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

Rozdział 1

Wstęp

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

Współczesne systemy wizyjne są powszechnie wykorzystywane m.in. w medycynie (np. do analizy obrazów RTG, MRI), przemyśle (automatyczna kontrola jakości i detekcja defektów), systemach bezpieczeństwa (rozpoznawanie twarzy, analiza zachowań), jak również w robotyce i pojazdach autonomicznych, gdzie umożliwiają detekcję przeszkód, planowanie trasy oraz podejmowanie decyzji w czasie rzeczywistym. Szczególne znaczenie ma tu implementacja systemów wizyjnych na platformach o ograniczonych zasobach obliczeniowych, takich jak mikrokontrolery czy jednopłytkowe komputery.

Celem niniejszej pracy jest ocena możliwości implementacji **systemu antykolizyjnego** działającego w czasie rzeczywistym na platformie **Raspberry Pi 5**, która reprezentuje nową generację tanich komputerów jednopłytkowych będąca $\sim 50\%$ szybszą od generacji niżej. Analizowana będzie efektywność przetwarzania obrazu w kontekście wykrywania przeszkód z wykorzystaniem dwóch różnych podejść wizyjnych:

- **wizji monokularnej** – wykorzystującej pojedynczą kamerę do analizy obrazu i estymacji odległości na podstawie metod geometrycznych lub cech wizualnych,
- **stereowizji** – opierającej się na dwóch zsynchronizowanych kamerach, umożliwiającej generowanie mapy dysparycji, a tym samym bezpośrednie oszacowanie głębokości

(odległości) do poszczególnych punktów sceny.

W ramach pracy przeprowadzona zostanie implementacja oraz porównanie obu podejść pod względem:

- dokładności detekcji przeszkód i wyznaczania odległości,
- obciążenia obliczeniowego procesora i możliwości działania w czasie rzeczywistym,
- potrzeb i skuteczności różnych technik optymalizacji, takich jak: ograniczenie rozdzielczości obrazu, selektywna segmentacja, uproszczone modele przetwarzania, redukcja liczby operacji konwolucyjnych czy zastosowanie sprzętowej akceleracji.

Ważnym aspektem analizy będzie również wpływ rektyfikacji obrazów oraz poprawna **kalibracja układu kamer** – zarówno wewnętrzna (parametry optyczne), jak i zewnętrzna (położenie względem siebie). W tym celu wykorzystana zostanie m.in. **metoda kalibracji kamer zaproponowana przez Bougueta**, implementowana w bibliotece OpenCV.

Ze względu na rosnącą potrzebę implementacji lekkich, energooszczędnych oraz tanich systemów wizyjnych, które mogłyby znaleźć zastosowanie m.in. w autonomicznych robotach mobilnych, pojazdach, dronach czy inteligentnych systemach IoT, istotnym problemem badawczym staje się określenie, czy nowoczesne platformy klasy *embedded* – takie jak Raspberry Pi 5 – są w stanie sprostać wymaganiom przetwarzania obrazów w czasie rzeczywistym.

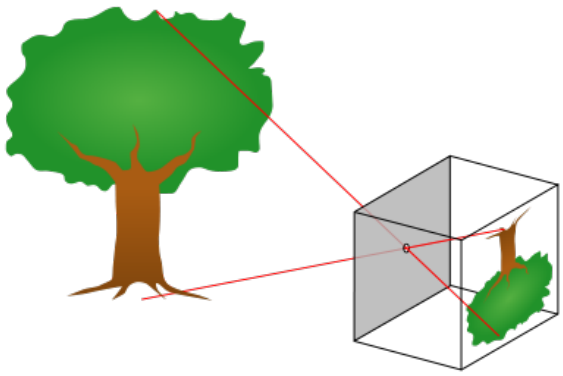
Zbadanie tego zagadnienia wymaga analizy kompromisów między jakością przetwarzania obrazu a wydajnością obliczeniową oraz porównania alternatywnych metod wizyjnych pod kątem dokładności i kosztów implementacyjnych. W szczególności stereowizja, choć znacznie bardziej kosztowna obliczeniowo niż analiza monokularna, oferuje bogatsze informacje o scenie, co może znacząco zwiększyć niezawodność detekcji przeszkód i obliczania odległości.

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”. 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 wewnątrz kamery, tworząc odwrócony obraz. Poniższa ilustracja przedstawia tę zasadę.



Rysunek 1: 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.

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:

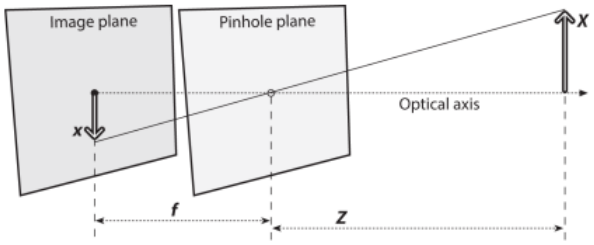
$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = f \cdot \frac{X}{Z}$$

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 2: 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:

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

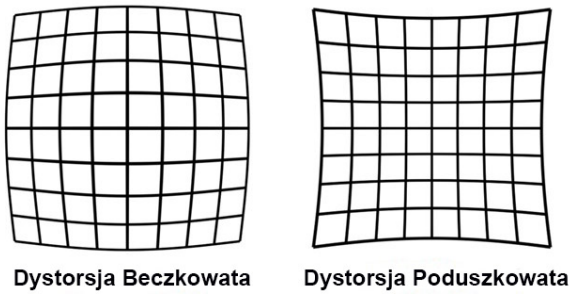
Macierz $P = M[R|t]$ nazywana jest macierzą projekcji kamery i zawiera zarówno informacje o parametrach wewnętrznych kamery (macierz M), jak i jej położeniu i orientacji w przestrzeni (macierze R i t).

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 3: Rodzaje dystorsji

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

Aby móc opisać i skorygować te zniekształcenia, 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.

