Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA AUTOMATYKI

KIERUNEK AUTOMATYKA I ROBOTYKA

PRACA MAGISTERSKA

MATEUSZ PANUŚ

STEREOWIZJA Z WYKORZYSTANIEM WIELU KAMER

PROMOTOR:
dr inż. Paweł Rotter

Kraków 2012

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

.....

PODPIS

AGH
University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF AUTOMATICS

DISCIPLINE AUTOMATICS AND ROBOTICS

MASTER OF SCIENCE THESIS

MATEUSZ PANUŚ

MULTI-CAMERA STEREOVISION

SUPERVISOR:
Paweł Rotter Ph.D

Krakow 2012

Serdecznie dziękuję promotorowi za poświęcony czas i wskazówki pomocne przy realizacji pracy.

Spis treści

1. Stereoskopowa rekonstrukcja sceny trójwymiarowej

1.1. Kilka słów o stereowizji

Stereowizja jest procesem obrazowania, umożliwiającym określenie położenia w przestrzeni obserwowanych obiektów. Natura w procesie ewolucji wyposażyła nas w zdolność widzenia stereoskopowego. Ta unikalna cecha pozwala na precyzyjną ocenę odległości oraz prędkości zbliżania i oddalania się obiektów, pomagając w zdobywaniu pożywienia i unikaniu zagrożeń. Do uzyskania zdolności widzenia przestrzennego niezbędna jest możliwość obserwacji tej samej sceny z różnych perspektyw. Drobne różnice w zarejestrowanych obrazach są przetwarzane przez nasz mózg, dzięki czemu uzyskujemy wrażenie przestrzennego postrzegania otaczającego nas świata.

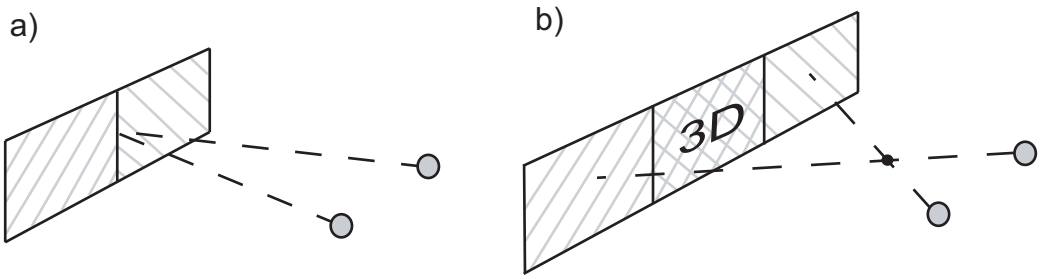
Możliwości ludzkiego mózgu w zakresie widzenia przestrzennego są zdumiewające i stanowią niedościgniony wzór dla każdego cyfrowego systemu stereowizyjnego. Jednocześnie przestrzenne postrzeganie przedmiotów jest dla nas czynnością tak naturalną, że trudno dostrzec, jak niezwykły jest to proces. Każdy jednak może to sprawdzić wykonując prosty test. Rysunek 1.1 przedstawia dwa zdjęcia tego samego miejsca, wykonane z niewielkim przesunięciem poziomym (tzw. stereoparę). Patrząc normalnie nie dostrzegamy w tych zdjęciach niczego niezwykłego (rysunek 1.2 a). Jeśli natomiast skupimy wzrok w punkcie pomiędzy kartką a naszą twarzą, w taki sposób aby oba zdjęcia nakryły się na siebie, uzyskamy bardzo ciekawy efekt (rysunek 1.2 b). Przez pewien czas obraz może być nieostry i rozmazany, ale po dłuższej chwili wpatrywania się w niego, uzyskamy perfekcyjnie ostry, trójwymiarowy obraz zarejestrowanej sceny.

Ogromne możliwości jakie daje zdolność określania położenia obiektów w przestrzeni sprawiają, że od wielu lat podejmowane są próby stworzenia cyfrowego systemu stereowizyjnego. Potencjalne zastosowania takiego systemu obejmują takie obszary jak: robotyka (sterowanie pojazdami, robotami przemysłowymi), kartografia, systemy nawigacyjne, rozpoznawanie i rekonstrukcja obiektów 3D czy wspomaganie osób niepełnosprawnych [2]. Ogólny schemat takiego systemu przedstawiony jest na rysunku 1.3. Efektem działania algorytmu jest tzw. mapa głębi, czyli obraz zawierający informację o odległości od kamery dla wszystkich zarejestrowanych punktów. Mapa głębi może być przedstawiana w różny sposób, na przykład za pomocą skali odcieni szarości jak to przedstawiono na rysunku 1.3 (jaśniejszy kolor oznacza mniejszą odległość od kamery). Stworzono wiele algorytmów umożliwiających wyznaczenie mapy głębi, jednak istnieje szereg problemów ograniczających wykorzystanie systemów stereowizyjnych w powszechnym użyciu. Najważniejsze z nich to: jakość generowanych map głębi, odporność na zmienne warunki oświetlenia oraz duża złożoność obliczeniowa (długi czas obliczeń).

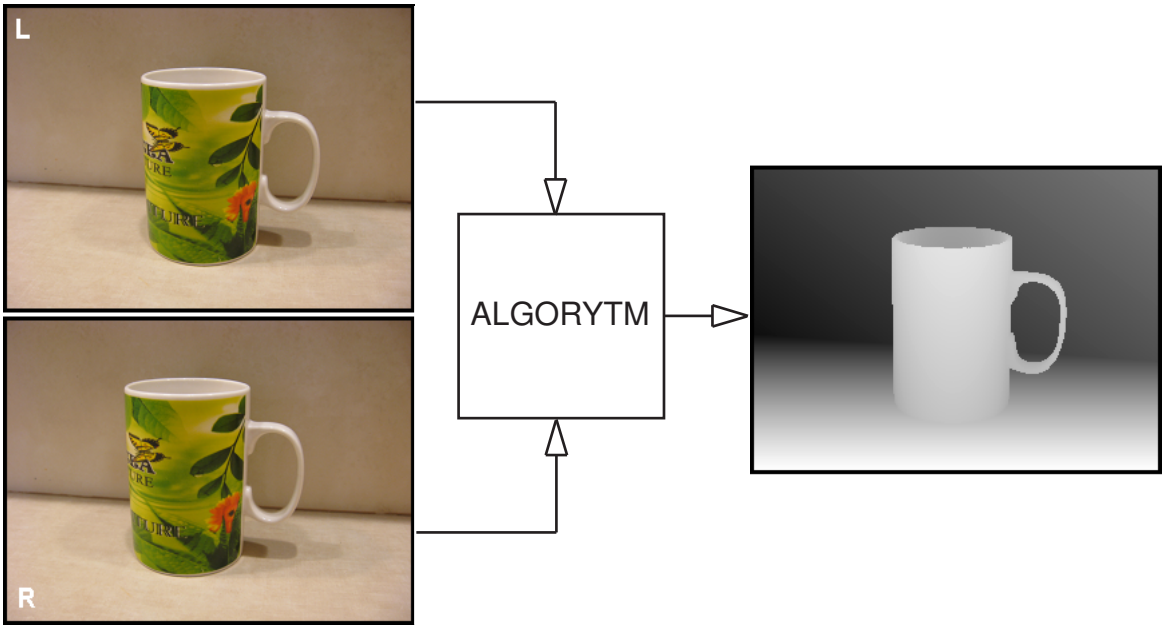
Niniejsza praca poświęcona jest analizie poprawy jakości systemu stereowizyjnego przy wykorzystaniu więcej niż dwóch kamer, dostarczających dodatkowych informacji o obserwowanej scenie. Zwiększenie ilości kamer spowoduje nieuchronny wzrost czasu potrzebnego na dokonanie niezbędnych ob-

liczeń, dlatego końcowy rozdział pracy poświęcony jest dostępnym metodom przyspieszenia działania programu z użyciem obliczeń wykonywanych równolegle. Pierwsze rozdziały poświęcone są przybliżeniu teorii oraz podstawowych zagadnień dotyczących klasycznej stereowizji z wykorzystaniem dwóch kamer.

