



**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 Zajączkowski  
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
<b>3. Stereowizja</b>	15
3.1. Kalibracja	15
3.2. Rektyfikacja stereo	17
3.2.1. Geometria epipolarna	18
3.2.2. Macierz fundamentalna i esencjalna	18
3.2.3. Macierz obrotu i wektor translacji	19
3.2.4. Algorytm Hartley'a	19
3.2.5. Algorytm Bouguet'a	20
3.3. Mapa dysparycji	20
3.3.1. Algorytm StereoSGBM	21
3.3.2. Algorytm StereoBM	23
3.4. Filtr WLS	25
3.4.1. Filtr zamkający	26
3.5. Triangulacja	26
<b>4. Funkcjonalność projektu</b>	29
<b>5. Wnioski i możliwe ulepszenia</b>	39
<b>Spis rysunków</b>	40
<b>Bibliografia</b>	43



# 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.

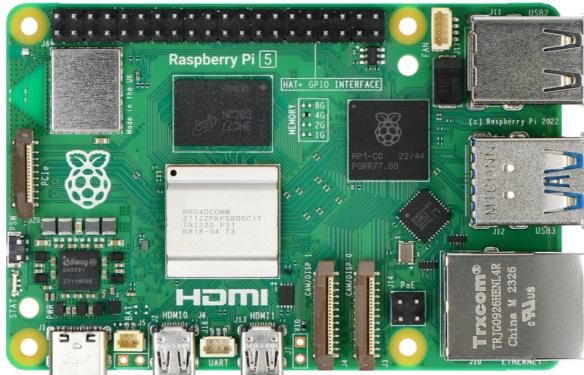
Celem niniejszego projektu jest implementacja oraz ocena działania **systemu antykolizyjnego** opartego na przetwarzaniu obrazu stereoskopowego w czasie rzeczywistym. Główną platformą obliczeniową jest **Raspberry Pi 5** – komputer jednoplatformowy nowej generacji, oferujący około 50% wyższą wydajność 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.



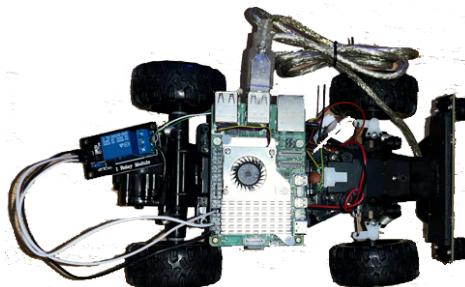
Rysunek 3: Moduł zasilający.

- **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.

Większa wartość paralaksy oznacza mniejszą odległość obiektu od kamery.

Opracowany system został zaimplementowany w języku Python z wykorzystaniem biblioteki OpenCV do przetwarzania obrazów oraz biblioteki `rpi.lgpio` do obsługi pinów wejścia/wyjścia *Raspberry Pi*.

Typowy proces obliczania głębi metodą stereowizyjną obejmuje następujące kluczowe kroki:

1. **Rektyfikacja obrazów** – transformacja obrazów w taki sposób, aby linie epipolarne były równoległe i współłaszczyznowe względem osi  $Y$ , co umożliwia wyszukiwanie korespondencji wyłącznie wzdłuż osi  $X$ .
2. **Wyszukiwanie korespondencji** – identyfikacja odpowiadających sobie punktów na obrazach lewym i prawym, prowadząca do wygenerowania *mapy dysparycji* przedstawiającej różnice położenia na osi  $X$ .
3. **Filtr WLS** – filtr wygładzający, który redukuje szумy w mapie dysparycji, zachowując jednocześnie ostrość krawędzi obiektów.
4. **Triangulacja** – przekształcenie mapy dysparycji na mapę głębi przy wykorzystaniu parametrów geometrycznych układu kamer.

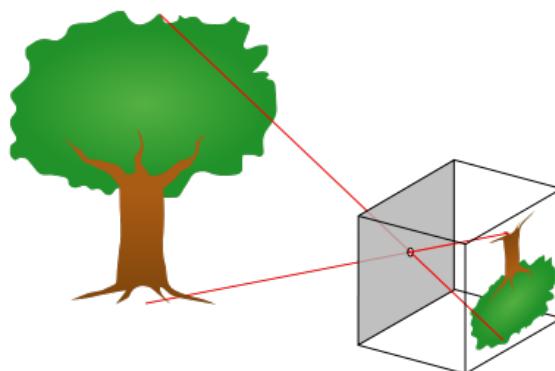
System uruchamiany jest automatycznie wraz ze startem systemu operacyjnego *Raspberry Pi*. Moduł przekaźnikowy domyślnie pozostaje w stanie odłączenia, a po aktywacji systemu obie kamery są inicjalizowane w trybie pracy ciągłej.

## 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:

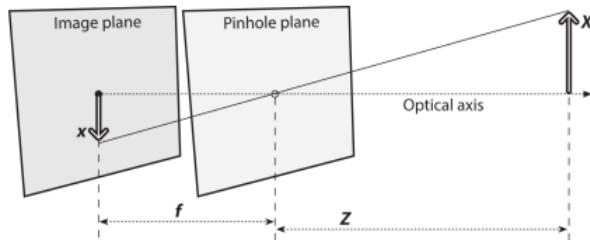
$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = f \cdot \frac{X}{Z}$$

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:

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

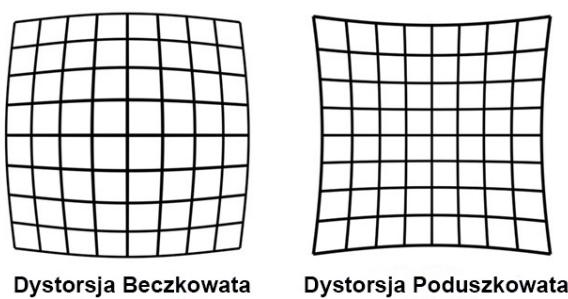
Macierz  $P = M[R|t]$  nazywana jest macierzą projekcji kamery i zawiera zarówno informacje o parametrach wewnętrznych kamery (macierz  $M$ ), jak i jej położeniu i orientacji w przestrzeni (macierze  $R$  i  $t$ ).