Zniekształcenia promieniowe i rozwinięcie Taylora

Zniekształcenia promieniowe (ang. *radial distortion*) 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 zniekształcenia 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 zwiększają złożoność obliczeń. Po uwzględnieniu tej funkcji korekta zniekształcenia promieniowego ma postać:

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

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:

$$x_{\text{tangential}} = x + \left[2p_1xy + p_2(r^2 + 2x^2)\right]$$
$$y_{\text{tangential}} = y + \left[p_1(r^2 + 2y^2) + 2p_2xy\right]$$

Pełna korekta punktu znormalizowanego

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

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

Powrót do współrzędnych obrazu

Na końcu przekształcamy punkt z powrotem do układu pikselowego:

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

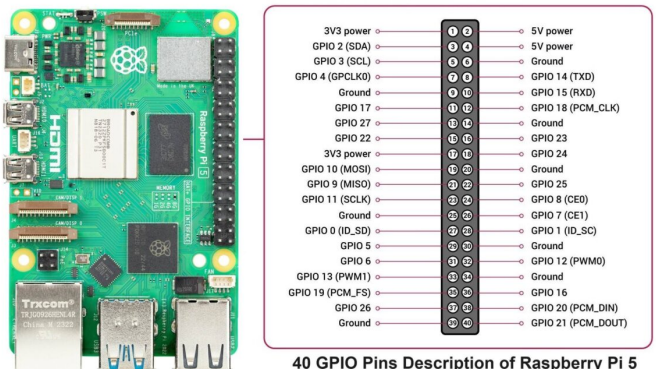
W ten sposób otrzymujemy ostateczną, skorygowaną pozycję punktu obrazu, która uwzględnia wpływ obu typów zniekształceń optycznych.

2.3. Dane z obrazu kamery

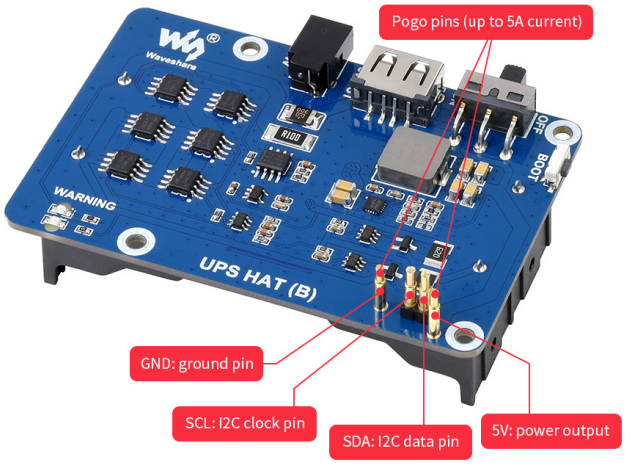
Znormalizowane współrzędne (x, y) oraz współrzędne pikselowe (u, v) służą do różnych celów, ale są ze sobą ściśle powiązane. Współrzędne znormalizowane są używane wszędzie tam, gdzie istotna jest struktura geometryczna sceny i relacje przestrzenne – np. w algorytmach rekonstrukcji 3D, lokalizacji kamery 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

Opis projektu



Rysunek 4: Płytka Raspberry Pi 5 i jej schemat pinów GPIO.



Rysunek 5: Moduł zasilający.

Umowżliwia bezpośrednie podpięcie zasilania pod Raspberry Pi (3/4/5) poprzez Pogo piny, nie wykorzystując GPIO.

Moduł posiada również: ochronę przed przeładowaniem, nadmiernym rozładowaniem, zbyt dużym poborem prądu, zwarciem oraz odwrotną polaryzacją

Napięcie wyjściowe: 5 V
Maksymalny prąd wyjściowy: do 5 A
Obsługiwane ogniwa: 2 × akumulatory 18650, 3,7 V (7,4 V nominalnie)

Obliczenia czasu pracy dwóch akumulatorów 18650

Dane:

- Napięcie nominalne akumulatora: $U = 3{,}7\,\mathrm{V}$
- Pojemność jednego akumulatora: $C = 8800\,\mathrm{mAh} = 8{,}8\,\mathrm{Ah}$
- Ilość akumulatorów: 2 (założenie: połączone równolegle lub przez przetwornicę)
- Całkowita pojemność: $C_{\text{total}} = 2 \cdot 8{,}8 = 17{,}6\,\mathrm{Ah}$
- Maksymalny pobór prądu przez urządzenie: $I = 5\,\mathrm{A}$
- Napięcie wyjściowe: $U_{\text{out}} = 5\,\mathrm{V}$
- Sprawność przetwornicy (założenie): $\eta = 0{,}85$ (czyli 85%)

Krok 1: Obliczenie energii dostępnej z akumulatorów:

$$E = U \cdot C_{\text{total}} = 3{,}7\,\mathrm{V} \cdot 17{,}6\,\mathrm{Ah} = 65{,}12\,\mathrm{Wh}$$

Krok 2: Obliczenie mocy pobieranej przez urządzenie:

$$P = U_{\text{out}} \cdot I = 5\,\mathrm{V} \cdot 5\,\mathrm{A} = 25\,\mathrm{W}$$

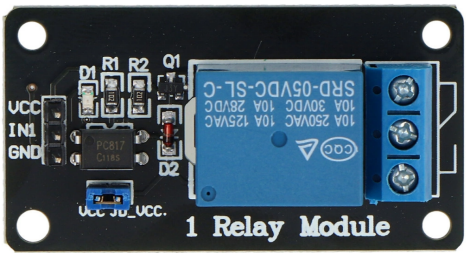
Krok 3: Uwzględnienie sprawności przetwornicy:

$$P_{\text{real}} = \frac{P}{\eta} = \frac{25\,\mathrm{W}}{0{,}85} \approx 29{,}41\,\mathrm{W}$$

Krok 4: Obliczenie czasu pracy:

$$t = \frac{E}{P_{\text{real}}} = \frac{65{,}12\,\mathrm{Wh}}{29{,}41\,\mathrm{W}} \approx 2{,}21\,\mathrm{h}$$

Odpowiedź: Przy maksymalnym poborze prądu 5 A moduł będzie działać około $2{,}2$ godziny.



Rysunek 6: Moduł przekaźnika.

Układ pozwala na sterowanie elementami wykonawczymi przy pomocy portów mikrokontrolera lub dowolnego zestawu uruchomieniowego.

Rysunek 7: Kamerka internetowa.

Monocular vision (widzenie monokularne) to technika pozyskiwania informacji wizualnych przy użyciu tylko jednej kamery, czyli takiej, która rejestruje obraz z pojedynczego punktu widzenia — podobnie jak jedno oko u człowieka. W przypadku obrazu monokularnego, każdy piksel dostarcza jedynie informacji o jasności i kolorze w płaszczyźnie 2D. Brakuje natomiast bezpośredniej informacji o głębokości, czyli odległości od kamery. Z tego względu estymacja głębi z pojedynczego obrazu jest niedookreślonym problemem inwersyjnym – wiele różnych trójwymiarowych scen może prowadzić do identycznej projekcji 2D.

