi, np. przy obliczaniu mapy dysparacji.

## 3.2. Rektyfikacja stereo

Jednym z kluczowych zagadnień w stereowizji jest **geometria epipolarna**, która opisuje zależność pomiędzy rzutami punktów przestrzennych na dwa obrazy uzyskane z różnych punktów widzenia. Celem tej geometrii jest ograniczenie przestrzeni poszukiwania punktów odpowiadających w drugim obrazie, co znacząco upraszcza dopasowywanie i rekonstrukcję głębi.

Aby dodatkowo uprościć ten proces, stosuje się **rektyfikację stereo**, czyli transformację obrazów, która sprowadza linie epipolarne do postaci równoległych i poziomych. Dzięki temu dopasowywanie punktów może być wykonywane wzdłuż jednej osi (najczęściej poziomej), co przyspiesza obliczenia i zwiększa ich dokładność.

### 3.2.1. Geometria epipolarna



Rysunek 10: Model kamery stereo.

Na rysunku 16 przedstawiono uproszczony model kamery stereo zbudowanej z dwóch kamer otworkowych. Punkty  $O_l$  i  $O_r$  to środki rzutów lewej i prawej kamery. Proste łączące punkty  $p_l$  z  $e_l$  oraz  $p_r$  z  $e_r$  to tzw. **linie epipolarne**, natomiast punkty  $e_l$  i  $e_r$  to **epipole** – rzuty środka jednej kamery na płaszczyznę obrazu drugiej.

Geometria epipolarna umożliwia ograniczenie przestrzeni przeszukiwania punktu odpowiadającego z pełnej płaszczyzny obrazu do jednej linii – linii epipolarnej. Ułatwia to znacznie proces dopasowywania punktów. Można to podsumować w następujących punktach:

- Każdy punkt przestrzenny należy do płaszczyzny epipolarnej.
- Odpowiadający mu punkt w drugim obrazie musi leżeć na odpowiedniej linii epipolarnej (warunek epipolarny).
- Proces wyszukiwania punktu korespondującego można zredukować do jednego wymiaru, jeżeli znana jest geometria epipolarna.
- Kolejność punktów na liniach epipolarnych jest zachowana między obrazami.

### 3.2.2. Macierz fundamentalna i esencjalna

Aby matematycznie opisać zależności pomiędzy punktami w dwóch obrazach, wykorzystuje się dwie macierze: **macierz esencjalną**  $E$  oraz **macierz fundamentalną**  $F$ . Macierz  $E$  opisuje wzajemną orientację kamer (rotację i translację), natomiast  $F$  uwzględnia dodatkowo parametry wewnętrzne kamer, takie jak ogniskowa czy środek obrazu.

Związek pomiędzy punktami  $p_l$  i  $p_r$  w obrazach opisuje równanie epipolarne:

$$p_r^T E p_l = 0$$

Ponieważ macierz  $E$  ma rangę 2, opisuje ona jedynie płaszczyznę, nie zaś pełną transformację punktów. Dlatego wprowadza się macierz  $F$ , którą oblicza się jako:

$$F = (M_r^{-1})^T E M_l^{-1}$$

gdzie  $M_l$  i  $M_r$  to macierze wewnętrzne lewej i prawej kamery, a  $q = Mp$  to przekształcenie punktu przestrzennego do przestrzeni obrazu. Zatem pełna forma równania epipolarnego to:

$$q_r^T F q_l = 0$$

### 3.2.3. Macierz obrotu i wektor translacji

Aby wyznaczyć relację przestrzenną między kamerami, definiuje się:

- $P_l = R_l P + T_l$  – przekształcenie punktu przestrzennego  $P$  do układu lewej kamery,
- $P_r = R_r P + T_r$  – analogiczne przekształcenie do układu prawej kamery.

Na tej podstawie można wyznaczyć:

$$P_l = R(P_r - T)$$

co prowadzi do zależności:

$$R = R_r R_l^T, \quad T = T_r - R T_l$$

### 3.2.4. Algorytmy rektyfikacji w OpenCV

Celem rektyfikacji jest takie przekształcenie obrazów, aby ich linie epipolarne były współliniowe i poziome. W praktyce oznacza to sprowadzenie układów optycznych obu kamer do tej samej płaszczyzny rzutowania.

W wyniku rektyfikacji uzyskuje się dla każdej kamery:

- wektor dystorsji,
- macierz rotacji rektyfikującej  $R_{\text{rect}}$ ,
- zrektyfikowaną macierz projekcji  $M_{\text{rect}}$ ,
- oryginalną (nierektyfikowaną) macierz kamery  $M$ .

## Algorytm Hartley'a

Algorytm Hartley'a [14] pozwala przeprowadzić rektyfikację obrazów bez znajomości parametrów wewnętrznych kamer (metoda niekalibrowana).

### Etapy działania:

- Wyszukiwanie punktów korespondencyjnych (np. z użyciem cech SIFT, SURF, ORB),
- Estymacja macierzy fundamentalnej  $F$ ,
- Obliczenie homografii, które przekształcają epipole do nieskończoności (linie epipolarne stają się poziome).