Rysunek 1.1: Stereopara



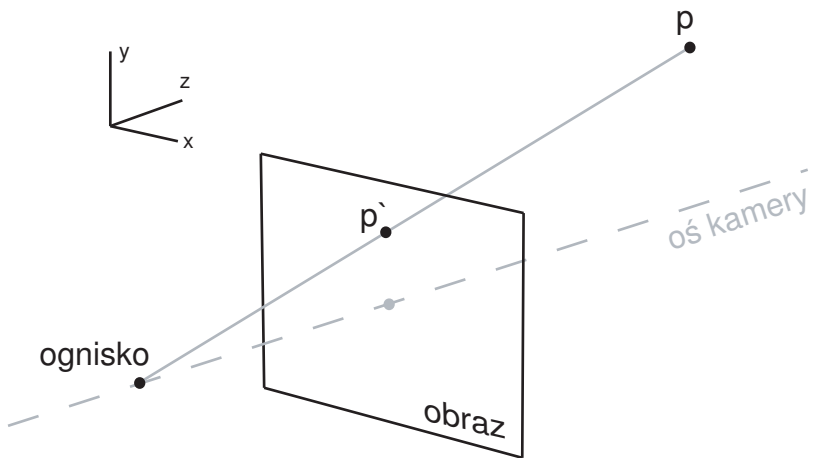
Rysunek 1.2: Metoda uzyskania wrażenia 3D



Rysunek 1.3: Podstawowa idea wykorzystania stereowizji w systemach cyfrowych

1.2. Podstawowe parametry kamery

Każde urządzenie zdolne do rejestrowania obrazu (nazywane w tej pracy ogólnie kamerą) niezależnie od szczegółów budowy, można przedstawić w sposób uproszczony, zawierający jedynie najważniejsze parametry takie jak: oś kamery, obraz, ognisko i ogniskowa. Ilustruje to dokładniej rysunek 1.4



Rysunek 1.4: Podstawowe parametry kamery

Ognisko kamery jest punktem, w którym przecinają się promienie świetlne, po przejściu przez układ optyczny kamery.

Ogniskowa określa odległość ogniska od płaszczyzny obrazu.

Oś optyczna jest prostą przechodzącą przez ognisko kamery i jednocześnie prostopadłą do płaszczyzny obrazu. Określa ona kierunek, w którym skierowana jest kamera.

Obraz uzyskiwany z kamery jest wycinkiem płaszczyzny, na którą rzutowane jest położenie postrzeganego punktu P, wzdłuż prostej przechodzącej przez ten punkt i ognisko kamery.

1.3. Układ kamer

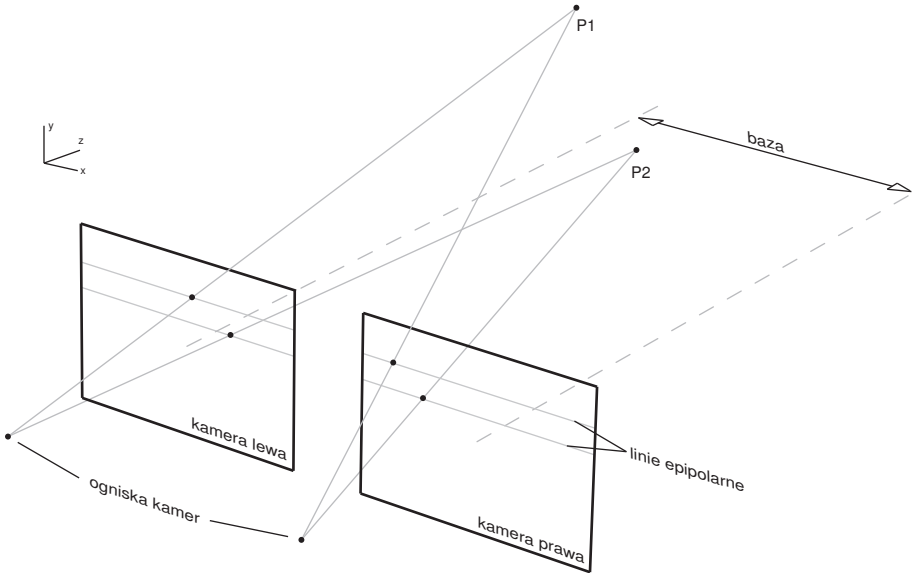
Zestawiając ze sobą dwie lub więcej kamer, otrzymujemy układ o parametrach zależnych od ich wzajemnego ustawienia. Największe znaczenie w systemach stereowizyjnych ma usytuowanie kamer w tzw. układzie kanonicznym. Kanoniczny układ kamer spełnia następujące założenia [9]:

- osie kamer są równoległe
- współrzędne Z ognisk kamer są jednakowe (takie same ogniskowe)

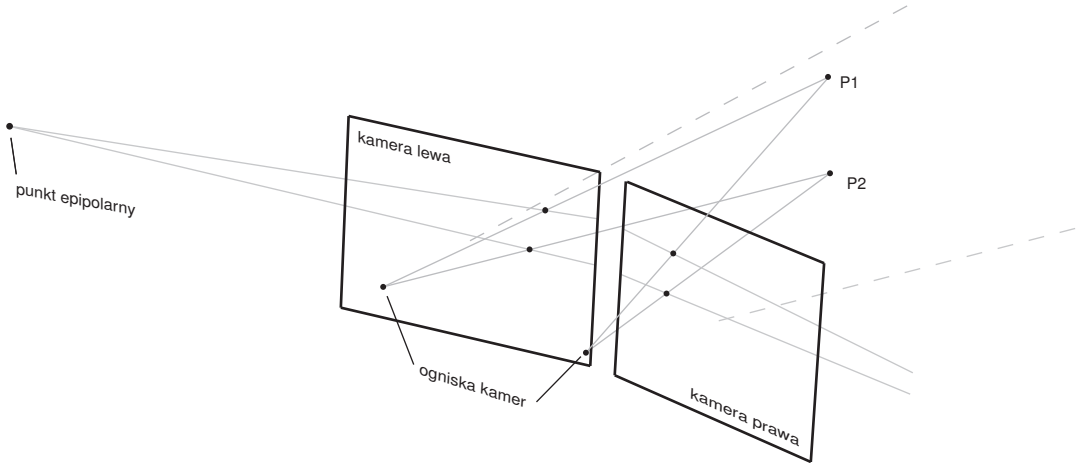
Odległość między osiami kamer nazywamy **bazą** układu kanonicznego.

Istotnym pojęciem w stereowizji dla układu kamer jest tzw. **linia epipolarna**. Pojęcie to najlepiej zilustrować na przykładzie: linią epipolarną dla punktu P1 (rysunek 1.5) w obrazie kamery prawej jest linia zawierająca obrazy wszystkich punktów leżących na prostej łączącej punkt P1 z ogniskiem kamery lewej. Analogicznie jest dla linii epipolarnej dla punktu P1 w obrazie kamery lewej. Znając przebieg linii epipolarnych, można ograniczyć poszukiwania obrazów danego punktu z przestrzeni jedynie do punktów znajdujących się na odpowiadających mu liniach epipolarnych. Dla szczególnego układu kamer,

jakim jest układ kanoniczny, liniami epipolarnymi są proste równoległe do wierszy obrazu (rysunek 1.5). Dla każdego niekanonicznego układu kamer wszystkie linie epipolarne nie są wzajemnie równoległe i przecinają się w jednym punkcie zwanym **punktem epipolarnym** [2] (rysunek 1.6).



Rysunek 1.5: Kanoniczny układ kamer



Rysunek 1.6: Niekanoniczny układ kamer

1.4. Rekonstrukcja sceny trójwymiarowej

Kluczowym etapem rekonstrukcji sceny trójwymiarowej na podstawie obrazów stereoskopowych jest rozwiązanie tzw. problemu odpowiedniości (ang. correspondence problem), polegającego na wyznaczeniu współrzędnych punktów p_l, p_r, tj. rzutów zadanego punktu przestrzeni $P\,(X, Y, Z)$ odpowiednio w obrazach lewej i prawej kamery [9]. Dla kamer w układzie kanonicznym sprowadza się to do przeszukiwania odpowiadających sobie wierszy obu obrazów (współrzędne y poszukiwanych punktów są jednakowe $y_l = y_r$), dzięki czemu problem ulega znacznemu uproszczeniu. Rozwiązaniem problemu odpowiedniości dla danego punktu obrazu jest wyznaczenie tzw. dysparycji (różnicy współrzędnych

związanych z wzajemnym przesunięciem obrazów tego punktu w obu kamerach) [8]. Podczas rozwiązywania problemu dysparycji należy przyjąć jeden z obrazów stereoskopowych jako obraz odniesienia. Jeżeli punktowi z obrazu odniesienia (np. prawego) p_r odpowiada punkt z drugiego obrazu (lewego) p_l, to dysparycja między tymi punktami dana jest wzorem:

$$d_{rl} = x_l - x_r$$

Dla kanonicznego układu kamer dysparycja przyjmuje zawsze wartość nieujemną. Dla punktów leżących nieskończenie daleko od kamer (w praktyce od pewnej skończonej odległości zależnej od rozdzielczości kamer) dysparycja wynosi 0 ($x_l = x_r$). Współrzędne każdego punktu $P\,(X, Y, Z)$, który jest widoczny w obu kamerach, można wyznaczyć ze wzorów [9]:

$$X = \frac{B\,(x_r + x_l - 2x_0)}{2\,(d_{rl})}$$

$$Y = \frac{By}{d_{rl}}$$

$$Z = \frac{Bf_p}{d_{rl}}$$

gdzie:

B - baza układu kamer

$y = y_l = y_r$

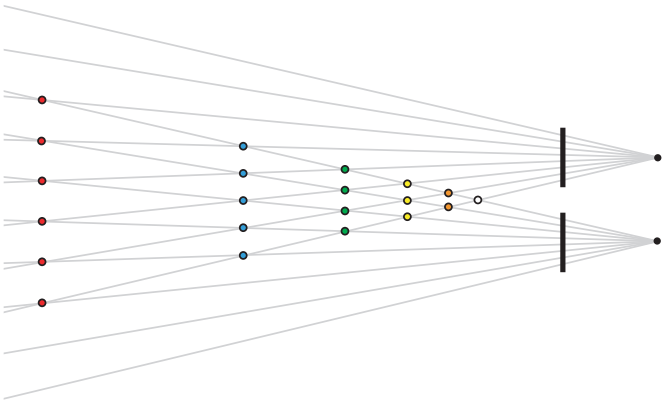
d_{rl} - dysparycja

f_p - ogniskowa kamer podana w liczbie pikseli

x_0 - współrzędna środka obrazu, przez którą przechodzi oś kamery

Powyższe wzory wyznaczają współrzędne względem układu o początku umieszczonym w środku odcinka łączącego ogniska obu kamer.

Ze względu na dyskretny (nieciągły) charakter dysparycji, wyznaczanie położenia obserwowanych obiektów jest już z założenia obarczone pewnym błędem. Błąd ten jest tym większy, im dalej od kamery znajduje się dany obiekt (rysunek 1.7). Wyznaczenie dysparycji dla każdego punktu obrazu tworzy **gęstą mape dysparycji** (mapę głębi) [8].



Rysunek 1.7: Rozmieszczenie punktów w przestrzeni dla kolejnych wartości dysparycji

2. Korekcja zniekształceń geometrycznych kamery, rektyfikacja obrazów

2.1. Rzeczywisty układ kamer

W praktycznych konstrukcjach układów stereowizyjnych bardzo trudno jest uzyskać kanoniczny układ kamer. Stosując jednak odpowiednie rzutowanie, można przekształcić współrzędne pary obrazów z układu niekanonicznego do współrzędnych układu kanonicznego. Przekształcenie takie określa się mianem rektyfikacji obrazów stereo [2]. W celu znalezienia pary odpowiednich macierzy przekształcających, konieczne jest przeprowadzenie kalibracji układu stereowizyjnego ze względu na zewnętrzne oraz wewnętrzne parametry kamer [9].

2.2. Parametry wewnętrzne kamery

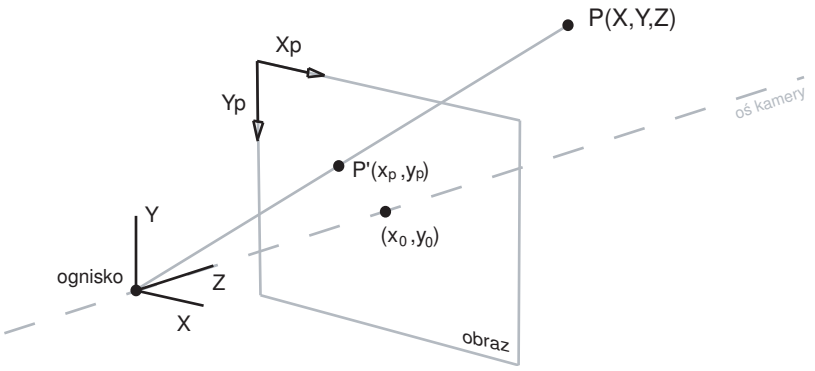
Parametry wewnętrzne określają zniekształcenia geometryczne wprowadzane przez układ optyczny kamery. Część parametrów wewnętrznych zgrupowana jest w tzw. *macierzy kamery* [2, 1]

$$A = \begin{bmatrix} f_x & \gamma & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

gdzie:
f_x, f_y - ogniskowa kamery wyrażona w liczbie pikseli (dla osi x i y),
x_0, y_0 - współrzędne punktu głównego (punkt przecięcia osi kamery z płaszczyzną obrazu),
γ - współczynnik określający kąt miedzy osiami x i y czujnika kamery (dla prostopadłych $\gamma = 0$),

Macierz ta określa związek pomiędzy tzw. znormalizowanymi współrzędnymi punktu w przestrzeni, danych wzorami: $x_n = X/Z$ i $y_n = Y/Z$ a odpowiadającymi im współrzędnymi punktów rejestrowanego obrazu x_p, y_p [9, 1] (rysunek 2.1)

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = A \times \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix}$$



Rysunek 2.1: Kamera z zaznaczonymi układami współrzędnych

Pozostałe parametry wewnętrzne określają zniekształcenia radialne i tangencjalne wprowadzane przez niedoskonałości układu optycznego kamery. Po ich uwzględnieniu, współrzędne znormalizowane możemy wyznaczyć ze wzorów [1]:

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \left(1 + k_1 r^2 + k_2 r^4\right) \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \Delta$$

gdzie:

$$r^2 = {x_n}^2 + {y_n}^2, \qquad \Delta = \begin{bmatrix} 2p_1 x_n y_n + p_2 \left(r^2 + 2{x_n}^2\right) \\ 2p_2 x_n y_n + p_1 \left(r^2 + 2{y_n}^2\right) \end{bmatrix}$$

k_1, k_2 - współczynniki zniekształceń radialnych,
p_1, p_2 - współczynniki zniekształceń tangencjalnych,
Δ - wektor zniekształceń tangencjalnych,

2.3. Parametry zewnętrzne kamery

Parametry zewnętrzne kamery określają pozycję i orientację kamery względem zewnętrznego układu współrzędnych. Przejścia z jednego układu współrzędnych do drugiego można dokonać poprzez złożenie translacji T oraz rotacji R. Wektor translacji T określa wzajemne przesuniecie środków układów współrzędnych, z kolei rotacja R dokonuje przekształcenia odpowiadających sobie osi [2]. Zależność między współrzędnymi punktu P w układzie współrzędnych kamery, a jego współrzędnymi w układzie zewnętrznym, wyraża się następująco:

$$P_Z = R \times P + T$$

W układzie dwóch kamer można wyznaczyć ich wzajemne położenie, znając zewnętrzne parametry obu kamer względem tego samego układu odniesienia.

Rorację i translację kamery drugiej względem kamery pierwszej można wyznaczyć ze wzoru:

$$R_{12} = R_1 R_2^T, \qquad T_{12} = T_2 - R_{12}^T T_1$$

gdzie:
R_1, R_2, T_1, T_2 - macierze rotacji oraz wektory translacji odpowiednio dla obu kamer
R_{12}, T_{12} - macierz rotacji i wektor translacji pierwszej kamery względem drugiej

Przeliczanie współrzędnych z układu jednej kamery do drugiej odbywa się wtedy według zależności:

$$P_1 = R_{12}\left(P_2 - T_{12}\right)$$

gdzie:
P_1, P_2 - współrzędne punktu P w układzie współrzędnych kamery pierwszej i drugiej

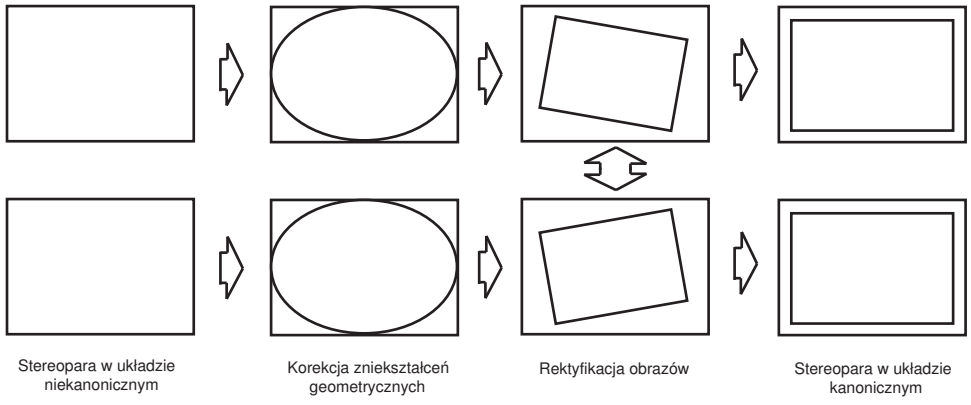
2.4. Rektyfikacja obrazów

Mając dane parametry zewnętrzne układu niekanonicznego (R, T) oraz parametry wewnętrzne kamery prawej i lewej, można wyznaczyć parę macierzy translacyjnych M_R, M_L, dzięki którym możliwa jest rektyfikacja obrazów z kamer będących w układzie niekanonicznym. Współrzędne obrazów „wyprostowanych" dla kamery lewej (x_{rectL}, y_{rectL}) oraz prawej (x_{rectR}, y_{rectR}) określone są następującymi wzorami [7, 9]:

$$\begin{bmatrix} a_L \\ b_L \\ c_L \end{bmatrix} = M_L \times \begin{bmatrix} x_{pL} \\ y_{pL} \\ 1 \end{bmatrix} \qquad \begin{bmatrix} a_R \\ b_R \\ c_R \end{bmatrix} = M_R \times \begin{bmatrix} x_{pR} \\ y_{pR} \\ 1 \end{bmatrix}$$

$$x_{rectL} = a_L/c_L \qquad x_{rectR} = a_R/c_R$$
$$y_{rectL} = b_L/c_L \qquad y_{rectL} = b_L/c_L$$