Aby rozwiązać ten problem, konieczne jest wprowadzenie priorytetów – dodatkowych założeń na temat struktury świata, geometrii sceny lub charakterystyki obiektów. Przykładowe priorytety to:

- Zakładanie horyzontalności podłoża i pionowości ścian.
- Regularność obiektów (np. ludzie mają podobną wysokość).
- Perspektywa (linie zbiegające się w punkcie zbiegu sugerują głębię).
- Znajomość statystycznych regularności obrazów.

Z matematycznego punktu widzenia, obraz monokularny powstaje na skutek rzutowania sceny trójwymiarowej na płaszczyznę dwuwymiarową za pomocą rzutowania perspektywicznego. Każdy punkt $P = (X, Y, Z)$ w przestrzeni 3D odwzorowany jest na punkt $p = (x, y)$ w obrazie 2D zgodnie z wzorami:

$$x = f \cdot \frac{X}{Z}, \quad y = f \cdot \frac{Y}{Z}$$

gdzie f to ogniskowa kamery, a Z to głębokość. Zauważmy, że głębokość Z znajduje się w mianowniku, co oznacza, że jej zmiany mają kluczowy wpływ na rozmiar i położenie obiektów na obrazie.

W kontekście wizji monokularnej, głębokie uczenie odgrywa kluczową rolę, umożliwiając estymację głębi, rekonstrukcję scen 3D czy detekcję obiektów na podstawie pojedynczego obrazu. Zamiast polegać na klasycznych metodach geometrycznych, takich jak triangulacja czy analiza ruchu, sieci neuronowe uczą się z dużych zbiorów danych, wychwytując złożone wzorce i zależności przestrzenne. Mimo wysokiej skuteczności, tego typu podejścia mają swoje ograniczenia – wymagają dużej mocy obliczeniowej, są podatne na błędy przy nietypowych danych wejściowych, a ich efektywność jest ściśle związana z jakością i zakresem danych użytych do treningu.

Wady i ograniczenia:

- Brak absolutnej skali – z jednego obrazu nie można jednoznacznie wywnioskować rzeczywistej odległości.
- Trudności w teksturowo jednorodnych obszarach – gdzie brak cech uniemożliwia dobre przewidywanie.
- Problemy z generalizacją – modele trenowane na jednej dziedzinie mogą słabo działać na innych.
- Dynamiczne sceny i obiekty poruszające się niezależnie od kamery – zaburzają proces estymacji.

Rysunek 8: Kamera stereowizyjna

Stereowizja (lub Wizja stereoskopowa) to technika polegająca na wykorzystaniu dwóch (lub więcej) obrazów tej samej sceny, uchwyconych z nieco innych punktów widzenia, do wyod-

rębnienia informacji przestrzennych. Jest to jedno z najstarszych i najintensywniej badanych podejść do estymacji głębi, inspirowane sposobem, w jaki ludzkie oczy – jako dwa przesunięte względem siebie punkty obserwacyjne – postrzegają świat trójwymiarowy.

W odróżnieniu od wizji monokularnej, stereowizja oferuje geometrycznie uzasadnioną możliwość bezpośredniego obliczenia głębokości, co czyni ją bardzo atrakcyjną w aplikacjach wymagających wysokiej dokładności. W tym rozdziale przedstawiono podstawy matematyczne wizji stereo, klasyczne i współczesne metody obliczania głębi, a także omówiono praktyczne zastosowania i ograniczenia tej technologii.

W systemie stereowizyjnym wykorzystuje się dwa obrazy uchwycone przez kamery umieszczone w znanej odległości od siebie (baza stereo). Podstawowym pojęciem jest paralaksa – przesunięcie obrazu tego samego punktu sceny pomiędzy obrazami lewego i prawego oka/kamery.

Zakładając idealną konfigurację (kamery wyrównane, płaszczyzny obrazu równoległe), głębokość Z danego punktu sceny można obliczyć ze wzoru:

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

- f - ogniskowa kamery.
- B - odległość między kamerami.
- d - przesunięcie danego punktu w obrazie lewym względem prawego.

Im większe d, tym mniejsza głębokość – obiekty bliżej kamery mają większe przesunięcie między obrazami.

Wady i ograniczenia:

- Wymóg kalibracji i synchronizacji kamer – błędy w tym zakresie przekładają się bezpośrednio na błędną głębokość.
- Brak dopasowania w teksturowo ubogich obszarach – np. białe ściany, niebo.
- Problemy przy silnym oświetleniu i odbiciach – zmienność intensywności zaburza dopasowanie.
- Duży koszt obliczeniowy – szczególnie w przypadku metod globalnych lub opartych na deep learningu.
- Widzenie tylko z jednej perspektywy – martwe strefy między kamerami lub poza polem widzenia jednej z nich.

Rozdział 4

Stereowizja

Stereowizja umożliwia rozpoznawanie głębi na obrazie, wykonywanie pomiarów na obrazie i przeprowadzanie lokalizacji 3D. Między innymi należy znaleźć punkty, które pasują do siebie między dwiema kamerami. Można to następnie wykorzystać do odległość między kamerą a punktem. Wykorzystywana jest geometria systemu w celu uproszczenia obliczeń.

Te cztery kroki są wykonywane podczas obrazowania stereo:

1. usuwanie zniekształceń promieniowych i stycznych za pomocą obliczeń matematycznych obliczenia. W ten sposób powstają obrazy bez deformacji.

2. rektyfikacja kąta i odległości obrazów. Na tym etapie oba obrazy są obrazy współpłaszczyznowe na osi Y, co ułatwia wyszukiwanie korespondencji. łatwiejsze i wystarczy szukać tylko na jednej osi (osi X).

3. znajdź tę samą cechę na prawym i lewym obrazie. Daje to mapę dysproporcji pokazującą różnice między obrazami na osi X.

4. Ostatnim krokiem jest triangulacja. Mapa dysparycji jest przekształcana w odległości za pomocą triangulacji.

Program jest zaprogramowany w Pythonie i wykorzystywana jest biblioteka OpenCV do przetwarzania obrazu oraz rpi-lgpio do kontroli pinów GPIO. W tym programie wystepuję kalibracja kamery za pomocą wykonanych zdjęć, generacja mapy dysproporcji i dzięki doświadczalnemu równaniu równania, które zostało znalezione eksperymentalnie, można zmierzyć odległość dla każdego piksela. Na końcu używany jest filtr WLS, aby łatwiej rozpoznawać krawędzie obiektów.