Wadą tej metody jest brak informacji o rzeczywistej skali sceny – relacje przestrzenne są wyłącznie względne.

Przykładowa implementacja w Pythonie z wykorzystaniem funkcji `stereoRectifyUncalibrated` z biblioteki OpenCV:

```
# Wykrycie i opis cech (np. ORB)
orb = cv2.ORB_create()
kp1, des1 = orb.detectAndCompute(imgL, None)
kp2, des2 = orb.detectAndCompute(imgR, None)

# Dopasowanie cech metodą BFMatcher
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1, des2)
matches = sorted(matches, key=lambda x: x.distance)

# Konwersja dopasowań do tablic punktów
pts1 = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1,1,2)
pts2 = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1,1,2)

# Estymacja macierzy fundamentalnej
F, mask = cv2.findFundamentalMat(pts1, pts2, cv2.FM_RANSAC)

# Wybór tylko inlierów
pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]

# Obliczenie homografii rektyfikujących
ret, H1, H2 = cv2.stereoRectifyUncalibrated(
    pts1, pts2, F, imgSize=imgL.shape[::-1]
)

# Zastosowanie przekształceń homograficznych
imgL_rect = cv2.warpPerspective(imgL, H1, imgL.shape[::-1])
imgR_rect = cv2.warpPerspective(imgR, H2, imgR.shape[::-1])
```

## Algorytm Bouguet'a

Algorytm Bouguet'a [15], stosowany m.in. w narzędziu *Camera Calibration Toolbox* dla MATLAB-a, opiera się na wcześniejszej kalibracji kamer (metoda kalibrowana).

### Etapy działania:

- Kalibracja kamer z wykorzystaniem wzorca (np. szachownicy),
- Wyznaczenie parametrów wewnętrznych i zewnętrznych ( $R$  i  $T$ ),
- Przekształcenie obrazów tak, aby ich osie optyczne były równoległe,
- Wygenerowanie zrektyfikowanych obrazów, w których odpowiadające piksele leżą na tych samych liniach poziomych.

Metoda ta zapewnia większą precyzyję, lecz jest wrażliwa na błędy kalibracji (np. niedokładne wykrycie wzorca).

```
# Funkcja stereoRectify
RL, RR, PL, PR, Q, roiL, roiR = cv2.stereoRectify(MLS, dLS, MRS, dRS,
                                                     ChessImgR.shape[::-1], R, T, 0, (0, 0))

# Generowanie map przekształceń
Left_Stereo_Map = cv2.initUndistortRectifyMap(MLS, dLS, RL, PL,
                                                ChessImgR.shape[::-1], cv2.CV_16SC2)
Right_Stereo_Map = cv2.initUndistortRectifyMap(MRS, dRS, RR, PR,
                                                ChessImgR.shape[::-1], cv2.CV_16SC2)
```

Funkcja `cv2.stereoRectify()` przekształca obrazy w taki sposób, aby ich linie epipolarne były poziome. Z kolei `cv2.initUndistortRectifyMap()` umożliwia wygenerowanie obrazów bez dystorsji i z poprawną geometrią epipolarną.

### Przykładowe macierze kalibracyjne:

$$\mathbf{M}_L = \begin{bmatrix} 791.0 & 0 & 390.6 \\ 0 & 791.0 & 194.4 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{M}_R = \begin{bmatrix} 791.0 & 0 & 390.6 \\ 0 & 791.0 & 194.4 \\ 0 & 0 & 1 \end{bmatrix}$$

Tak przygotowane obrazy mogą być bezpośrednio wykorzystane do obliczania mapy dysparcji oraz rekonstrukcji 3D.

### 3.3. Mapa dysparycji

Mapa dysparycji[6] to obraz, w którym każdemu pikselowi przypisywana jest wartość reprezentująca przesunięcie (różnicę pozycji) pomiędzy odpowiedającymi sobie punktami w

obrazie lewym i prawym. Im większe przesunięcie (czyli dysparycja), tym bliżej znajduje się dany obiekt względem kamery. Taka mapa stanowi podstawę do obliczenia mapy głębokości, która pozwala oszacować odległość poszczególnych punktów sceny od obserwatora.