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.

## 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

### Stereowizja

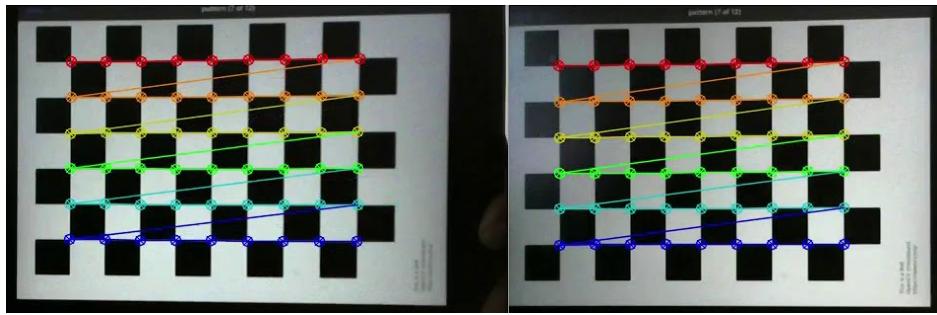
Stereowizja, nazywana również wizją stereoskopową, jest techniką pozyskiwania informacji przestrzennych na podstawie dwóch lub więcej obrazów tej samej sceny, zarejestrowanych z różnych punktów obserwacyjnych. Metoda ta, inspirowana sposobem postrzegania przestrzeni przez ludzkie oczy, stanowi jedno 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.

#### 3.1. Kalibracja

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

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().

Macierze kamer lewej i prawej uzyskane po rektyfikacji przedstawiają się następująco:

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

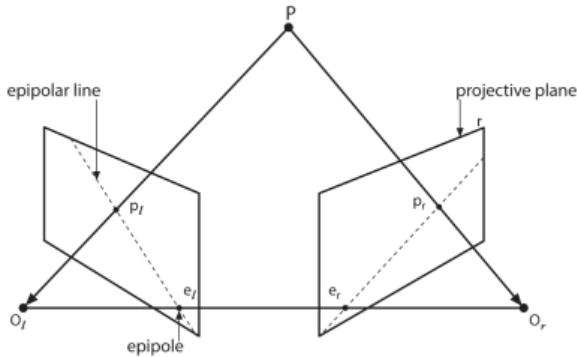
Aby wyznaczyć wzajemne położenie kamer względem siebie (rotację i translację), wykorzystywana jest funkcja cv2.stereoCalibrate(), która pozwala określić pełną konfigurację geometryczną układu stereo. Parametry te są niezbędne do poprawnego przekształcenia obrazów oraz dalszej analizy głębi, np. przy 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 znaczowo 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 powyżej przedstawiono uproszczony model kamery stereo zbudowanej z dwóch kamer otockowych. 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:

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

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 (niezrektyfikowaną) macierz kamery  $M$ .

### 3.2.4. 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.

### 3.2.5. 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 precyzję, lecz jest wrażliwa na błędy kalibracji (np. niedokładne wykrycie wzorca).

$$M_L^{\text{RECT}} = \begin{bmatrix} 791 & 0 & 390.6 \\ 0 & 791 & 194.4 \\ 0 & 0 & 1 \end{bmatrix} \quad M_R^{\text{RECT}} = \begin{bmatrix} 791 & 0 & 390.6 \\ 0 & 791 & 194.4 \\ 0 & 0 & 1 \end{bmatrix}$$

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

## 3.3. Mapa dysparacji

Mapa dysparacji [6] to obraz, w którym każdemu pikselowi przypisywana jest wartość reprezentująca przesunięcie (różnicę pozycji) pomiędzy odpowiadającymi sobie punktami w obrazie lewym i prawym. Im większe przesunięcie (czyli dysparycja), tym bliżej znajduje się

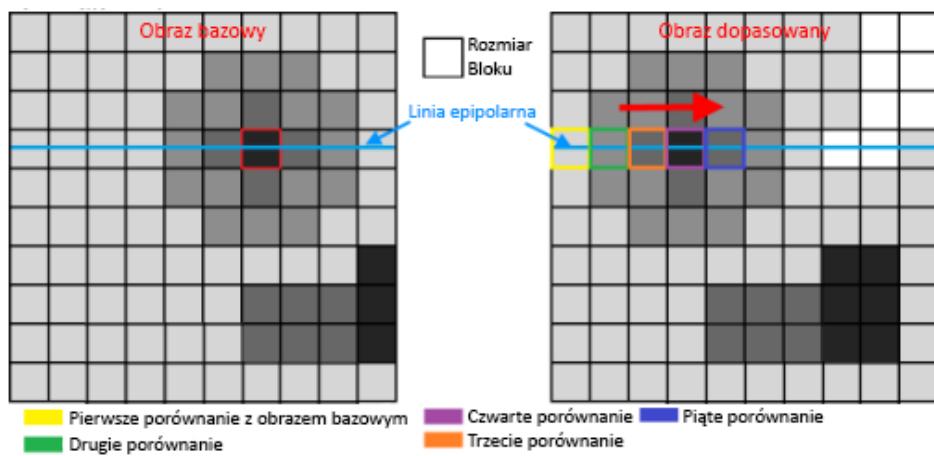
dany obiekt względem kamery. Taka mapa stanowi podstawę do obliczenia mapy głębokości, która pozwala oszacować odległość poszczególnych punktów sceny od obserwatora.