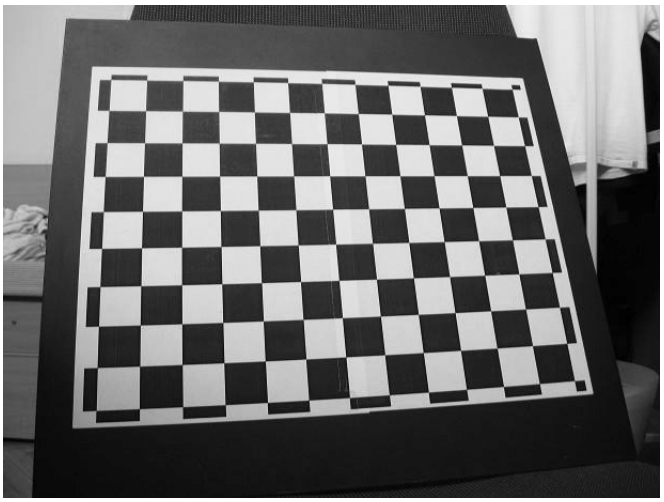
Cały proces korekcji obrazów stereowizyjnych, można przedstawić blokowo jak na rysunku 2.2.



Rysunek 2.2: Etapy korekcji obrazów stereowizyjnych

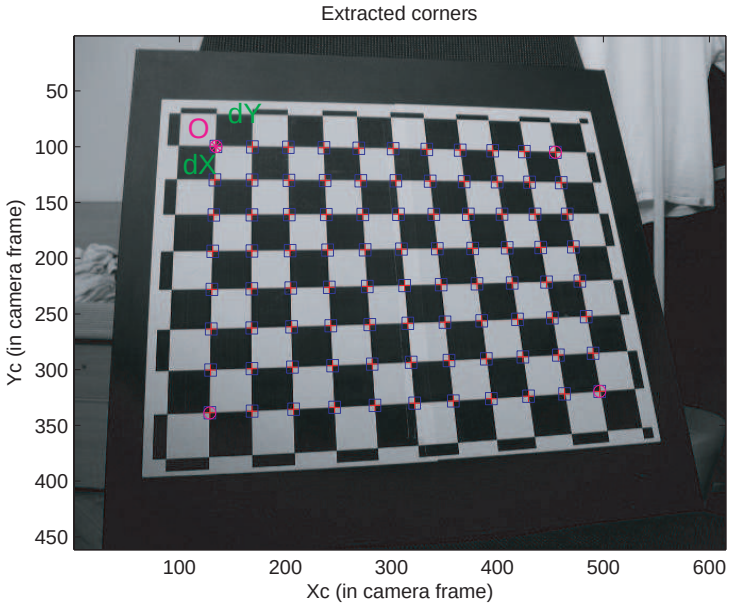
2.5. Kalibracja kamer z Camera Calibration Toolbox for Matlab

Na potrzeby tej pracy, kalibrację kamer wykonano z wykorzystaniem pakietu Camera Calibration Toolbox. Jego autorem jest Jean-Yves Bouguet, doktor California Institute of Technology w Pasadenie (USA) [1]. Pakiet ten pozwala wyznaczyć wewnętrzne i zewnętrzne parametry kamery na podstawie obrazu płaskiego przedmiotu z naniesioną biało-czarną szachownicą (rysunek 2.3).



Rysunek 2.3: Obraz wejściowy do kalibracji kamery

Podczas procesu kalibracji należy podać długość boku jednego kwadratu szachownicy, jej rozmiar (ilość kwadratów w pionie i poziomie) oraz wskazać kursorem wszystkie cztery narożniki w odpowiedniej kolejności. Następnie toolbox wyznaczy orientacyjnie położenia narożników dla każdego małego kwadratu szachownicy, po czym dokona ich optymalizacji (rysunek 2.4).



Rysunek 2.4: Lokalizacja narożników w szachownicy

Po wyznaczeniu współrzędnych wszystkich narożników, można już rozpocząć właściwy proces kalibracji kamery. Dla danych z obrazka powyżej otrzymujemy następujące wyniki:

$f_x = 711.49475 \pm 14.30421$

$f_y = 730.14404 \pm 21.36334$

$x_0 = 319.5$

$x_0 = 239.5$

$\gamma = 0$

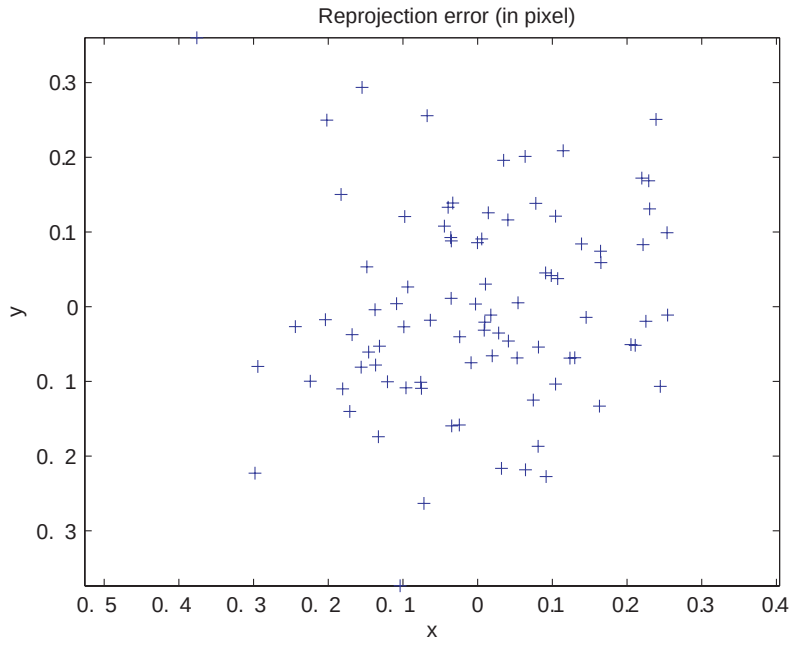
$k_1 = -0.31619 \pm 0.03658$

$k_2 = 0.28540 \pm 0.02834$

$p_1 = -0.00518$

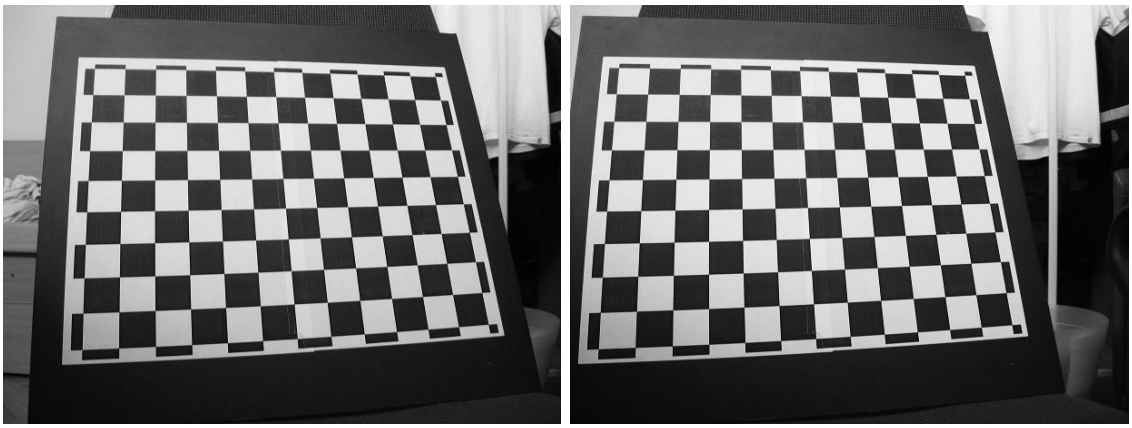
$p_2 = 0$

Rysunek 2.5 przedstawia rozrzut błędu wyznaczania pozycji narożników szachownicy względem otrzymanych wyników. Jeżeli punkty na tym wykresie tworzą zwartą grupę blisko środka układu współrzędnych, to kalibrację można uznać za prawidłową.