## Algorytm StereoSGBM

```
# Inicjalizacja StereoSGBM
block_size = 7
min_disp = 2
num_disp = 130-min_disp
stereo = cv2.StereoSGBM_create(
    minDisparity = min_disp,
    # Minimalna dysparycja (najmniejsze oczekiwane przesunięcie pikseli).
    # Zwykle 0 lub lekko dodatnia wartość, jeśli obiekty mogą być bardzo daleko.
    numDisparities = num_disp,
    # Liczba możliwych poziomów dysparycji (musi być podzielna przez 16!).
    # Określa zakres przeszukiwania. Większy = można wykryć bliższe obiekty.
    blockSize = block_size,
    # Rozmiar bloku dopasowania (okno porównywane między obrazami).
    # Większe wartości = lepsza odporność na szumy, ale gorsza precyzja brzegów.
    uniquenessRatio = 10,
    # Minimalna różnica procentowa między najlepszym a drugim najlepszym dopasowaniem.
    # Pomaga odrzucać niepewne wyniki - im wyższa wartość, tym ostrzejsze kryterium.
    speckleWindowSize = 100,
    # Maksymalny rozmiar obszaru z "plamkami" (ang. speckles), który zostanie usunięty.
    # Używane do usuwania małych, niespójnych obszarów w mapie dysparycji.
    speckleRange = 32,
    # Maksymalna dozwolona różnica dysparycji w obrębie speckle.
    # Pomaga wyciąć obszary o nietypowych skokach dysparycji.
    disp12MaxDiff = 5,
    # Maksymalna dopuszczalna różnica między wynikami z dopasowania lewo-prawo i prawo-lewo.
    # Służy do sprawdzania spójności dopasowania - niskie wartości odrzucają więcej niepewnych pikseli.
    P1 = 8*3*block_size**2,
    # Kara za niewielkie zmiany dysparycji między sąsiednimi pikselami (gładkość).
    # Wzór zależy od liczby kanałów (3 dla koloru) i wielkości bloku.
    P2 = 32*3*block_size**2
    # Kara za większe zmiany dysparycji (duże skoki).
    # Powinna być większa niż P1 - większe wartości = bardziej gładka mapa dysparycji.
)
# Tworzy drugą instancję StereoSGBM, identyczna jak stereo,
# ale z ustawieniami odpowiednimi do przetwarzania prawego obrazu względem lewego.
stereoR=cv2.ximgproc.createRightMatcher(stereo)
```

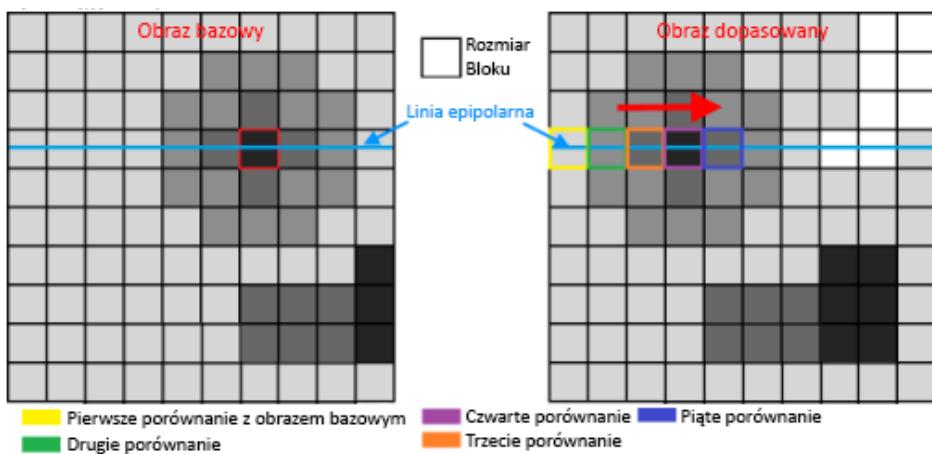
Aby obliczyć mapę dysparycji, wykorzystywany jest obiekt klasy StereoSGBM, tworzony za pomocą funkcji `cv2.StereoSGBM-create()`. Algorytm ten wykorzystuje metodę półglobalnego dopasowania blokowego (ang. *Semi-Global Block Matching*) [13], która umożliwia estymację przesunięć pomiędzy obrazami stereo — z lewej i prawej kamery.

Jednym z kluczowych parametrów wejściowych algorytmu jest rozmiar bloku, który określa

wielkość lokalnego sąsiedztwa wykorzystywanego do dopasowania. W przypadku, gdy wartość ta jest większa niż 1, algorytm operuje nie na pojedynczych pikselach, lecz na blokach. W praktyce oznacza to, że każdy blok z obrazu referencyjnego (np. lewego) porównywany jest z blokami z obrazu dopasowywanego (np. prawego) w celu znalezienia najlepszego dopasowania.

Jeśli stereo-kalibracja i rektyfikacja zostały przeprowadzone poprawnie, porównania odbywają się jedynie wzdłuż odpowiadających sobie wierszy, czyli linii epipolarnych. Dzięki temu przeszukiwanie ogranicza się do jednej wymiarowej przestrzeni (poziomej), co znacznie zmniejsza złożoność obliczeniową algorytmu.

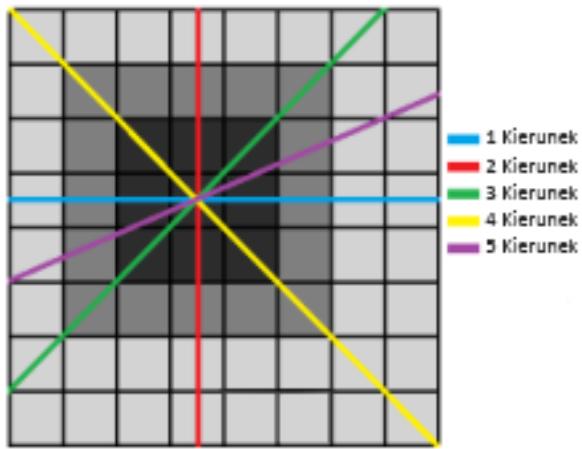
Na przykład, blok o współrzędnych (4,7) w obrazie bazowym zostaje porównany z wszystkimi blokami (4, i) w tej samej linii epipolarnej obrazu dopasowywanego.