Program jest uruchaminy razem z uruchomieniem systemu, moduł przekaźnika jest domyśl-

nie wyłączony, obie kamery stają się aktywne.

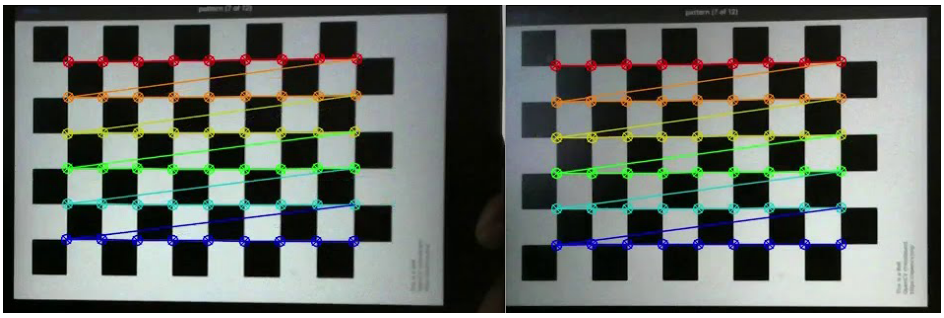
4.1. Kalibracja

```
# Okresla warunki, przy ktorych iteracyjny algorytm sie zatrzymuje
# Przerwij iteracje, gdy wykonano 30 krokow lub zmiana wyniku jest mniejsza niz 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 punktow obiektu
objp = np.zeros((9*6,3), np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)

# Tablice do przechowywania punktow obiektu i punktow obrazu ze wszystkich obrazow
objpoints= []   # Punkty 3D w przestrzeni swiata rzeczywistego
imgpointsR= []   # Punkty 2D na plaszczyznie 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])
```

Biblioteka OpenCV pozwala nam obliczyć parametry wewnętrzne za pomocą określonych funkcji. Proces ten nazywany jest kalibracją. Jest to możliwe dzięki zdjęciom szachownicy wykonanym pod różnymi kątami.

Odpowiednimi obrazami są te, na których narożniki są bardzo wyraźnie rozpoznawalne za pomocą funkcji cv2.findChesssboardCorners(). OpenCV zaleca posiadanie co najmniej 10 zdjęcia dla każdej kamery, aby uzyskać dobrą kalibrację. Do projektu zostało pobrane 32

zdjęcia dla każdej kamery.



Rysunek 9: Widok wykrytych wierzchołków szachownic.

Funkcja cv2.findChessboardCorners() wyszuka określoną liczbę narożników szachownicy i wygenerowane zostaną następujące wektory:

- 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.

Precyzja współrzędnych znalezionych narożników jest zwiększana za pomocą funkcji cv2.cornerSubPix().

Funkcja cv2.getOptimalNewCameraMatrix() umożliwia uzyskanie dokładnych macierzy kamer, które później będą wykorzystane w rektyfikacji.

Do korekcji dystorsji kamer używane są zarejestrowane obrazy, gdzie w zmiennych imgpoints i objpoints zapisywane są pozycje narożników szachownicy.

Funkcja cv2.calibrateCamera() jest wykorzystywana do uzyskania nowych macierzy kamer (macierz kamery opisuje projekcję punktu ze świata 3D na obraz 2D), współczynników dystorsji, wektorów rotacji oraz translacji dla każdej z kamer. Dane te są później używane do usunięcia zniekształceń dla każdej kamery.

Macierz wewnętrzna kamery $\mathbf{M}$, przekształca współrzędne punktu w układzie współrzędnych kamery 3D do układu współrzędnych obrazu 2D. Dla każdej kamery (np. lewej i prawej w systemie stereo), macierz ta ma postać:

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

- $f_x = f \cdot s_x$ — ogniskowa wyrażona w pikselach w poziomie (oś X)
- $f_y = f \cdot s_y$ — ogniskowa wyrażona w pikselach w pionie (oś 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

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

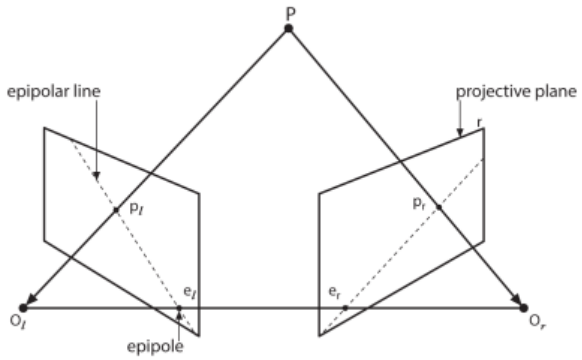
Aby uzyskać optymalne macierze kamer dla każdej z nich, używana jest funkcja cv2.getOptimalNewCameraMatrix() (w celu zwiększenia precyzji), które są wykorzystane w funkcji cv2.stereoRectify().

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

Do kalibracji stereo używana jest funkcja cv2.StereoCalibrate(), która oblicza transformację pomiędzy dwiema kamerami (jedna kamera służy jako odniesienie dla drugiej).

4.2. Rektyfikacja stereo

4.2.1. Geometria epipolarna



Rysunek 10: Model kamery stereo.

Powyższy obrazek przedstawia model niedoskonałej kamery stereo składającej się z dwóch modeli kamer otworkowych. Przecięcie linii środków projekcji (O_l, O_r) z płaszczyznami projekcji tworzone są punkty epipolarne e_l i e_r. Linie (p_l, e_l) i (p_r, e_r) są nazywane liniami epipolarnymi. Obrazem wszystkich możliwych punktów punktu na płaszczyźnie rzutowania jest linia epipolarna, która leży na drugiej płaszczyźnie obrazu i przechodzi przez punkt epipolarny i szukany punkt. Umożliwia to ograniczenie wyszukiwania punktu na jednym wymiarze zamiast na całej płaszczyźnie.
Można zatem podsumować następujące punkty:

- Każdy punkt 3D w widoku kamery jest zawarty w planie epipolarnym
- Element w jednej płaszczyźnie musi znajdować się na odpowiednich liniach epipolarnych drugiej płaszczyzny (warunek epipolarny). drugiej płaszczyzny (warunek epipolarny)
- Dwuwymiarowe wyszukiwanie odpowiadającego elementu jest konwertowane na jednowymiarowe, jeśli znana jest geometria epipolarna.
- Kolejność punktów jest zachowana, tzn. dwa punkty A i B są w tej samej kolejności na liniach epipolarnych płaszczyzny.

ta sama kolejność na liniach epipolarnych jednej płaszczyzny, co na liniach epipolarnych drugiej płaszczyzny.