Rysunek 2.5: Błąd wyznaczania pozycji narożników

Camera Calibration Toolbox umożliwia także rektyfikację obrazów stereoskopowych. W pierwszej kolejności należy wykonać osobno kalibrację dla prawej i lewej kamery. Posiadając parametry wewnętrzne dla obu kamer, można wyznaczyć parametry zewnętrzne dla całego układu (przyjmując kamerę lewą za kamerę odniesienia). Rysunek 2.6 przedstawia przykładowe obrazy wejściowe.

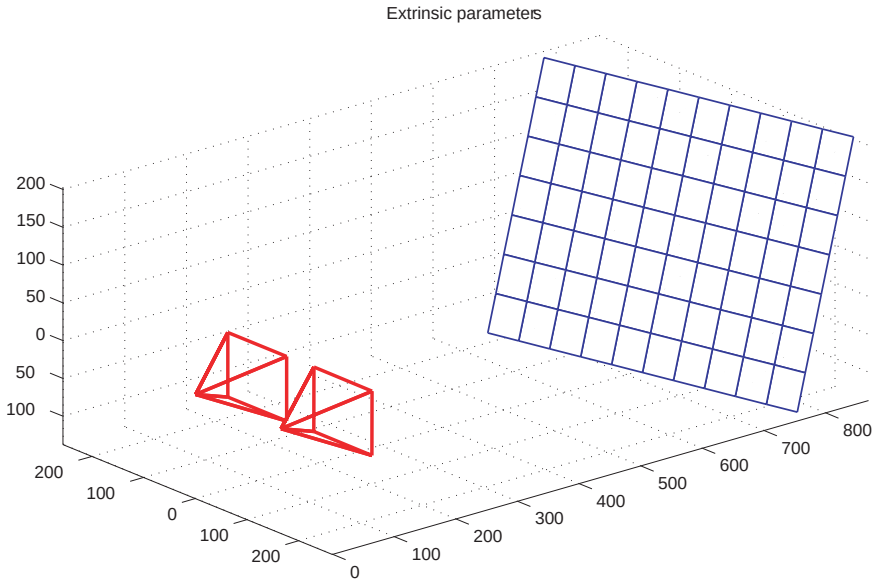


Rysunek 2.6: Obrazy wejściowe do kalibracji systemu stereoskopowego

Jako wyniki kalibracji dostajemy nowe, poprawione parametry wewnętrzne obu kamer oraz wektor translacji T i rotację R kamery prawej, wzgledem kamery lewej.

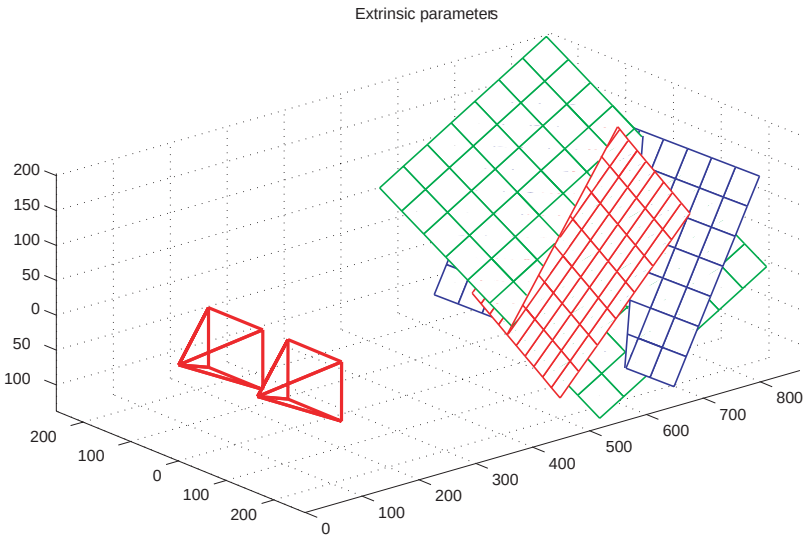
$$T = [-165.53390 \quad 0.85510 \quad -1.10264]$$
$$R = [0.00222 \quad -0.00626 \quad -0.00091]$$

Wyznaczoną pozycję obu kamer można zobrazować na diagramie (rysunek 2.7)



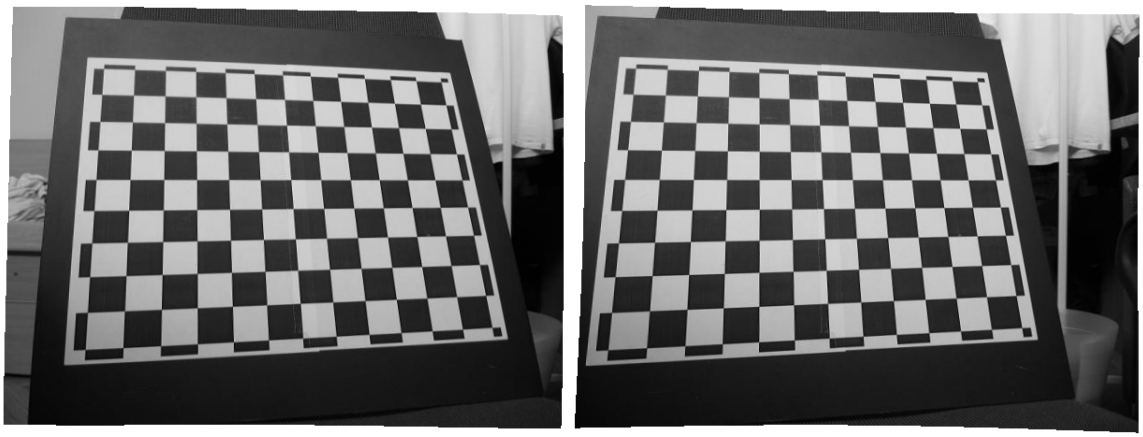
Rysunek 2.7: Wizualizacja układu kamer w przestrzeni

Wykonując kilka serii zdjęć, za każdym razem z innym ustawieniem szachownicy kalibracyjnej, można w prosty sposób zwiększyć dokladność obliczeń (rysunek 2.8)



Rysunek 2.8: Zwiększenie dokładności kalibracji

Wyznaczenie parametrów wewnętrznych i zewnętrznych układu kamer, pozwala na rektyfikację obrazów wejściowych. Jako wynik dostajemy parę obrazów jaką otrzymano by dysponując kamerami tworzącymi układ kanoniczny (rysunek 2.9).



Rysunek 2.9: Para obrazów po rektyfikacji

3. Algorytm wyznaczania gęstej mapy dysparycji

3.1. Dopasowanie obszarami, miara SAD

Właściwe wyznaczenie pozycji obserwowanych obiektów, sprowadza się do wyliczenia dysparycji dla każdego punktu obrazu odniesienia. Dysponując obrazami z kanonicznego układu kamer (lub obrazami po rektyfikacji) wystarczy znaleźć parę odpowiadających sobie punktów w odpowiednich wierszach obrazów. W praktyce dopasowanie pojedynczych punktów jest niezwykle trudne, dlatego częstym sposobem wyznaczania dysparycji jest tzw. dopasowanie obszarami [3, 5, 7, 10]. Algorytmy oparte na dopasowaniu obszarami porównują do siebie prostokątne fragmenty obu obrazów, zawierające badany punkt wraz z pewnym otoczeniem. Najczęstszą miarą służącą do wyznaczania podobieństwa tych obszarów, jest suma bezwzględnych różnic (SAD - ang. Sum of Absolute Differences) wyliczona według wzoru [7]:

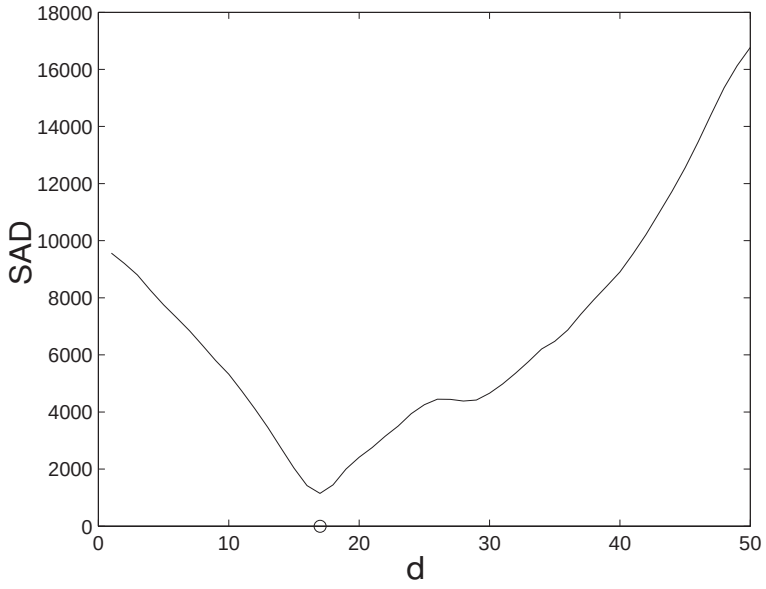
$$
\begin{aligned}
SAD\left(x_r, y_r, d\right) = \sum_{i=-\frac{1}{2}(winy-1)}^{\frac{1}{2}(winx-1)} \sum_{j=-\frac{1}{2}(winy-1)}^{\frac{1}{2}(winx-1)} & \left[\left|R_r\left(x_r+i, y_r+j\right) - R_l\left(x_l+i+d, y_l+j\right)\right| + \right. \\
& +\left|G_r\left(x_r+i, y_r+j\right) - G_l\left(x_l+i+d, y_l+j\right)\right| + \\
& \left.+\left|B_r\left(x_r+i, y_r+j\right) - B_l\left(x_l+i+d, y_l+j\right)\right|\right]
\end{aligned}
$$

gdzie:
x_p, y_p - współrzędne badanego punktu obrazu
d - aktualne przesunięcie
$winx$, $winy$ - rozmiary okna dopasowania (w pikselach)
R, G, B - składowe obrazów odpowiadające podstawowym kolorom