Rysunek 11: Wyszukiwanie pasujących bloków za pomocą StereoSGBM.

Im większy stopień dopasowania między blokami, tym większe prawdopodobieństwo, że reprezentują one ten sam punkt w przestrzeni trójwymiarowej. W przedstawionym przykładzie najwyższe dopasowanie dla bloku referencyjnego (4,7) uzyskano względem bloku (4,4) w obrazie dopasowywanym.

Choć w teorii dopasowania dokonuje się tylko w jednym kierunku (poziomym), implementacja algorytmu w OpenCV zakłada analizę także w dodatkowych kierunkach (łącznie pięciu), co pozwala na zwiększenie dokładności dopasowań poprzez uwzględnienie lokalnych kontekstów z różnych stron.



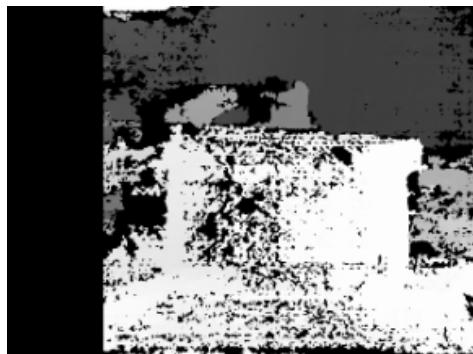
Rysunek 12: Pięć kierunków przeszukiwania w algorytmie StereoSGBM.

Wartość dysparcji uzyskuje się poprzez obliczenie różnicy poziomej współrzędnej piksela (lub bloku) w obrazie referencyjnym i odpowiadającej mu pozycji w obrazie dopasowywanym. W praktyce przyjmuje się wartość bezwzględną tej różnicy, a im jest ona większa, tym obiekt znajduje się bliżej kamery (zgodnie z zasadą triangulacji w stereowizji).

Algorytm zazwyczaj operuje na obrazach w odcieniach szarości (jednokanałowych), co pozwala znacząco zredukować czas obliczeń. Możliwe jest również zastosowanie obrazów kolorowych (np. w przestrzeni BGR), jednak wiąże się to ze zwiększoną obciążeniem procesora, bez gwarancji proporcjonalnej poprawy wyników.

Właściwa mapa dysparcji obliczana jest przy użyciu metody `compute()` na wcześniej skonfigurowanym obiekcie StereoSGBM.

Dzięki parametrom ustalonym podczas inicjalizacji otrzymujemy następujący wynik dla mapy dysparcji.



Rysunek 13: Wynik dla mapy dysproporcji.

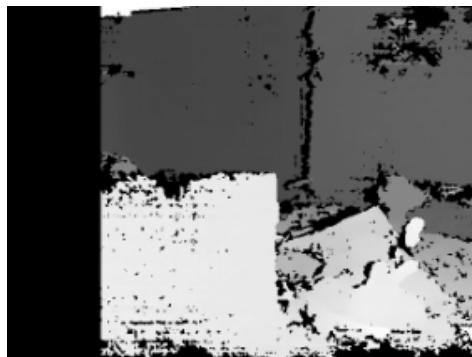
Na wygenerowanej mapie dysparcji mogą nadal występować zakłócenia w postaci lokalnych

szumów. W celu ich redukcji stosuje się filtrację morfologiczną, która pozwala poprawić spójność i ciągłość danych głębokości. W szczególności wykorzystywany jest operator zamykania (morphological closing), realizowany w OpenCV za pomocą funkcji `cv2.morphologyEx()` z flagą `cv2.MORPH_CLOSE`. Zastosowanie tego filtra pozwala na eliminację drobnych czarnych artefaktów, które mogą pojawić się wewnętrz jednorodnych obszarów na mapie.



Rysunek 14: Przykład filtra zamkajającego.

Poniżej inny przykład tej samej sceny, aby lepiej zobaczyć różnicę.



Rysunek 15: Wynik mapy dysproporcji po filtrze zamkajającym.

## Algorytm StereoBM

Alternatywą dla metody StereoSGBM jest StereoBM (ang. *Block Matching*) [11], będący jedną z najstarszych implementacji dopasowania stereoskopowego w bibliotece OpenCV. Algorytm ten bazuje na bezpośrednim porównywaniu bloków pikseli pomiędzy obrazem lewym i prawym wzdłuż linii epipolarnych, wykorzystując prostą funkcję kosztu, np. *Sum of Absolute Differences* (SAD).

Inicjalizacja przebiega następująco:

```
Inicjalizacja StereoBM  
num_disp = 64 # musi być podzielne przez 16
```

```

block_size = 15 # większy blok = większa odporność na szумy

stereo_bm = cv2.StereoBM_create(
    numDisparities=num_disp,
    blockSize=block_size
)

Obliczenie mapy dysparycji
disparity_bm = stereo_bm.compute(grayL, grayR)

```

W odróżnieniu od StereoSGBM, algorytm StereoBM analizuje jedynie lokalne dopasowania bez optymalizacji globalnej. Dzięki temu jest znaczco szybszy i mniej zasobozerny, co czyni go dobrym wyborem w systemach wbudowanych (np. Raspberry Pi) lub w aplikacjach wymagających przetwarzania w czasie rzeczywistym. Jednak jego prostota powoduje większą wrażliwość na szumy i mniejszą dokładność w obszarach jednorodnych lub przy słabym tekście.