4.2.2. Macierze podstawowe i fundamentalne

Aby zrozumieć, w jaki sposób obliczane są linie epipolarne, musimy najpierw wyjaśnić macierze podstawowe i macierze fundamentalne (odpowiadające macierzom E i F). Macierz podstawowa E zawiera informacje o tym, jak fizycznie rozmieszczone są obie kamery. Opisuje ona lokalizację drugiej kamery względem pierwszej za pomocą parametrów translacji i rotacji. Parametrów tych nie można odczytać bezpośrednio w macierzy, ponieważ jest ona używana do planowania projektu. W sekcji Kalibracja stereo wyjaśnone będzie, jak obliczyć R i T (macierz rotacji i wektor translacji). Macierz F zawiera informacje z podstawowej macierzy E, fizyczny układ kamer i informacje o wewnętrznych parametrach kamer. Relacja między rzutowanym punktem na lewym obrazie p_l i rzutowanym punktem na prawym obrazie p_r jest zdefiniowana następująco:

$$prTEpl = 0$$

Można by pomyśleć, że ta formuła w pełni opisuje związek między lewym i prawym punktem. prawym punktem. Należy jednak zauważyć, że macierz 3x3 E jest rzędu jest rangi 2. Oznacza to, że wzór ten jest równaniem prostej. Aby w pełni zdefiniować relację między punktami, parametry wewnętrzne. parametry wewnętrzne. Pamiętamy, że $q = Mp$, z macierzą wewnętrzną M. Podstawienie do poprzedniego równania daje wynik:

$$qrT(Ml-1)TEMl-1ql = 0$$

Podstawienie:

$$F = (Ml-1)TEMl-1$$

W ten sposób otrzymujemy

$$qrTFql = 0$$

4.2.3. Macierz obrotu i wektor przesunięcia

Teraz, gdy zostałą wyjaśniona macierz podstawową E i macierz podstawową F, trzeba zobaczyć, jak obliczyć macierz obrotu i wektor translacji. Zdefiniujemy następujące oznaczenia:

- P_l i P_r definiują pozycje punktu w układzie współrzędnych odpowiednio lewej i prawej kamery.

- R_l i T_l (lub R_r i T_r) definiują obrót i translację z kamery do punktu w otoczeniu dla lewej (lub prawej) kamery.
- R i T to obrót i translacja układu współrzędnych prawej kamery w układzie współrzędnych lewej kamery.

Daje to następujące wyniki

$$P_l = R_l P + T_l \qquad P_r = R_r P + T_r$$

Mamy również:

$$Pl = RT(Pr - T)$$

Z tych trzech równań ostateczny wynik to

$$R = R_r R_l^T \qquad T = T_r - RT_l$$

Zadaniem rektyfikacji obrazów stereo jest przekształcenie obu obrazów w taki sposób, aby leżały w tej samej płaszczyźnie obrazowania oraz by linie epipolarne stały się wzajemnie równoległe i poziome. Dzięki temu, dla każdego punktu na jednym obrazie, jego odpowiednik na drugim obrazie znajduje się na tej samej linii poziomej, co znacznie upraszcza proces wyszukiwania korespondencji punktów.

W wyniku procesu rektyfikacji uzyskuje się osiem parametrów — po cztery dla każdej kamery:

- wektor zniekształceń
- macierz rotacji rektyfikującej R_{rect}
- wyprostowana (zrektyfikowana) macierz M_{rect}
- nierektyfikowana macierz kamery M

OpenCV pozwala nam obliczyć te warunki za pomocą dwóch algorytmów: algorytmu Hartley'a i algorytmu Bouguet'a.

4.2.4. Algorytm Hartley'a

Algorytm Hartley'a [8] jest metodą rektyfikacji niekalibrowanej, czyli nie wymaga znajomości parametrów wewnętrznych kamer (np. ogniskowej, współrzędnych środka obrazu, współczynników zniekształceń soczewek). Jest to istotne w zastosowaniach, gdzie dostępne są tylko obrazy, bez wcześniejszej kalibracji sprzętu.

Kluczowe kroki:

- Najpierw algorytm odnajduje zestawy punktów, które występują na obu obrazach — tzw. korespondencje punktowe. W praktyce mogą to być np. punkty narożne, cechy SIFT, SURF czy ORB.
- Z punktów odpowiadających estymowana jest macierz fundamentalna, która opisuje geometryczne powiązania pomiędzy dwoma obrazami.
- Za pomocą macierzy F obliczane są transformacje projekcyjne (homografie) dla obu obrazów, które przekształcają je w taki sposób, aby epipole zostały przekształcone w nieskończoność, co oznacza, że linie epipolowe stają się poziome i równoległe.

  Dzięki temu poszukiwanie dopasowań między obrazami można sprowadzić do jednej osi (najczęściej poziomej), co znacząco upraszcza dalszą analizę.

Główną wadą tej metody jest jednak to, że nie dostarcza informacji o rzeczywistej skali sceny. Oznacza to, że uzyskane dane dotyczą jedynie względnego położenia obiektów, bez możliwości dokładnego określenia ich odległości od kamery. W rezultacie, mimo że algorytm umożliwia uzyskanie rektyfikowanych obrazów, jego precyzja jest niższa w porównaniu do metod opartych na pełnej kalibracji kamer.

4.2.5. Algorytm Bouguet'a

Algorytm Bouguet'a [9] (opracowany m.in. w ramach narzędzia Camera Calibration Toolbox do MATLABa) to metoda rektyfikacji kalibrowanej, czyli wykorzystująca uprzednio wyznaczone parametry wewnętrzne i zewnętrzne kamer.

Kluczowe kroki:

- Na podstawie obrazów wzorca (np. szachownicy) określane są parametry wewnętrzne każdej kamery (ogniskowa, środek obrazu, zniekształcenia) oraz parametry zewnętrzne (rotacja i translacja między kamerami).
- Obrazy są odwzorowywane do wspólnej przestrzeni trójwymiarowej, w której określona jest pozycja i orientacja obu kamer.
- Płaszczyzny obrazów są obracane w taki sposób, by ich główne osie optyczne stały się równoległe (tzn. skierowane w tym samym kierunku), a obrazy znajdowały się w tej samej płaszczyźnie. Dzięki temu linie epipolowe stają się poziome i równoległe — jak w przypadku metody Hartley'a, ale z większą precyzją.

- Na podstawie wyliczonych rzutów generowane są nowe obrazy zrektyfikowane, w których każdy piksel z jednej kamery można bezpośrednio porównać z pikselem z drugiej kamery w tej samej linii poziomej.