### 3.3.1. Algorytm StereoSGBM

Jeden z algorytmów do obliczenia mapy dysparcji w bibliotece OpenCV jest StereoSGBM. Algorytm ten implementuje metodę półglobalnego dopasowania blokowego (ang. *Semi-Global Block Matching*) [13], która umożliwia estymację przesunięć pomiędzy odpowiadającymi sobie punktami w obrazach stereo — pochodzących z lewej i prawej kamery.

Podstawową ideą algorytmu jest porównywanie fragmentów (bloków) obrazów wzdłuż odpowiadających sobie linii epipolarnych, w celu znalezienia najlepszego dopasowania. Jednym z kluczowych parametrów wejściowych jest rozmiar bloku (*block size*), który określa wielkość lokalnego sąsiedztwa uwzględnianego podczas dopasowania. Dla wartości większych niż 1, algorytm operuje na blokach pikseli, a nie na pojedynczych pikselach, co pozwala zwiększyć odporność na szum kosztem precyzji w obszarach o drobnych szczegółach.

Zakładając, że kalibracja i rektyfikacja zostały przeprowadzone prawidłowo, dopasowanie odbywa się wzdłuż linii epipolarnych, które w zrektyfikowanych obrazach pokrywają się z liniami poziomymi. Dzięki temu proces wyszukiwania korespondencji ogranicza się do jednej osi (poziomej), co znaczaco redukuje złożoność obliczeniową. Przykładowo, blok o współrzędnych (4, 7) w obrazie referencyjnym (np. lewym) porównywany jest z blokami w tej samej linii poziomej obrazu dopasowywanego (np. prawego), tj. (4, *i*).

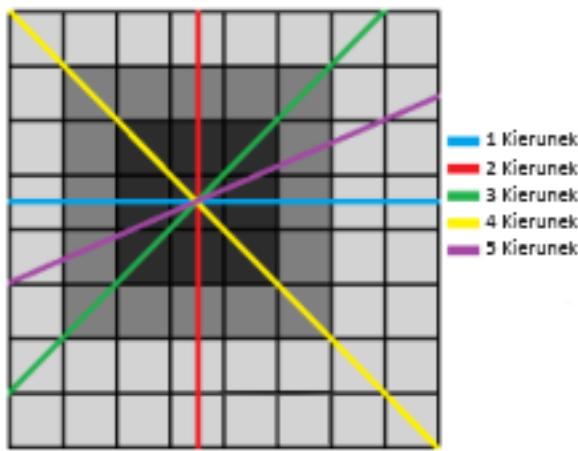


Rysunek 11: Wyszukiwanie pasujących bloków za pomocą StereoSGBM.

Dopasowanie oceniane jest na podstawie funkcji koszta, a im mniejsza wartość koszta, tym większe prawdopodobieństwo, że porównywane bloki odpowiadają temu samemu punktowi

w przestrzeni trójwymiarowej. W przedstawionym przykładzie najwyższe dopasowanie dla bloku (4, 7) uzyskano względem bloku (4, 4) w obrazie prawym.

W przeciwieństwie do metod opierających się wyłącznie na lokalnych dopasowaniach, algorytm StereoSGBM minimalizuje energię globalną, uwzględniając dodatkowe kierunki propagacji dopasowania. W implementacji OpenCV analizowanych jest pięć kierunków, co pozwala ograniczyć ryzyko lokalnych błędów dopasowania i poprawia spójność wyników.



Rysunek 12: Pięć kierunków przeszukiwania w algorytmie StereoSGBM.

Wartość dysparcji obliczana jest jako różnica współrzędnych poziomych piksela (lub bloku) w obrazie lewym i odpowiadającej mu pozycji w obrazie prawym. Zgodnie z zasadą triangulacji, większa wartość dysparcji oznacza, że obiekt znajduje się bliżej kamery. W praktyce przyjmuje się wartość bezwzględną tej różnicy.

Algorytm operuje zazwyczaj na obrazach w odcieniach szarości, co znaczco zmniejsza czas obliczeń. Możliwe jest użycie obrazów kolorowych, jednak wiąże się to ze zwiększoną zapotrzebowaniem na moc obliczeniową, bez istotnej poprawy jakości wyników.

Po skonfigurowaniu obiektu StereoSGBM, obliczenie mapy dysparcji następuje poprzez wywołanie metody `compute()`. Wynikowy obraz przedstawia poziomą różnicę położenia punktów w obrazach stereo, co stanowi podstawę do dalszej rekonstrukcji głębi.



Rysunek 13: Rezultat mapy dysparcji StereoSGBM.