Podobnie jak w przypadku StereoSGBM, poprawna rektyfikacja obrazów jest warunkiem użyskania wiarygodnej mapy dysparycji. Ze względu jednak na mniejszą złożoność obliczeń, StereoBM wymaga często dodatkowych filtrów (np. medianowego lub morfologicznego) w celu poprawy jakości wyników.

### 3.4. Filtr WLS

```

# Parametry filtra WLS
wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=stereo)
wls_filter.setLambda(80000)
wls_filter.setSigmaColor(1.8)

```

W celu dalszego ograniczenia szumu w mapie dysparycji stosowany jest filtr ważonych najmniejszych kwadratów (ang. *Weighted Least Squares*) [12], dostępny w module `cv2.ximgproc`. Jest on szczególnie skuteczny w poprawianiu ciągłości strukturalnej oraz w zachowaniu ostrych krawędzi w scenach o złożonej geometrii.

Parametr lambda kontroluje stopień wygładzania mapy: im wyższa jego wartość, tym bardziej struktura wynikowej mapy przypomina obraz referencyjny (np. lewy obraz stereo). W praktyce często stosuje się wartości rzędu 8000, jednak w omawianym przypadku zastosowano wartość `lambda = 80000`, co pozwoliło uzyskać bardziej stabilne rezultaty. Z kolei parametr sigma określa, jak precyjnie filtr ma traktować obszary znajdujące się w pobliżu krawędzi – wyższe wartości sprzyjają lepszemu zachowaniu konturów obiektów.

Aby skorzystać z filtra WLS, tworzony jest dodatkowy obiekt dopasowania stereo dla prawej

go obrazu, przy użyciu funkcji `cv2.ximgproc.createRightMatcher()`, bazujący na głównym obiekcie StereoSGBM. Następnie inicjalizowany jest filtr WLS poprzez `cv2.ximgproc.createDisparityWLSFilter()` i konfigurowany z użyciem uprzednio utworzonych dopasowań.

Sam proces filtracji przeprowadzany jest metodą `filter()` filtra WLS, a jego wynik poddawany jest normalizacji za pomocą funkcji `cv2.normalize()`, co pozwala na wizualizację rezultatu.

Należy jednak zauważyć, że wynik filtra WLS nie jest już bezpośrednio wykorzystywalną mapą dysparcji — jest to obraz zakodowany w formacie `uint8`, który dobrze uwidacznia krawędzie, lecz nie zawiera już rzeczywistych wartości głębokości (w przeciwieństwie do oryginalnej mapy dysparcji w formacie `float32`).

### 3.5. Główna pętla

```
while True:

    ret, frame = Cam.read()
    if not ret:
        # Przerwij jesli nie odczytano klatki z kamery
        break

    height, width, _ = frame.shape
    mid = width // 2

    left_frame = executor.submit(lambda: frame[:, :mid])
    right_frame = executor.submit(lambda: frame[:, mid:])
    LFrame = left_frame.result()
    RFrame = right_frame.result()

    # Równolegle operacje remapowania
    future_Left_nice = executor.submit(cv2.remap, LFrame,
                                         Left_Stereo_Map[0], Left_Stereo_Map[1],
                                         interpolation=cv2.INTER_LANCZOS4,
                                         borderMode=cv2.BORDER_CONSTANT)
    future_Right_nice = executor.submit(cv2.remap, RFrame,
                                         Right_Stereo_Map[0], Right_Stereo_Map[1],
                                         interpolation=cv2.INTER_LANCZOS4,
                                         borderMode=cv2.BORDER_CONSTANT)

    Left_nice = future_Left_nice.result()
    Right_nice = future_Right_nice.result()

    # Równoległa konwersja na skale szarości
    future_gray_left = executor.submit(cv2.cvtColor, Left_nice, cv2.COLOR_BGR2GRAY)
    future_gray_right = executor.submit(cv2.cvtColor, Right_nice, cv2.COLOR_BGR2GRAY)
    Gray_left = future_gray_left.result()
    Gray_right = future_gray_right.result()
```

```

# Równoległa kalkulacja dysproporcji
future_dispL = executor.submit(stereoL.compute, Gray_left, Gray_right)
future_dispR = executor.submit(stereoR.compute, Gray_right, Gray_left)
DispL = np.int16(future_dispL.result())
DispR = np.int16(future_dispR.result())

disp = ((DispL.astype(np.float32) / 16) - min_disp) / num_disp

filteredImg = wls_filter.filter(DispL, Gray_left, None, DispR)
filteredImg = cv2.normalize(src=filteredImg, dst=filteredImg, beta=0,
                           alpha=255, norm_type=cv2.NORM_MINMAX)
filteredImg = np.uint8(filteredImg)

filt_Color = cv2.applyColorMap(filteredImg, cv2.COLORMAP_OCEAN)

```

Po wygenerowaniu mapy dysparycji należy określić odległość. Zadanie polega na znalezieniu zależności między wartością dysparycji, a odlegością.