Metoda ta jest wrażliwa na błędy kalibracji — na przykład niedokładne wykrycie wzorca kalibracyjnego lub jego niewłaściwe ujęcie na zdjęciach może znacząco pogorszyć jakość rektyfikacji, prowadząc do błędów w dalszym przetwarzaniu obrazów.

```
# Funkcja stereo rektyfikacji
RL, RR, PL, PR, Q, roiL, roiR= cv2.stereoRectify(MLS, dLS, MRS, dRS,
                                                 ChessImgR.shape[::-1], R, T, 0,(0,0))
# ostatni parametr to alfa, jesli 0= przyciety, jesli 1= nie przyciety

# Funkcja przygotowania map przeksztalcen
Left_Stereo_Map= cv2.initUndistortRectifyMap(MLS, dLS, RL, PL, ChessImgR.shape[::-1], cv2.CV_16SC2)
# cv2.CV_16SC2 ten format umozliwia nam szybsza prace programu
Right_Stereo_Map= cv2.initUndistortRectifyMap(MRS, dRS, RR, PR, ChessImgR.shape[::-1], cv2.CV_16SC2)
```

Funkcja cv2.stereoRectify() umożliwia sprowadzenie linii epipolarnych obu kamer do tej samej płaszczyzny. Taka transformacja ułatwia działanie funkcji generującej mapę dysparycji, ponieważ dopasowywanie bloków musi być wtedy wykonywane tylko w jednym wymiarze.

Za pomocą tej funkcji uzyskuje się również macierz esencjalną oraz macierz fundamentalną, które są wykorzystywane w kolejnej funkcji.

$$\mathbf{M}_L = \begin{bmatrix} 791,0 & 0 & 390,6 \\ 0 & 791,0 & 194,4 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{M}_R = \begin{bmatrix} 791,0 & 0 & 390,6 \\ 0 & 791,0 & 194,4 \\ 0 & 0 & 1 \end{bmatrix}$$

Funkcja cv2.initUndistortRectifyMap() generuje obraz bez zniekształceń (dystorsji). Obrazy te są następnie wykorzystywane przy obliczaniu mapy dysparycji.

4.3. Mapa dysparycji

```
# 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 druga instancje StereoSGBM, identyczna jak stereo,
# ale z ustawieniami odpowiednimi do przetwarzania prawego obrazu wzgledem lewego.
stereoR=cv2.ximgproc.createRightMatcher(stereo)
```

Mapa dysparycji[2] 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.

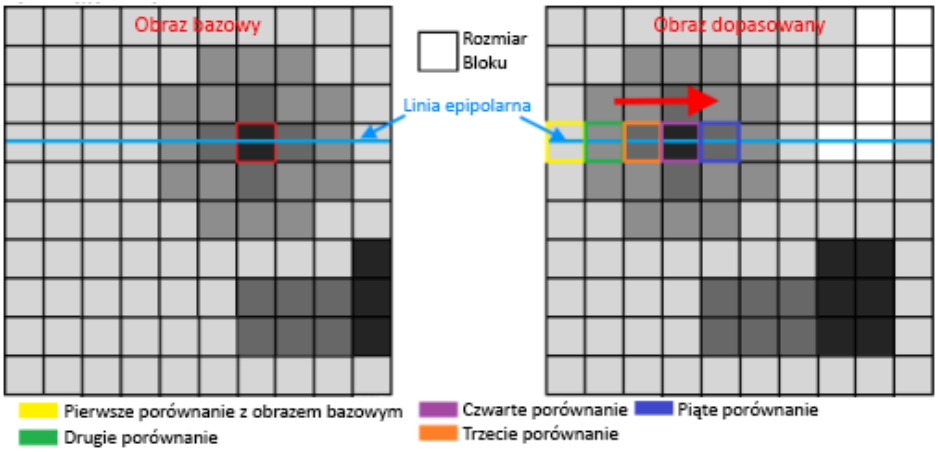
Aby obliczyć mapę dysparycji, wykorzystywany jest obiekt klasy StereoSGBM, tworzony za pomocą funkcji cv2.StereoSGBM-create(). Algorytm ten implementuje metodę półglobalnego dopasowania blokowego (Semi-Global Block Matching) [7], która umożliwia estymację przesunięć pomiędzy obrazami stereo — z lewej 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 te-

mu przeszukiwanie ogranicza się do jednej wymiarowej przestrzeni (poziomej), co znacznie zmniejsza złożoność obliczeniową algorytmu.

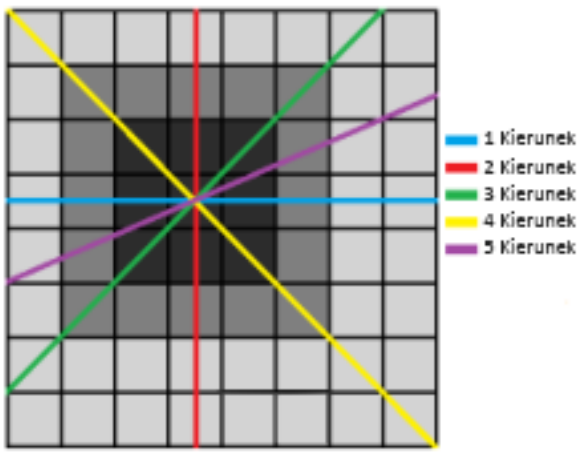
Na przykład, blok o współrzędnych (4,7) w obrazie bazowym zostaje porównany z wszystkimi blokami (4, i) w tej samej linii epipolarnej obrazu dopasowywanego.



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.



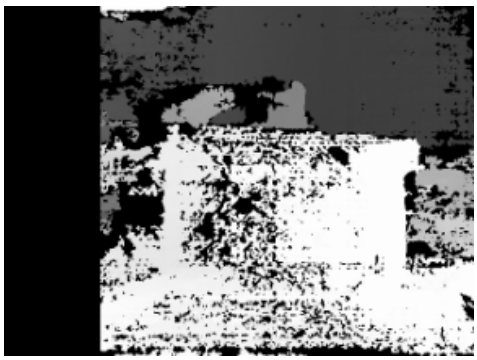
Rysunek 12: Pięć kierunków przeszukiwania w algorytmie StereoSGBM.

Wartość dysparycji 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 zwiększonym obciążeniem procesora, bez gwarancji proporcjonalnej poprawy wyników.