### 3.3.2. Algorytm StereoBM

Alternatywą dla metody StereoSGBM jest algorytm StereoBM (ang. *Block Matching*) [11], będący jedną z najstarszych i najprostszych implementacji dopasowania stereoskopowego w bibliotece OpenCV. Jego działanie opiera się na zasadzie bezpośredniego porównywania lokalnych bloków pikseli pomiędzy obrazem lewym i prawym wzdłuż odpowiadających sobie linii epipolarnych. StereoBM nie wykorzystuje bezpośrednio parametrów kalibracyjnych w procesie dopasowania – działa tylko na rektyfikowanych obrazach.

Proces dopasowania rozpoczyna się od wyboru bloku referencyjnego w obrazie lewym. Dla każdego takiego bloku algorytm przeszukuje wzdłuż tej samej linii epipolarnej w obrazie prawym obszar o zadanym zakresie dysparcji, aby znaleźć najbardziej podobny fragment. Do oceny podobieństwa stosowana jest prosta funkcja kosztu, najczęściej **Sum of Absolute Differences (SAD)**, definiowana jako:

$$\text{SAD}(x, y, d) = \sum_{i=-\frac{w}{2}}^{\frac{w}{2}} \sum_{j=-\frac{h}{2}}^{\frac{h}{2}} |I_L(x + i, y + j) - I_R(x + i - d, y + j)|$$

gdzie:

- $(x, y)$  – współrzędne środka bloku w obrazie lewym,
- $d$  – wartość przesunięcia (kandydat na dysparcję),
- $w, h$  – szerokość i wysokość bloku dopasowania,
- $I_L, I_R$  – intensywności pikseli w obrazie lewym i prawym.

Dysparcja dla danego punktu wybierana jest jako wartość  $d$ , dla której funkcja kosztu osiąga minimum.

Algorytm ten analizuje wyłącznie dopasowania lokalne, bez uwzględnienia kontekstu globalnego, co odróżnia go od metody StereoSGBM, która implementuje optymalizację w wielu kierunkach i minimalizuje energię globalną. Brak optymalizacji globalnej skutkuje mniejszą dokładnością w obszarach jednorodnych (bez wyraźnej tekstury), w cieniu oraz przy obecności szumów.

#### Najważniejsze różnice między StereoBM a StereoSGBM:

- **Metoda dopasowania:** StereoBM stosuje lokalne dopasowanie blokowe na podstawie funkcji SAD, natomiast StereoSGBM rozszerza dopasowanie o koszt gradientu intensywności i optymalizację półglobalną.
- **Kontekst globalny:** StereoBM ignoruje zależności przestrzenne między sąsiadującymi pikselami, podczas gdy StereoSGBM propaguje koszty w wielu kierunkach, co redukuje błędy dopasowania w jednorodnych obszarach.
- **Jakość mapy dysparcji:** Wyniki StereoBM są zazwyczaj gorsze w obszarach o niskim kontraście i w cieniu. StereoSGBM lepiej radzi sobie z tymi przypadkami dzięki dodatkowym ograniczeniom ciągłości dysparcji.
- **Złożoność obliczeniowa:** StereoBM jest znaczco szybszy i mniej zasobozerny, co czyni go atrakcyjnym rozwiązaniem w systemach o ograniczonej mocy obliczeniowej, natomiast StereoSGBM wymaga więcej pamięci i czasu obliczeń.

Ze względu na prostotę algorytmu i brak optymalizacji globalnej, wyniki uzyskane przy użyciu StereoBM są bardziej podatne na błędy dopasowania (tzw. szумy dysparcji), szczególnie w obszarach pozbawionych tekstury lub przy zakłóceniach oświetlenia.

StereoBM jest powszechnie stosowany w systemach wymagających przetwarzania w czasie rzeczywistym przy ograniczonych zasobach, np. w robotyce mobilnej lub systemach wizyjnych dla pojazdów autonomicznych, gdy wymagana jest szybka, ale przybliżona estymacja głębi.



Rysunek 14: Rezultat mapy dysparcji uzyskanej metodą StereoBM.

### 3.4. Filtr WLS

W celu dalszego ograniczenia szumu w mapie dysparcji 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 precyzyjnie 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 prawnego 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`).

Tak wygląda rezultat filtra WLS:



Rysunek 15: Rezultat filtra WLS dla StereoSGBM i StereoBM.

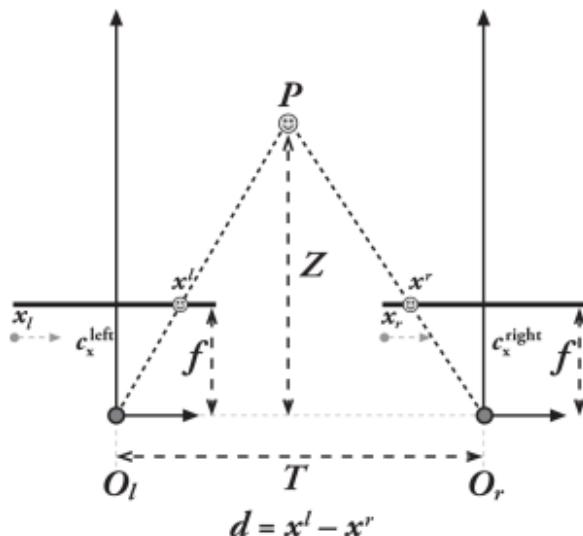
### 3.4.1. Filtr zamykający

Na uzyskanym obrazie 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 16: Przykład filtra zamykającego.