### 3.6. Estymacja odległości

```

_, close_mask = cv2.threshold(filteredImg, 160, 255, cv2.THRESH_BINARY)
close_mask = cv2.morphologyEx(close_mask, cv2.MORPH_CLOSE, kernel)
contours, _ = cv2.findContours(close_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    if cv2.contourArea(cnt) > 500:
        x, y, w, h = cv2.boundingRect(cnt)

        roi_disp = disp[y:y + h, x:x + w].astype(np.float32)
        cx, cy = x + w // 2, y + h // 2
        sample_disp = disp[cy - 1:cy + 2, cx - 1:cx + 2].astype(np.float32)
        average_disp = np.mean(sample_disp[sample_disp > 0])

        if average_disp > 0:
            distance = -593.97 * average_disp**3 + 1506.8
            * average_disp**2 - 1373.1
            * average_disp + 522.06
        distance = np.around(distance * 0.01, decimals=2)

        if distance < 1.0:
            box_color = (0, 0, 255) if distance < 0.5 else (0, 255, 0)
            cv2.rectangle(filt_Color, (x, y), (x + w, y + h), box_color, 2)
            cv2.putText(filt_Color, f"{distance:.2f}m", (x, y - 10),
                       cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 2)

```

Obraz filteredImg (np. przefiltrowany obraz po detekcji koloru lub kształtu) jest zamieniany na obraz czarno-biały (binary mask), gdzie tylko jasne piksele > 160 zostają. Następnie nakłada się na ten obraz filtr domknięcia (closing).

Funkcja `findContours` wykrywa obiekty (kontury) w masce. `RETR_EXTERNAL` oznacza, że interesują nas tylko zewnętrzne kontury.

Pętla sprawdza każdy kontur. Kontury mniejsze niż 500 pikseli są ignorowane (eliminacja szumów).

Z mapy dysparcji obliczany jest bounding box wokół konturu, następnie wycinany jest fragment mapy dysparcji w tym obszarze. Z jego środka ( $cx, cy$ ) pobierany jest mały fragment ( $3 \times 3$ ) do analizy.

Obliczana jest średnia dysparcja. Z  $3 \times 3$  pikseli ze środka wybierane są tylko te z sensowną wartością (większą od zera).

Na podstawie średniej dysparcji obliczana jest odległość w metrach. Użyty jest wzór aproksymacyjny (polinom).

Wynik skalowany jest do metrów i zaokrąglany do 2 miejsc po przecinku.

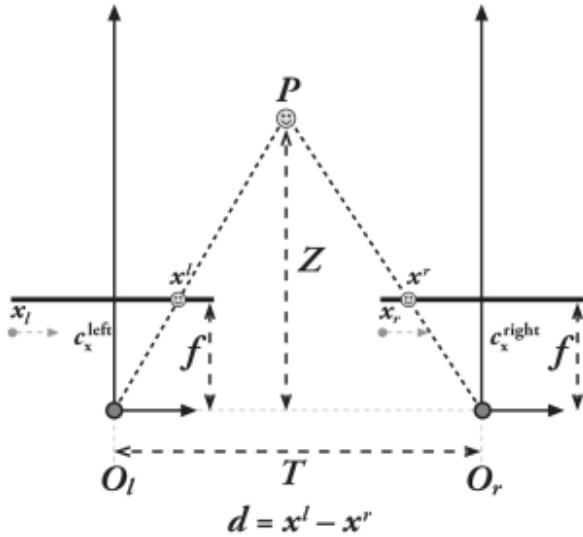
Jeśli obiekt znajduje się bliżej niż 1 metr, rysowana jest prostokątna ramka: czerwona jeśli  $< 0.5$  m, zielona jeśli  $0.5 - 1.0$  m.

Nad obiektem wyświetlana jest informacja o odległości w metrach.

Pomiar odległości dotyczy tylko odległości od 67 cm do 203 cm, aby uzyskać dobre wyniki. Precyzja pomiaru zależy również od jakości kalibracji. Kamera stereo był w stanie zmierzyć odległość do obiektu z precyzją  $\pm 5$  cm.

Zwrócona wartość to średnia dysparcji z macierzy  $9 \times 9$  pikseli.

### 3.6.1. Triangulacja



Rysunek 16: Schemat triangulacji.

W ostatnim kroku, triangulacji, zakłada się, że oba obrazy projekcji są współplaszczyznowe i że poziomy rząd pikseli lewego obrazu jest wyrównany z odpowiadającym mu obrazem prawego.

Punkt  $P$  leży w środowisku i jest pokazany na lewym i prawym obrazie na  $P_l$  i  $P_r$ , z odpowiadającymi im współrzędnymi odpowiadającymi współrzędnymi  $x^l$  i  $x^r$ . To pozwala nam wprowadzić nową wielkość  $d = x^l - x^r$ . Można zauważyć, że im dalej punkt  $P$ , tym mniejsza staje się wielkość  $d$ . Dysproporcja jest zatem odwrotnie proporcjonalna do odległości.