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

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

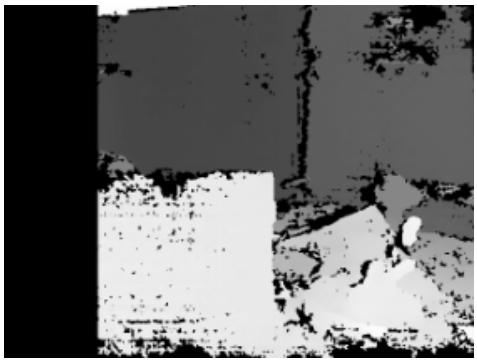


Rysunek 13: Wynik dla mapy dysproporcji.

Na wygenerowanej mapie dysparycji 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 filtru pozwala na eliminację drobnych czarnych artefaktów, które mogą pojawić się wewnątrz jednorodnych obszarów na mapie.

Rysunek 14: Przykład filtra zamykającego.

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



Rysunek 15: Wynik mapy dysproporcji po filtrze zamykającym.

4.4. Filtr WLS

```
# WLS FILTER Parameters
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 WLS (Weighted Least Squares) — filtr ważonych najmniejszych kwadratów, 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 prawego 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 filtru WLS nie jest już bezpośrednio wykorzystywalną mapą dysparycji — jest to obraz zakodowany w formacie uint8, który dobrze uwidacznia krawędzie, lecz nie zawiera już rzeczywistych wartości głębokości (w przeciwieństwie do oryginalnej mapy dysparycji w formacie float32).

4.5. 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()

    # Rownolegle 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()

    # Rownolegla konwersja na skale szarosci
    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()

    # Rownolegla kalkukacja dysproporcji
```

```
    future_dispL = executor.submit(stereo.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 odległością. Aby to zrobić, eksperymentalnie zmierzyliśmy wartości dysparycji w kilku miejscach, aby na tej podstawie określić regresję.

```
_, 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 dysparycji obliczany jest bounding box wokół konturu, następnie wycinany jest fragment mapy dysparycji w tym obszarze. Z jego środka (cx, cy) pobierany jest mały fragment (3×3) do analizy.

Obliczana jest średnia dysparycja. Z 3×3 pikseli ze środka wybierane są tylko te z sensowną wartością (większą od zera).

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

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

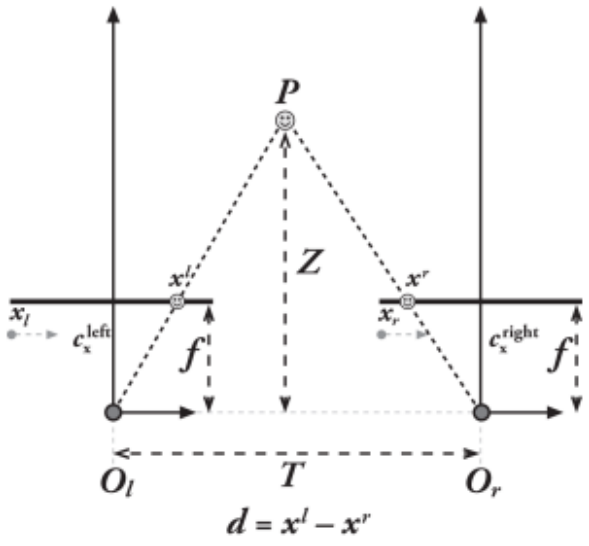
Jeśli obiekt znajduje się bliżej niż 1 metr, rysowana jest prostokątna ramka: czerwona jeśli < 0.5 m, zielona jeśli 0.5 – 1.0 m.

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 były w stanie zmierzyć odległość do obiektu z precyzją ± 5 cm.

Zwrócona wartość to średnia dysparycji z macierzy 9x9 pikseli.

4.6. Triangulacja



Rysunek 16: Schemat triangulacji.

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

Poniższy obraz można teraz skonstruować przy użyciu poprzednich hipotez.

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:

$$Z = f * T/(xl - xr)$$

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.

Metoda ta działa jednak tylko wtedy, gdy konfiguracja kamery stereo jest idealna. W rzeczywistości tak nie jest. Właśnie dlatego lewy i prawy obraz są matematycznie wyrównane równolegle. Oczywiście kamery muszą być fizycznie ustawione równolegle. Zanim zostanie wyjaśniona metoda matematycznego wyrównywania obrazów, trzeba najpierw zrozumieć geometrię epipolarną.

4.7. Wnioski i możliwe ulepszenia

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.

Wystarczyłoby zapisać wartości macierzy z kalibracji stereo, aby móc je ponownie wykorzystać. Dużo czasu mogłoby to zaoszczędzić w trakcie inicjalizacji.

Uruchomienie programu na GPU pozwoliłoby również uzyskać gładsze obrazy, gdy kamera stereo jest w ruchu.

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

Structured Light

Rysunek 17: Kamera Xbox 360 Kinect

Structured Light (pol. światło strukturalne) to technika aktywnej wizji komputerowej wykorzystywana do precyzyjnego pomiaru kształtu i głębokości obiektów. Polega na projekcji znanego wzorca świetlnego (np. siatki, kropek, pasków) na powierzchnię sceny, a następnie analizie deformacji tego wzorca za pomocą kamery.

System Structured Light składa się zazwyczaj z dwóch komponentów:

- Projektora – emituje wzorzec świetlny (np. siatkę punktów lub paski) na obserwowaną scenę.
- Kamery – rejestruje zniekształcony wzorzec po odbiciu od obiektów w przestrzeni.

Proces działa następująco:

1. Znany wzorzec zostaje wyświetlony na scenie.
2. Gdy wzorzec napotyka obiekty o różnych kształtach i odległościach, zostaje geometrycznie zniekształcony.

3. Kamera rejestruje te deformacje.

4. System porównuje zarejestrowany obraz wzorca ze wzorcem referencyjnym, który byłby widoczny na płaskiej powierzchni.

5. Na podstawie różnic (tzw. disparity) obliczana jest głębokość – za pomocą triangulacji.

Wady i ograniczenia:

- W jasnym świetle dziennym (szczególnie na zewnątrz), wzorzec świetlny może zostać zaburzony lub całkowicie zaniknąć – szczególnie jeśli działa w paśmie IR.
- Technika najlepiej sprawdza się na krótkich dystansach (0,5–2 m). Dalsze obiekty dają mniej wyraźne zniekształcenia wzorca.
- Szkło, lustra, woda lub powierzchnie metaliczne mogą zaburzyć wzorzec lub wprowadzać wielokrotne odbicia.
- Projektor i kamera muszą być precyzyjnie skalibrowane względem siebie – błędy kalibracji mogą znacząco wpłynąć na jakość głębi.
- Gdy wzorzec nie dotrze do części sceny (np. w załomach, pod kątem), pomiar głębokości w tych miejscach będzie niemożliwy.