Zakres przesunięcia dla którego liczona jest miara SAD, należy dobrać adekwatnie do względnego „przemieszczenia" obiektów znajdujących się najbliżej kamery. Zbyt mała wartość spowoduje powstanie dużych błędów w wynikowej mapie głębi, a zbyt duża niepotrzebnie wydłuży czas obliczeń. Wyznaczenie miary SAD dla całego zakresu przesunięcia tworzy funkcję dopasowania danego punktu obrazu odniesienia. Rysunek 3.2 przedstawia obrazowo ideę algorytmu dopasowującego obszarami dla jednego punktu stereopary 3.1, a rysunek 3.3 jego funkcję dopasowania.

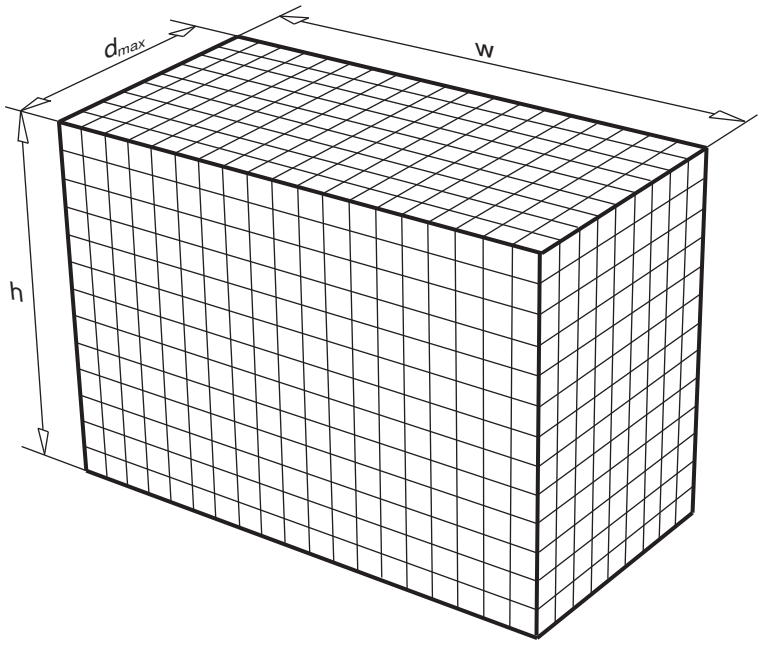
Rysunek 3.1: Stereopara testowa

Rysunek 3.2: Idea algorytmu dopasowania obszarami



Rysunek 3.3: Funkcja dopasowania miary SAD dla punktu z rysunku 3.2

Wyznaczone funkcje dopasowania dla całego obrazu odniesienia tworzą trójwymiarową macierz o wymiarach $w \times h \times d_{max}$ (w, h - szerokość i wysokość obrazu odniesienia, d_{max} - maksymalne przesunięcie na obrazie), zwaną w literaturze „przestrzenią dysparycji” (ang. disparity space volume) [9] (rysunek 3.4).

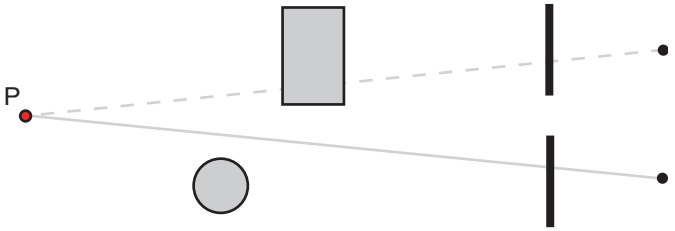


Rysunek 3.4: Przestrzeń dysparycji

Ostatnim krokiem do wyznaczenia mapy głębi jest określenie minimalnej wartości dla każdej funkcji dopasowania tworzącej przestrzeń dysparycji. Konkretna wartość tego minimum nie jest istotna, liczy się jedynie jego pozycja, ponieważ właśnie ta pozycja jest wartością dysparycji dla danego punktu obrazu.

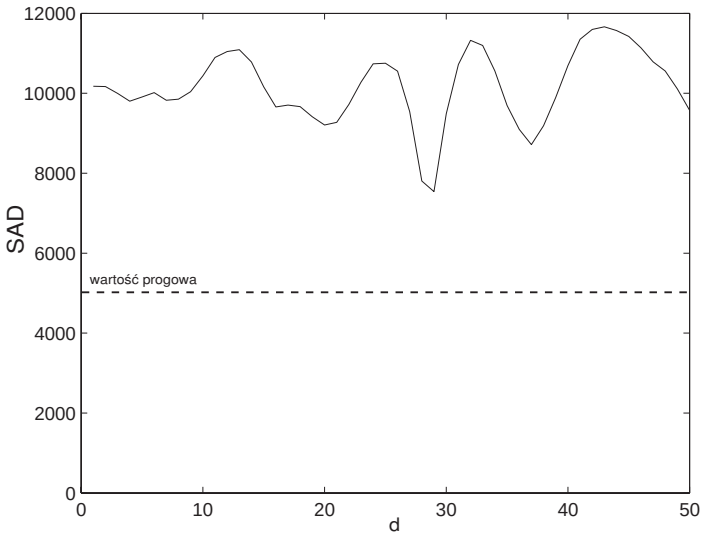
3.2. Problem przesłonięć

Do poprawnego działania algorytmu dopasowania obszarami konieczna jest widoczność danego punktu w obu obrazach stereowizyjnych. Jednak już sama idea stereowizji powoduje, że warunek ten nie może być spełniony dla wszystkich punktów obserwowanej sceny. Niewielkie przesunięcia postrzeganych przedmiotów (zależne od ich odległości od kamery), na których bazuje cały algorytm stereowizyjny, powodują jednocześnie przesłonięcie niektórych fragmentów obrazu (rysunek 3.5). W rezultacie do wyznaczenia położenia niektórych punktów dysponujemy jedynie informacją z jednej kamery, co uniemożliwia uzyskanie poprawnych wyników. Dla tych punktów algorytm zwróci mniej lub bardziej przypadkowe wartości zależne od otoczenia danego punktu oraz od właściwości obiektu przesłaniającego.



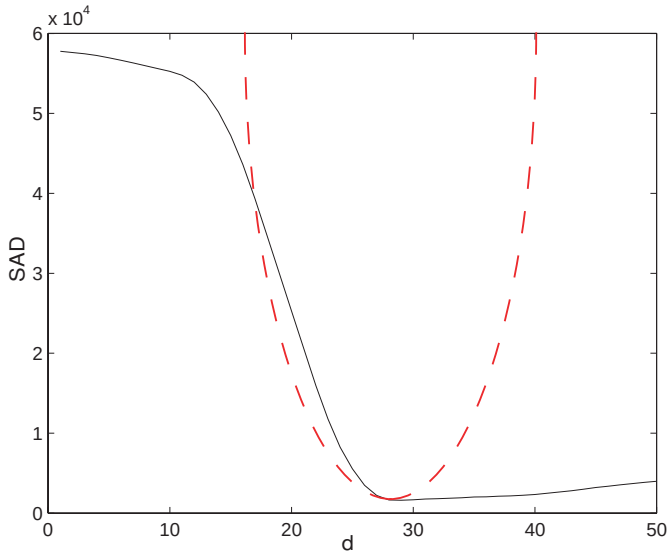
Rysunek 3.5: Problem przesłonięć (punkt P jest widoczny tylko dla jednej kamery)

Dla poprawy jakości generowanej w ten sposób mapy głębi korzystne byłoby wykluczenie błędnych wyników (zastąpienie ich zerową wartością dysparycji). W tym celu konieczne jest wprowadzenie kryteriów oceny poprawności działania algorytmu. Jednym z takich kryteriów może być wprowadzenie wartości progowej dla znalezionego minimum funkcji dopasowania. W ten sposób wykluczone zostałyby punkty, dla których nie znaleziono dostatecznie dobrego dopasowania w obrazie z sąsiedniej kamery (rysunek 3.6).