Do obliczenia odległości można użyć następującego wzoru można obliczyć:

$$Z = f * T / (x^l - x^r)$$

Można zauważyć, że istnieje nieliniowa zależność między rozbieżnością, a odległością. Jeśli rozbieżność jest bliska 0, małe różnice w rozbieżności prowadzą do dużych różnic w odległości. Zjawisko to ulega odwróceniu, gdy rozbieżność jest duża. Małe różnice dysproporcji nie prowadzą do dużych różnic odległości. Na tej podstawie można wywnioskować, że stereowizja ma wysoką rozdzielczość głębi, tylko dla obiektów znajdujących się blisko kamery.

## Rozdział 4

### Wnioski i możliwe ulepszenia

Z przeprowadzonej analizy uzyskanych obrazów można stwierdzić, że zastosowanie systemu antykolizyjnego opartego wyłącznie na analizie obrazu jest niewystarczające. Stereowizja posiada bowiem szereg ograniczeń, które wpływają na jej niezawodność, m.in.:

- problemy przy silnym oświetleniu oraz odbiciach, gdzie zmienność intensywności zakłóca proces dopasowania pikseli,
- brak możliwości dopasowania w obszarach ubogich w teksturę, takich jak białe ściany czy jednolite niebo,
- ograniczenie widzenia do jednej perspektywy, co powoduje występowanie martwych stref pomiędzy kamerami lub poza polem widzenia jednej z nich.

Aby stereowizja mogła funkcjonować w sposób użyteczny w systemach antykolizyjnych, konieczne jest zastosowanie dodatkowych czujników, takich jak wodoodporny czujnik ultradźwiękowy A02YYUW czy laserowy czujnik LiDAR. Należy jednak podkreślić, że również te urządzenia posiadają własne ograniczenia.

Kolejnym istotnym problemem jest liczba klatek na sekundę (ang. *Frames Per Second, FPS*). Analiza wyników wykazała, że osiągnięcie wartości około 4 FPS jest niewystarczające do zapewnienia pewnej i szybkiej reakcji systemu na wykrycie zagrożenia kolizją. Główną przyczyną tak niskiej wydajności jest fakt, iż obliczenia realizowane są na procesorze centralnym (CPU), który wykonuje instrukcje sekwencyjnie.

Dla porównania, przy zastosowaniu wydajniejszego CPU udało się uzyskać około 16 FPS, co wciąż stanowi wartość niewystarczającą.

CPU Name ‡	CPU Mark (higher is better) ‡
Intel Core i5-13600KF	37,491
Broadcom BCM2712	4,309

Rysunek 17: Porównanie CPU ARM i x86

Rozwiązaniem tego problemu może być zastosowanie odpowiedniego komputera jednoplatformowego z wydajnym procesorem graficznym (GPU), np. z rodziny Jetson firmy NVIDIA.



Rysunek 18: NVIDIA Jetson Nano B01.

Taka platforma umożliwia na przeniesienie omawianych obliczeń na GPU. Pozwoli to na znaczące zwiększenie liczby przetwarzanych klatek na sekundę, a tym samym poprawi responsywność systemu w kontekście wykrywania kolizji.

Możliwe ulepszenia dla programu:

Należy wziąć kształt filtra WLS i projektować go na mapie dysparycji. Ta projekcja zostanie następnie użyta do pobrania wszystkich wartości dysparycji, które znajdują się w tym kształcie, a wartość, która występuje najczęściej, zostanie ustawiona jako wartość dla całej powierzchni.

Użycie filtra bilateralnego na skalibrowanych obrazach przed wygenerowaniem mapy dysparycji, w ten sposób mogłoby być możliwe, aby nie stosować filtra WLS. Należy to sprawdzić, ale filtr WLS służy głównie do dobrego rozpoznawania krawędzi obiektów, być może istnieje lepsza metoda.

Aby skrócić czas obliczeń przy generowaniu mapy dysparycji, należy zmniejszyć skalibrowane obrazy za pomocą funkcji cv2.resize(cv2.INTER\_AbyREA). Należy przy tym pamiętać,

że wartości macierzy esencjalnych i fundamentalnych również muszą być proporcjonalnie zmniejszone.

Generowanie mapy głębokości mogłoby również być korzystne.

Używana równość prosta zawsze zwracałaby dokładną odległość do obiektu z większą precyzją i mniejszym nakładem pracy.