LIDAR (Light Detection and Ranging)

Rysunek 18: Kamera Intel RealSense z technologią LIDAR.

LIDAR (Light Detection and Ranging) to technologia zdalnego pomiaru odległości, która działa poprzez wysyłanie impulsów laserowych i mierzenie czasu, jaki upływa od ich odbicia od obiektu do powrotu do sensora. Na tej podstawie LIDAR tworzy bardzo dokładne mapy 3D otoczenia.

Podstawowy mechanizm działania LiDAR opiera się na bardzo prostej zasadzie:

- Sensor emituje impuls laserowy w kierunku otoczenia.

- Światło odbija się od powierzchni obiektów i wraca do detektora.
- System mierzy czas, jaki upłynął od wysłania do odebrania sygnału (Time-of-Flight, ToF).
- Znając prędkość światła, obliczana jest dokładna odległość:

$$d = \frac{c \cdot \Delta t}{2}$$

gdzie:

$$d - \text{odległość do obiektu}$$
$$c - \text{prędkość światła (ok. } 3 \cdot 10^8 \text{ m/s)}$$
$$\Delta t - \text{czas przelotu sygnału}$$

LiDAR-y mogą wykonywać takie pomiary miliony razy na sekundę, skanując środowisko w 2D (jeden plan) lub 3D (pełna chmura punktów).

Wady i ograniczenia:

- Mgła, deszcz, śnieg i kurz mogą zakłócać odbicie promieni lasera, co wpływa na dokładność pomiarów.
- Brak dopasowania w teksturowo ubogich obszarach – np. białe ściany, niebo.
- LiDAR rejestruje wyłącznie dane geometryczne – nie dostarcza żadnych informacji o kolorze czy teksturze powierzchni.
- W porównaniu do kamer, LiDAR-y mają stosunkowo rzadką siatkę pomiarową, co skutkuje niższą rozdzielczością przy dużych odległościach (np. obiekt 100 m dalej może być opisany przez kilka punktów).
- Bardzo ciemne lub przezroczyste powierzchnie (np. szyby) mogą słabo odbijać impulsy lasera lub w ogóle je przepuszczać.

Kamery zdarzeniowe

Kamery zdarzeniowe (ang. Event Cameras) to innowacyjne sensory wizyjne, które różnią się fundamentalnie od tradycyjnych kamer opartych na matrycy CMOS. Zamiast przechwytywać obraz w sposób klatkowy (frame-based), rejestrują one zmiany jasności na poziomie pojedynczych pikseli, co pozwala na znacznie wyższą rozdzielczość czasową i lepszą reakcję na dynamiczne sceny. Dzięki temu technologia ta znajduje coraz szersze zastosowanie w systemach robotycznych, autonomicznych pojazdach, AR/VR i przetwarzaniu sygnałów w czasie rzeczywistym.

W tradycyjnych kamerach każda klatka rejestrowana jest w określonym interwale czasowym, co powoduje powstawanie rozmycia ruchu i dużego opóźnienia w dynamicznych scenach. Kamery zdarzeniowe działają zupełnie inaczej:

Każdy piksel działa niezależnie i stale monitoruje zmiany lokalnej jasności.

Gdy zmiana przekroczy ustalony próg (np. 10

Zdarzenie zawiera informację o:

- położeniu piksela (x, y),
- czasie zdarzenia (z dokładnością do mikrosekund),
- polaryzacji zmiany (jasność wzrosła lub zmalała).

Dzięki temu kamera generuje strumień asynchronicznych zdarzeń, a nie szereg klatek. Przykładami takich kamer są m.in. DVS (Dynamic Vision Sensor), DAVIS (łączący klasyczną kamerę z kamerą zdarzeniową) oraz CeleX.

Brak informacji o statycznych obiektach: jeśli scena się nie zmienia, kamera nie generuje zdarzeń – co utrudnia pełną rekonstrukcję otoczenia.

Trudności w przetwarzaniu danych: strumień zdarzeń ma inną strukturę niż klasyczne obrazy – wymaga specjalnych algorytmów i często dedykowanego sprzętu (np. FPGA).

Niska rozdzielczość przestrzenna: w porównaniu do tradycyjnych kamer, choć technologia ta dynamicznie się rozwija.

Szum przy słabym oświetleniu: niektóre sensory są bardziej podatne na fałszywe zdarzenia w nocy lub w ciemnych pomieszczeniach.

Koszt: kamery zdarzeniowe są wciąż relatywnie drogie i mniej dostępne komercyjnie.

Wady i ograniczenia:

- Wymóg kalibracji i synchronizacji kamer – błędy w tym zakresie przekładają się bezpośrednio na błędną głębokość.
- Brak dopasowania w teksturowo ubogich obszarach – np. białe ściany, niebo.
- Problemy przy silnym oświetleniu i odbiciach – zmienność intensywności zaburza dopasowanie.
- Duży koszt obliczeniowy – szczególnie w przypadku metod globalnych lub opartych na deep learningu.
- Widzenie tylko z jednej perspektywy – martwe strefy między kamerami lub poza polem widzenia jednej z nich.

Rozdział 5

Monowizja

Rozdział 6

Zakończenie

Spis rysunków

Bibliografia

[1] John Lambert, Stereo and Disparity, *https://johnwlambert.github.io/stereo/*.

[2] Baeldung authors, Disparity Map in Stereo Vision, *https://www.baeldung.com/cs/disparity-map-stereo-vision*, 2025-03-26.

[3] Rajesh Rao, Stereo and 3D Vision, *https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf*.

[4] Adrian Kaehler, Gary Bradski, Learning OpenCV 3, O'Reilly, 2017-11.

[5] Dystorsja obrazu w fotografii, *https://beafoto.pl/dystorsja*, 2021-01.

[6] Making A Low-Cost Stereo Camera Using OpenCV, *https://learnopencv.com/making_a_low_cost_stereo_camera_using_opencv/*, 2021-01-11.

[7] Heiko Hirschmüller *https://en.wikipedia.org/wiki/Semi_global_matching*, 2005.

[8] Ralph Hartley *https://en.wikipedia.org/wiki/Hartley_function*, 1928.

[9] Jean-Yves Bouguet *https://robots.stanford.edu/cs223b04/JeanYvesCalib/*.