### 3.5. Triangulacja



Rysunek 17: 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 odlegością. 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

# Funkcjonalność projektu

```
# Termination criteria
term_criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
```

Na początku definiowane są kryteria zakończenia algorytmu optymalizacji (term\_criteria), które łączą dwa warunki:

- osiągnięcie maksymalnej liczby iteracji (30),
- osiągnięcie dokładności wyrażonej w zmianie pozycji punktu poniżej zadanego progu (0.001).

Kryteria te są następnie wykorzystywane w procedurze doprecyzowywania lokalizacji narożników szachownicy.

```
# Prepare object points
objp = np.zeros((9*6,3), np.float32)
objp[:, :2] = np.mgrid[0:9, 0:6].T.reshape(-1, 2)
```

Tworzona jest macierz punktów obiektowych (objp), która reprezentuje idealny, znany geometrycznie układ narożników szachownicy w przestrzeni 3D. Przyjęto, że szachownica ma wymiar  $9 \times 6$  pól (tj. 54 narożniki). Punkty są rozmiieszczane w płaszczyźnie  $Z = 0$ , dzięki czemu uzyskano dwuwymiarową siatkę punktów w przestrzeni obiektowej

```
# Arrays to store object points and image points from all images
objpoints= []      # 3d points in real world space
imgpointsR= []     # 2d points in image plane
imgpointsL= []

ChessImaR = None
ChessImaL = None

for i in range(0, 64):
    t = str(i)
```

```

ChessImaR = cv2.imread('calib_images/right_chessboard-' + t + '.png', 0)
ChessImaL = cv2.imread('calib_images/left_chessboard-' + t + '.png', 0)
if ChessImaR is None or ChessImaL is None:
    continue # Skip this iteration if loading failed
retR, cornersR = cv2.findChessboardCorners(ChessImaR, (9, 6), None)
retL, cornersL = cv2.findChessboardCorners(ChessImaL, (9, 6), None)
if retR and retL:
    objpoints.append(objp)
    cv2.cornerSubPix(ChessImaR, cornersR, (11, 11), (-1, -1), term_criteria)
    cv2.cornerSubPix(ChessImaL, cornersL, (11, 11), (-1, -1), term_criteria)
    imgpointsR.append(cornersR)
    imgpointsL.append(cornersL)

```

Program iteruje przez zestaw obrazów kalibracyjnych (64 par zdjęć lewej i prawej kamery).

Dla każdej pary:

- wczytywane są obrazy w odcieniach szarości,
- wykonywana jest detekcja narożników szachownicy (cv2.findChessboardCorners),
- w przypadku sukcesu, pozycje narożników są doprecyzowywane metodą subpixelsową (cv2.cornerSubPix),
- wykonywana jest detekcja narożników szachownicy (cv2.findChessboardCorners),
- współrzędne narożników (punkty obrazu) dodawane są do listy punktów obrazowych dla każdej kamery (imgpointsL, imgpointsR), przy jednoczesnym przypisaniu do odpowiadających im punktów obiektowych (objpoints).

```

# Calibration
retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(objpoints, imgpointsR, ChessImaR.shape[::-1], None, 0)
retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(objpoints, imgpointsL, ChessImaL.shape[::-1], None, 0)

```

Dla każdej z kamer przeprowadzana jest osobna kalibracja przy użyciu funkcji cv2.calibrateCamera.

Wynikiem są:

- macierze wewnętrzne kamer (mtxL, mtxR),
- współczynniki dystorsji (distL, distR),
- parametry związane z położeniem i orientacją kamery dla każdej pozycji wzorca (rvecs, tvecs).

```

OmtxR, roiR = cv2.getOptimalNewCameraMatrix(mtxR, distR, ChessImaR.shape[::-1], 1, ChessImaR.shape[::-1])
OmtxL, roiL = cv2.getOptimalNewCameraMatrix(mtxL, distL, ChessImaL.shape[::-1], 1, ChessImaL.shape[::-1])

```

Następnie obliczane są optymalne macierze rzutowania (cv2.getOptimalNewCameraMatrix), które minimalizują wpływ dystorsji przy jednoczesnym zachowaniu maksymalnego obszaru obrazu.

```

0mtxR, roiR = cv2.getOptimalNewCameraMatrix(mtxR, distR, ChessImaR.shape[::-1], 1, ChessImaR.shape[::-1])
0mtxL, roiL = cv2.getOptimalNewCameraMatrix(mtxL, distL, ChessImaL.shape[::-1], 1, ChessImaL.shape[::-1])

retS, MLS, dLS, MRS, dRS, R, T, E, F = cv2.stereoCalibrate(objpoints, imgpointsL, imgpointsR, mtxL, distL,

```

W ostatnim etapie wykonywana jest właściwa stereokalibracja przy użyciu funkcji cv2.stereoCalibrate. Proces ten, przy założeniu stałych parametrów wewnętrznych kamer (cv2.CALIB\_FIX\_INTRINSIC), estymuje:

- macierz rotacji  $R$ ,
- wektor translacji  $T$ ,
- macierz fundamentalną  $F$ ,
- macierz epipolarną  $E$ .