Rysunek 3.6: Wartość progowa dla funkcji dopasowania

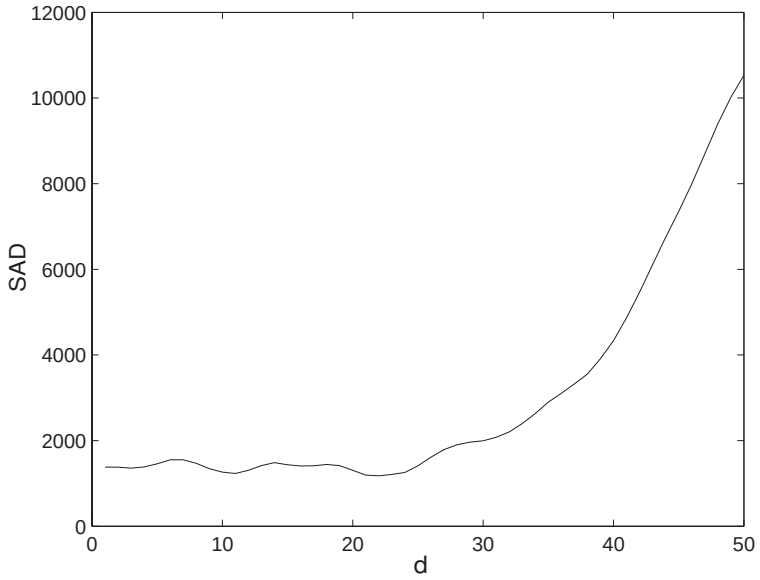
Innym kryterium może być kształt funkcji dopasowania w okolicach jej minimum. Dla poprawnych wartości, minimum funkcji dopasowania znajduje się w „dolinie” o stromych zboczach. Przykład niespełnienia tego kryterium pokazuje rysunek 3.7.



Rysunek 3.7: Błędny przebieg funkcji dopasowania

3.3. Problem poziomych linii

Nawet w przypadku dobrej widoczności punktów w obu kamerach i spełnienia wszystkich kryteriów poprawności, algorytm dopasowania obszarami może zwrócić błędne wartości dysparycji. Dotyczy to punktów obrazu tworzących poziome krawędzie obserwowanych obiektów. Przyczyną powstawania tych błędów jest przesunięcie kamer w układzie stereowizyjnym tylko w płaszczyźnie poziomej. W skutek tego przebieg funkcji dopasowania (na jej początkowym odcinku) dla punktów na tych krawędziach jest zniekształcony (rysunek 3.8).



Rysunek 3.8: Zniekształcenie funkcji dopasowania dla punktów na poziomych krawędziach

4. Realizacja układu stereowizyjnego (dwie kamery)

4.1. Akwizycja obrazów

W ramach tej pracy zrealizowano system, umożliwiający generowanie map głębi na podstawie obrazów stereoskopowych z dwóch oraz większej ilości kamer. Ten rozdział opisuje realizację klasycznego systemu stereowizyjnego z wykorzystaniem dwóch kamer. Systemy z większą ilością kamer opisane są w następnym rozdziale. Rejestracja obrazów wejściowych odbywała się z użyciem kompaktowego aparatu fotograficznego marki Canon (A560) zamontowanego na prowadnicy liniowej (rysunek 4.1). Taki sposób akwizycji obrazu, pozwalał uzyskać stereoskopowe zdjęcia nieruchomych obiektów w układzie zbliżonym do kanonicznego. Dodatkowo w prosty sposób można było regulować bazę „układu kamer" wpływając tym samym na maksymalną wartość dysparycji w obrazie wynikowym. Prowadnica została wykonana z dwóch szlifowanych wałków o średnicy 9 mm zamontowanych stabilnie w niewielkich stalowych blokach. Po wałkach na tulejkach ślizgowych przesuwa się wózek z zamontowanym aparatem fotograficznym. Między aparatem a wózkiem znajdował się jeszcze element dystansowy pozwalający na regulację orientacji aparatu w trzech osiach. Element ten umożliwiał skierowanie aparatu w pożądanym kierunku oraz dostrojenie całego układu do wymaganej precyzji.

Rysunek 4.1: Wyposażenie umożliwiające wykonywanie zdjęć stereoskopowych

4.2. Środowisko obliczeniowe, struktura algorytmu

Jako środowisko do wykonywania obliczeń i tworzenia aplikacji wykorzystano program Matlab (w wersji 2011a). Rysunek 4.2 przedstawia wygląd okna głównego wykonanej aplikacji z przykładową stereoparą (widoczny jest tylko obraz odniesienia). Program umożliwia wczytanie dowolnych stereopar o maksymalnej rozdzielczości 600x800 pikseli oraz zapisanie wyników obliczeń w formacie BMP. Generowane mapy głębi kodowane są w skali odcieni szarości.

Algorytm wyznaczania map głębi można podzielić na cztery etapy:

Etap1: Utworzenie tablicy danych wejściowych

Etap2: Sumowanie po wierszach

Etap3: Sumowanie po kolumnach, utworzenie przestrzeni dysparycji

Etap4: Wyznaczenie minimów funkcji dopasowań, zestawienie mapy głębi

W dalszych podpunktach tego rozdziału, poszczególne etapy działania algorytmu zostaną szczegółowo opisane.

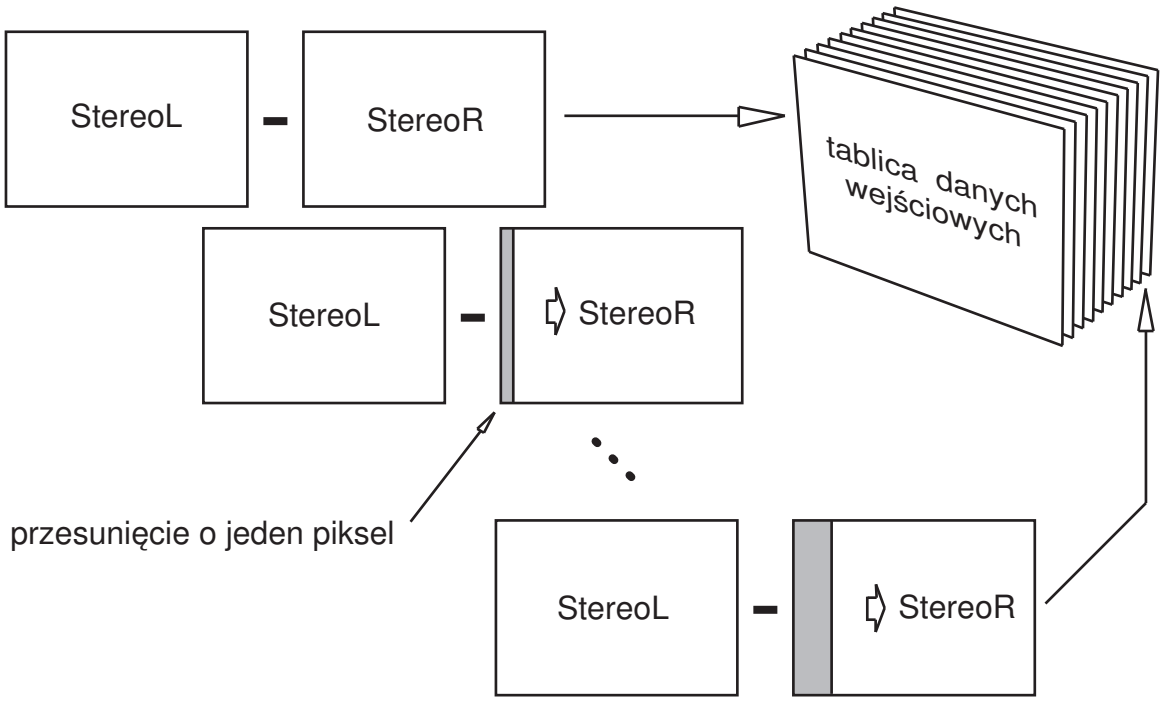
Rysunek 4.2: Okno główne programu

4.3. Etap 1

Początkowy etap przetwarzania obrazów stereoskopowych na mapę głębi, polega na utworzeniu tablicy danych wejściowych o wymiarach $w \times h \times d_{max}$(gdzie: w, h - szerokość i wysokość stereopar w pikselach, d_{max} - maksymalne przesunięcie)[1]. Pierwszy element tej tablicy to prosta różnica obu stereopar. Każdy następny również jest podobny, przy czym obraz z prawej kamery jest przesuwany za każdym razem o jeden piksel w prawo. Czynność ta powtarzana jest d_{max} razy. Graficznie ilustruje to rysunek 4.3. Ostatnią operacją pierwszego etapu jest zastąpienie każdej wartości w tablicy na jej wartość bezwzględną.