# Spis rysunków

1.	Płytki Raspberry Pi 5. <a href="https://www.hackatronic.com/wp-content/uploads/2024/03/Raspberry-Pi-5-Pinout-1210x642.jpg">https://www.hackatronic.com/wp-content/uploads/2024/03/Raspberry-Pi-5-Pinout-1210x642.jpg</a> . . . . .	6
2.	Kamera stereowizyjna. <a href="https://cell-kom.com/inne/21454-kamera-internetowa-full-hd-b16-1080p-5900217390350.html">https://cell-kom.com/inne/21454-kamera-internetowa-full-hd-b16-1080p-5900217390350.html</a> . . . . .	6
3.	Moduł zasilający. <a href="https://www.waveshare.com/wiki/UPS-HAT-(B)">https://www.waveshare.com/wiki/UPS-HAT-(B)</a> . . . . .	6
4.	Moduł przekaźnika. <a href="https://l1nq.com/hQOm9">https://l1nq.com/hQOm9</a> . . . . .	7
5.	Pojazd projektu. Opracowanie własne . . . . .	7
6.	Odwrocenie obrazu przez soczewkę. <a href="https://funsizephysics.com/use-light-turn-world-upside/">https://funsizephysics.com/use-light-turn-world-upside/</a> . . . . .	9
7.	Model kamery otworkowej. Learning OpenCV 3, O'Reilly, Str. 639 . . . . .	11
8.	Rodzaje dystorsji. <a href="https://beafoto.pl/dystorsja">https://beafoto.pl/dystorsja</a> . . . . .	12
9.	Widok wykrytych wierzchołków szachownic. <a href="https://learnopencv.com/making-a-low-cost-stereo-camera-using-opencv/">https://learnopencv.com/making-a-low-cost-stereo-camera-using-opencv/</a> . . . . .	18
10.	Model kamery stereo. Learning OpenCV 3, O'Reilly, Str. 709 . . . . .	20
11.	Wyszukiwanie pasujących bloków za pomocą StereoSGBM. Opracowanie własne. . . . .	25
12.	Pięć kierunków przeszukiwania w algorytmie StereoSGBM. Opracowanie własne. . . . .	26
13.	Wynik dla mapy dysproporcji. Opracowanie własne. . . . .	26
14.	Przykład filtra zamkającego. <a href="https://docs.opencv.org/3.4/d9/d61/tutorial-pymorphological-ops.html">https://docs.opencv.org/3.4/d9/d61/tutorial-pymorphological-ops.html</a> . . . . .	27
15.	Wynik mapy dysproporcji po filtrze zamkającym. Opracowanie własne. . . . .	27
16.	Schemat triangulacji. Learning OpenCV 3, O'Reilly, Str. 705 . . . . .	31
17.	Porównanie CPU ARM i x86. <a href="https://www.cpubenchmark.net/">https://www.cpubenchmark.net/</a> . . . . .	34



# Bibliografia

- [1] Fotografia otworkowa. Jak zrobić zdjęcie bez specjalistycznego aparatu fotograficznego?, <https://gaudemater.pl/fotografia-otworkowa/>, 2021-04-23.
- [2] Shawn Ingersoll, Derek Boyd, Kilen Murphy, Khara Plicanic, Anna Goellner, Zapoznanie z pojęciem i zastosowaniami ogniskowej, <https://www.adobe.com/pl/creativecloud/photography/discover/focal-length.html>.
- [3] Richard Hartley, Andrew Zisserman, Multiple View Geometry in Computer Vision, [https://www.r-5.org/files/books/computers/algo-list/image-processing/vision/Richard\\_Hartley\\_Andrew\\_Zisserman-Multiple\\_View\\_Geometry\\_in\\_Computer\\_Vision-EN.pdf](https://www.r-5.org/files/books/computers/algo-list/image-processing/vision/Richard_Hartley_Andrew_Zisserman-Multiple_View_Geometry_in_Computer_Vision-EN.pdf), 2004.
- [4] Dave Christian, Brown's Distortion Model and How To Use It  
<https://www.foamcoreprint.com/blog/what-are-calibration-targets>, 2023-02-20.
- [5] John Lambert, Stereo and Disparity, <https://johnwlambert.github.io/stereo/>.
- [6] Baeldung authors, Disparity Map in Stereo Vision,  
<https://www.baeldung.com/cs/disparity-map-stereo-vision>, 2025-03-26.
- [7] Rajesh Rao, Stereo and 3D Vision,  
<https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf>.
- [8] Adrian Kaehler, Gary Bradski, Learning OpenCV 3, O'Reilly, 2017-11.
- [9] Dystorsja w fotografii, <https://beafoto.pl/dystorsja>, 2021-01.
- [10] Kaustubh Sadekar, Making A Low-Cost Stereo Camera Using OpenCV,  
[https://learnopencv.com/making\\_a\\_low\\_cost\\_stereo\\_camera\\_using\\_opencv/](https://learnopencv.com/making_a_low_cost_stereo_camera_using_opencv/), 2021-01-11.

- [11] Sushma Sri, Motion Estimation using Block Matching Algorithm,  
<https://media.neliti.com/media/publications/237501-motion-estimation-using-block-matching-a-4e613693.pdf>, 2018-05.
- [12] Renzhi Mao, Kaijie Wei, Hideharu Amano, Yuki Kuno, Masatoshi Arai, Weighted Least Square Filter for Improving the Quality of Depth Map on FPGA,  
<http://www.ijnc.org/index.php/ijnc/article/view/291>, 2022-07.
- [13] Heiko Hirschmüller [https://en.wikipedia.org/wiki/Semi\\_global\\_matching](https://en.wikipedia.org/wiki/Semi_global_matching), 2005.
- [14] Ralph Hartley [https://en.wikipedia.org/wiki/Hartley\\_function](https://en.wikipedia.org/wiki/Hartley_function), 1928.
- [15] Jean-Yves Bouguet <https://robots.stanford.edu/cs223b04/JeanYvesCalib/>.