Parametry te określają wzajemne położenie i orientację obu kamer, a także zależności geometryczne między punktami obrazu zarejestrowanymi przez lewą i prawą kamerę.

```

rectify_scale= 0 # if 0 image cropped, if 1 image nor cropped
RL, RR, PL, PR, Q, roiL, roiR= cv2.stereoRectify(MLS, dLS, MRS, dRS, ChessImaR.shape[::-1], R, T, rectify_scale)
# last parameter is alpha, if 0= cropped, if 1= not cropped

```

Powyższy fragment kodu implementuje **algorytm Bougueta** (rektyfikacja kalibrowana), który wykorzystuje pełny zestaw parametrów kalibracyjnych kamer do wyznaczenia rzutów obrazów na wspólną płaszczyznę epipolarną. Funkcja cv2.stereoRectify korzysta z macierzy wewnętrznych kamer ( $M_{LS}, M_{RS}$ ), współczynników dystorsji ( $d_{LS}, d_{RS}$ ), a także parametrów opisujących wzajemne położenie kamer w przestrzeni ( $R, T$ ). Na tej podstawie estymowane są:

- macierze rektyfikacji ( $R_L, R_R$ ) dla lewej i prawej kamery,
- nowe macierze rzutowania ( $P_L, P_R$ ),
- macierz dysparcji do głębi  $Q$ , umożliwiająca odwzorowanie punktów obrazu w trójwymiarową przestrzeń,
- regiony zainteresowania (ROI), określające użytkownika obszar obrazu po rektyfikacji.

Parametr `rectify_scale` definiuje stopień przycięcia obrazu:

- $\alpha = 0$  – obrazy są przycięte w celu wyeliminowania obszarów pozostawionych informacji,
- $\alpha = 1$  – obrazy nie są przycinane, co pozwala zachować pełne pole widzenia, kosztem obecności obszarów pustych.

```

Left_Stereo_Map= cv2.initUndistortRectifyMap(MLS, dLS, RL, PL, ChessImaR.shape[::-1], cv2.CV_16SC2)
# cv2.CV_16SC2 this format enables us the programme to work faster
Right_Stereo_Map= cv2.initUndistortRectifyMap(MRS, dRS, RR, PR, ChessImaR.shape[::-1], cv2.CV_16SC2)

```

Następnie, przy pomocy funkcji `cv2.initUndistortRectifyMap`, obliczane są mapy przekształceń dla lewej i prawej kamery. Mapy te definiują sposób, w jaki każdy piksel obrazu wejściowego powinien zostać przesunięty w celu eliminacji znieksztalceń soczewki oraz dostosowania do nowego układu rektyfikowanego.

W kodzie generowane są dwie mapy:

- `Left_Stereo_Map` dla obrazu lewego,
- `Right_Stereo_Map` dla obrazu prawego.

Dane wyjściowe mają format `cv2.CV_16SC2`, co pozwala na szybsze działanie programu dzięki wykorzystaniu mapy przesunięć w formie stałoprzecinkowej.

Algorytm Hartleya [14] umożliwia rektyfikację obrazów stereo w przypadku braku wcześniejszej kalibracji układu kamer. W odróżnieniu od metody opartej na `stereoRectify`, która wymaga znajomości parametrów wewnętrznych i zewnętrznych kamer, tutaj proces bazuje wyłącznie na dopasowanych punktach korespondencyjnych pomiędzy obrazami lewym i prawym.

**Wykrycie i opis cech lokalnych** – w obu obrazach (lewy i prawy) detekcja cech przeprowadzana jest za pomocą algorytmu ORB, który zwraca punkty kluczowe oraz deskryptory opisujące ich lokalny kontekst.

```

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

```

**Dopasowanie punktów korespondencyjnych** – wykorzystując algorytm BFMatcher z metryką Hamminga, dopasowywane są deskryptory z obu obrazów. Wyniki sortowane są według odległości, co pozwala wybrać najlepiej dopasowane pary punktów.

```

# 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)

```

**Estymacja macierzy fundamentalnej** – na podstawie uzyskanych dopasowań estymowana jest macierz fundamentalna  $F$ , opisująca geometryczne powiązanie pomiędzy obrazami stereo. Wykorzystanie metody RANSAC zapewnia odporność na błędne dopasowania (outliery).

```
# 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)
```

**Filtracja inlierów** – do dalszych obliczeń wykorzystywane są jedynie punkty spełniające model geometryczny wynikający z macierzy fundamentalnej.

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

**Obliczenie homografii rektyfikujących** – funkcja stereoRectifyUncalibrated wyznacza transformacje homograficzne  $H_1$  i  $H_2$ , które przekształcają obrazy tak, aby epipolarne linie były wyrównane wzduż osi poziomej.

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

**Rektyfikacja obrazów** – poprzez zastosowanie przekształceń homograficznych ( $H_1$  i  $H_2$ ) uzyskuje się zrektyfikowane obrazy lewe i prawe, które mogą zostać użyte w dalszych etapach przetwarzania (np. obliczania mapy dysparcji).

```
# 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])
```

Poniższy fragment kodu przedstawia proces inicjalizacji algorytmu *Semi-Global Block Matching* (StereoSGBM) w bibliotece OpenCV. Algorytm ten jest wykorzystywany do estymacji mapy dysparcji na podstawie sparowanych obrazów stereo.

```
# Inicjalizacja StereoSGBM
block_size = 7
min_disp = 2
num_disp = 130 - min_disp
stereo = cv2.StereoSGBM_create(
    minDisparity = min_disp,
    numDisparities = num_disp,
    blockSize = block_size,
    uniquenessRatio = 10,
    speckleWindowSize = 100,
    speckleRange = 32,
    disp12MaxDiff = 5,
    P1 = 8 * 3 * block_size ** 2,
    P2 = 32 * 3 * block_size ** 2
)
```

Parametry algorytmu pełnią następujące funkcje:

- **minDisparity** - minimalna oczekiwana wartość dysparcji. Określa najmniejsze przesunięcie pikseli między obrazami lewym i prawym. Zwykle jest równa 0 lub niewielkiej wartości dodatniej.
- **numDisparities** - liczba poziomów dysparcji, które będą analizowane. Musi być wielokrotnością 16. Im większa wartość, tym większy zakres głębi można zarejestrować, kosztem wydajności obliczeniowej.
- **blockSize** - rozmiar bloku (*okna*), na podstawie którego porównywane są fragmenty obrazów. Duże wartości zwiększą odporność na szumy, ale pogarszają precyzję przy krawędziach obiektów.
- **uniquenessRatio** - minimalna różnica procentowa między najlepszym a drugim najlepszym dopasowaniem. Wyższe wartości pozwalają odrzucać niepewne wyniki.
- **speckleWindowSize** - maksymalny rozmiar obszaru złożonego z przypadkowych artefaktów (tzw. *speckles*), który zostanie usunięty.
- **speckleRange** - maksymalna dozwolona różnica dysparcji w obrębie speckle. Parametr ten pomaga eliminować niespójne obszary.
- **disp12MaxDiff** - maksymalna dopuszczalna różnica pomiędzy wynikami uzyskany mi w dopasowaniu lewo-prawo oraz prawo-lewo. Ma na celu zapewnienie spójności obliczeń.
- **P1** - kara za niewielkie zmiany dysparcji między sąsiednimi pikselami. Wartość ta jest proporcjonalna do liczby kanałów obrazu oraz kwadratu wielkości bloku.
- **P2** - kara za większe zmiany dysparcji, znaczco wyższa niż **P1**. Zapewnia gładkość mapy dysparcji poprzez redukcję gwałtownych skoków wartości.

Poniższy fragment kodu ilustruje zastosowanie metody *Block Matching* (StereoBM) do wyznaczania mapy dysparcji na podstawie obrazów stereo. Algorytm ten opiera się na lokalnym porównywaniu bloków pikseli pomiędzy lewym i prawym obrazem w celu oszacowania przesunięć odpowiadających sobie punktów.

```
# Inicjalizacja StereoBM
num_disp = 64 # musi być podzielne przez 16
block_size = 15 # większy blok = większa odporność na szumy

stereo = cv2.StereoBM_create(numDisparities=num_disp, blockSize=block_size)
```

W powyższym kodzie najpierw inicjalizowany jest obiekt klasy StereoBM, przy czym parametr numDisparities określa maksymalny zakres przesunięć (dysparycji), a blockSize definiuje rozmiar lokalnego okna porównawczego. Większy rozmiar bloku zwiększa odporność na szумy, jednak odbywa się to kosztem precyzji na krawędziach i przy drobnych szczegółach obrazu.

```
# Tworzenie dopasowania w kierunku prawym
stereoR = cv2.ximgproc.createRightMatcher(stereo)
```

Potrzebna jest druga instancja dopasowania (stereoR) z wykorzystaniem funkcji cv2.ximgproc.createRightMatcher(). Jest to wariant algorytmu obliczający dopasowanie obrazu prawego względem lewego, co umożliwia późniejsze zastosowanie filtrów spójności.

W celu poprawy jakości uzyskanej mapy dysparycji stosuje się filtrację post-procesową. W poniższym fragmencie kodu wykorzystano filtrację metodą *Weighted Least Squares* (WLS), dostępną w module cv2.ximgproc. Technika ta redukuje artefakty i szumy w mapie dysparycji, zwłaszcza w rejonach o niskim kontraste i w pobliżu krawędzi obiektów.

```
lmbda = 80000
sigma = 1.8
visual_multiplier = 1.0
wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=stereo)
wls_filter.setLambda(lmbda)
wls_filter.setSigmaColor(sigma)
```

Parametr lambda ( $\lambda$ ) kontroluje siłę wygładzania – większa wartość powoduje bardziej globalne uśrednianie mapy dysparycji, co redukuje zakłócenia, lecz może prowadzić do utraty szczegółów. Z kolei parametr sigma reguluje wpływ informacji barwnej podczas filtracji; większe wartości umożliwiają zachowanie ciągłości na jednolitych obszarach, a jednocześnie lepsze odwzorowanie krawędzi obiektów.

Zmienna visual\_multiplier określa współczynnik skalujący, wykorzystywany wyłącznie do wizualizacji przefiltrowanej mapy, nie wpływając na same wyniki obliczeń.

```
Cam = cv2.VideoCapture(0)
Cam.set(cv2.CAP_PROP_FRAME_WIDTH, 1100)
Cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 270)
Cam.set(cv2.CAP_PROP_FPS, 60)

executor = ThreadPoolExecutor(max_workers=4)

focal_length_px = PL[0, 0]
baseline_m = abs(T[0][0]) / 100

# --- Parametry wykrywania obiektów --- #
MIN_DISTANCE_TRIGGER = 0.65
MAX_DISTANCE_RELEASE = 0.7
DISPARITY_RANGE = (1.0, 150.0)
```

```

CONTOUR_AREA_THRESHOLD = 500
DISP_AVG_HISTORY = 5

distance_history = deque(maxlen=DISP_AVG_HISTORY)

while True:
    current_time = time.time()
    ret, frame = Cam.read()
    if not ret:
        print("Failed to capture frame. Exiting...")
        break

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

    left_frame = frame[:, :mid]
    right_frame = frame[:, mid:]

    # --- Remap rectified images ---
    Left_nice = executor.submit(cv2.remap, left_frame, Left_Stereo_Map[0], Left_Stereo_Map[1],
                                interpolation=cv2.INTER_LANCZOS4, borderMode=cv2.BORDER_CONSTANT).result()
    Right_nice = executor.submit(cv2.remap, right_frame, Right_Stereo_Map[0], Right_Stereo_Map[1],
                                interpolation=cv2.INTER_LANCZOS4, borderMode=cv2.BORDER_CONSTANT).result()

    # --- Convert to grayscale ---
    gray_left = executor.submit(cv2.cvtColor, Left_nice, cv2.COLOR_BGR2GRAY).result()
    gray_right = executor.submit(cv2.cvtColor, Right_nice, cv2.COLOR_BGR2GRAY).result()

    # --- Compute disparity maps ---
    futureL = executor.submit(lambda: stereo.compute(gray_left, gray_right).astype(np.float32) / 16.0)
    futureR = executor.submit(lambda: stereoR.compute(gray_right, gray_left).astype(np.float32) / 16.0)
    dispL = futureL.result()
    dispR = futureR.result()

    # --- Apply WLS filter ---
    filtered_disp = wls_filter.filter(dispL, gray_left, None, dispR)

    # --- Closing Filter ---
    disp_closed = cv2.morphologyEx(filtered_disp, cv2.MORPH_CLOSE, kernel)

    disp_vis = cv2.normalize(disp_closed, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    # --- ROI image to analyze ---
    disp_height, disp_width = disp_vis.shape
    roi_width = disp_width // 3 # 1/3 width
    roi_height = disp_height // 2 # 1/2 height
    roi_x = (disp_width - roi_width + 150) // 2
    roi_y = (disp_height - roi_height) // 2
    roi_rect = (roi_x, roi_y, roi_width, roi_height)

    # --- Visualize ROI ---
    cv2.rectangle(disp_closed, (roi_x, roi_y), (roi_x + roi_width, roi_y + roi_height), (0, 255, 255), 2)

    # Detect close objects via thresholding
    _, close_mask = cv2.threshold(disp_closed, 160, 255, cv2.THRESH_BINARY)

```

```

# Restrict mask to ROI only
roi_mask = np.zeros_like(close_mask)
roi_mask[roi_y:roi_y + roi_height, roi_x:roi_x + roi_width] = close_mask[roi_y:roi_y + roi_height]

# Find contours in ROI restricted region
contours, _ = cv2.findContours(roi_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

current_stop_flag = False

for cnt in contours:
    if cv2.contourArea(cnt) < CONTOUR_AREA_THRESHOLD:
        continue

    x, y, w, h = cv2.boundingRect(cnt)
    cx, cy = x + w // 2, y + h // 2

    # Make sure coordinates are within image bounds
    if cy - 1 < 0 or cy + 2 > dispL.shape[0] or cx - 1 < 0 or cx + 2 > dispL.shape[1]:
        continue

    roi_disp = dispL[cy - 1:cy + 2, cx - 1:cx + 2]
    valid_disp = roi_disp[(roi_disp > DISPARITY_RANGE[0]) & (roi_disp < DISPARITY_RANGE[1])]

    if valid_disp.size == 0:
        continue

    avg_disp = np.median(valid_disp)
    distance = (focal_length_px * baseline_m) / avg_disp
    distance_history.append(distance)
    smoothed_distance = np.median(distance_history)

    # Apply hysteresis logic for stability
    if smoothed_distance < MIN_DISTANCE_TRIGGER:
        current_stop_flag = True
        box_color = (0, 0, 255) if smoothed_distance < 0.5 else (0, 255, 0)
        cv2.rectangle(disp_closed, (x, y), (x + w, y + h), box_color, 2)
        cv2.putText(disp_closed, f"{smoothed_distance:.2f}m", (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 2)
        break

# Trigger GPIO only once per frame
# if current_stop_flag:
#     lgpio.gpio_write(chip, PIN, 0) # Stop
# elif len(distance_history) == DISP_AVG_HISTORY and smoothed_distance > MAX_DISTANCE_RELEASE:
#     lgpio.gpio_write(chip, PIN, 1) # Move forward

```



## Rozdział 5

### 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 18: Porównanie CPU ARM i x86

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



Rysunek 19: NVIDIA Jetson Nano B01.

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

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.

# 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> . . . . .	16
10.	Model kamery stereo. Learning OpenCV 3, O'Reilly, Str. 709 . . . . .	18
11.	Wyszukiwanie pasujących bloków za pomocą StereoSGBM. Opracowanie własne. . . . .	21
12.	Pięć kierunków przeszukiwania w algorytmie StereoSGBM. Opracowanie własne. . . . .	22
13.	Rezultat mapy dysparcji StereoSGBM. Opracowanie własne. . . . .	23
14.	Rezultat mapy dysparcji StereoBM. Opracowanie własne. . . . .	24
15.	Rezultat filtra WLS dla StereoSGBM i StereoBM. Opracowanie własne. . . . .	25
16.	Przykład filtra zamkającego. <a href="https://www.graphicsmill.com/docs/gm/minimum-maximum-median-filters.htm">https://www.graphicsmill.com/docs/gm/minimum-maximum-median-filters.htm</a> . . . . .	26
17.	Schemat triangulacji. Learning OpenCV 3, O'Reilly, Str. 705 . . . . .	26
18.	Porównanie CPU ARM i x86. <a href="https://www.cpubenchmark.net/">https://www.cpubenchmark.net/</a> . . . . .	40



# 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/>.