Stosowny kod w Matlabie wykonujący powyższe zadania będzie wyglądał następująco:

```
etap1 = zeros(h,w,3,k,'int32');     //inicjacja tablicy zerami
for d = 1:dmax
   etap1(:,:,:,d) = circshift(StereoR,d-1);
end
for d = 1:dmax
   etap1(:,:,:,d) = etap1(:,:,:,d) - StereoL;
end
etap1 = abs(etap1);     //wartość bezwzględna
```

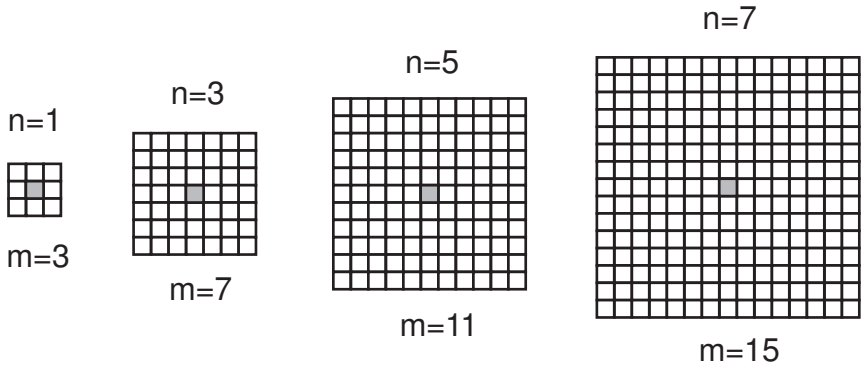


Rysunek 4.3: Schemat tworzenia tablicy danych wejściowych

[1]Prawdziwy rozmiar tablicy danych wejściowych wynosi $w \times h \times 3 \times d_{max}$ (składowe RGB), jednak ze względu na takie same operacje dla każdego koloru, pominięto ten wymiar dla uproszczenia rozważań.

4.4. Etap 2

Następną czynnością niezbędną z punktu widzenia algorytmu dopasowania obszarami, jest sumowanie otoczenia dla każdego punktu tablicy z etapu pierwszego. Proces ten rozbity jest na dwa części: sumowanie po wierszach i sumowanie po kolumnach. Otoczenie danego punktu tablicy definiowane jest jako zbiór przyległych do niego punktów tworzących prostokątny obszar. Otoczenie charakteryzują dwa parametry: n - „promień" otoczenia oraz m - jego szerokość. Parametry te są ze sobą powiązane według zależności: $m = 2n + 1$. Rysunek 4.4 przedstawia te wielkości w sposób graficzny.

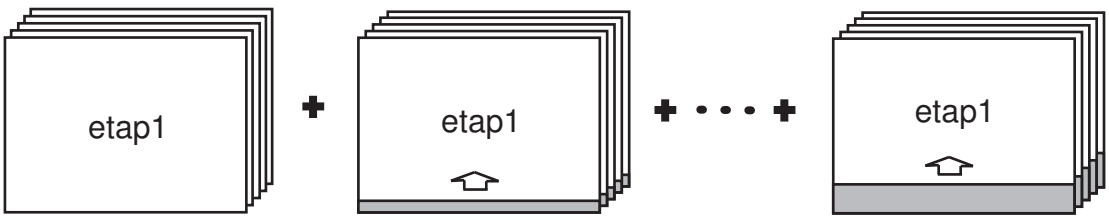


Rysunek 4.4: Wielkość obszaru dopasowania

Sam proces sumowania po wierszach odbywa się jednocześnie dla wszystkich elementów tablicy. Składa się on z m sum cząstkowych, w których dodawana jest zawartość tablicy z etapu pierwszego do samej siebie, przesuniętej za każdym razem o jeden piksel w górę. Ostatecznie otrzymujemy tablicę, której każdy element zawiera sumę m elementów sąsiadujących ze sobą w pionie w tablicy sprzed dodawania. Dla elementów znajdujących się na krawędzi tablicy, dla których nie można przeprowadzić pełnego dodawania, wynik jest nieistotny. Punkty te nie są brane pod uwagę przy generowaniu ostatecznej mapy głębi.

Kod w Matlabie dla etapu drugiego:

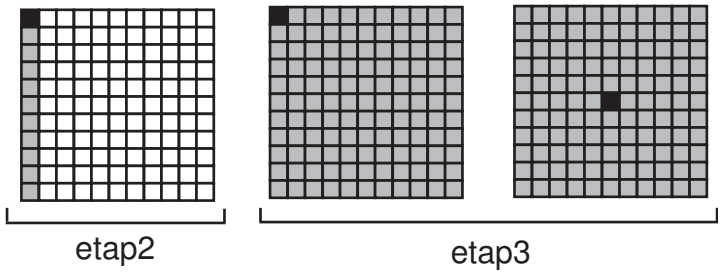
```
for r = 1:m-1
   etap2 = etap2 + circshift(etap1,[-r 0]);
end
```



Rysunek 4.5: Sumowanie po wierszach

4.5. Etap 3

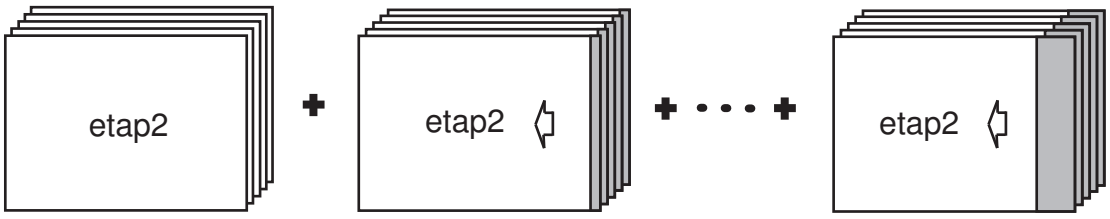
Trzeci etab algorytmu stanowi drugą część sumowania otoczenia elementów tablicy początkowej. Dodawanie odbywa się względem kolumn w sposób analogiczny do tego z poprzedniego etapu (rysunek 4.7). Wcześniej dodawana była tablica z etapu prerwszego, tym razem sumowana jest tablica z etapu drugiego. W efekcie każdy element wynikowy będzie zawierał w sobie sumę m elementów z tablicy z etapu drugiego oraz jednocześnie sumę $m \times m$ elmentów z tablicy początkowej. Wykonanie sumowania otoczenia punktów tablicy początkowej w ten sposób sprawia, że jest ono przesunięte względem tych punktów o wektor $[-n, -n]$, dlatego ostatnią czynniścią etapu trzeciego jest przesunięcie całej tablicy wynikowej o wektor $[n, n]$. Uzyskujemy w ten sposób prawidłową pozycję otoczenia (rysunek 4.6).



Rysunek 4.6: Zakres sumowania (szary kolor) w otoczeniu elementu tablicy (czarny punkt)

Kod w Matlabie dla trzeciego etapu:

```
for c = 1:m-1
   etap3 = etap3 + circshift(etap2,[0 -c]);
end
etap3 = circshift(etap3,[n n]);
```



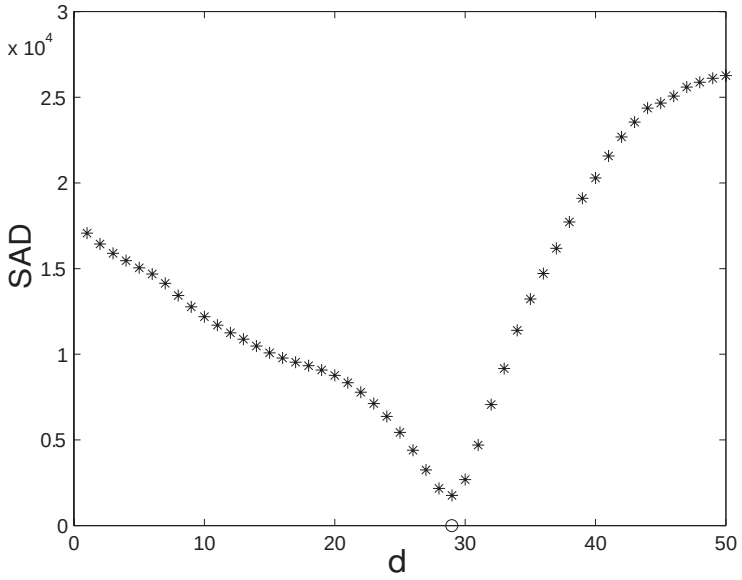
Rysunek 4.7: Sumowanie po kolumnach

Funkcja `circshift` użyta do przesuwania macierzy w istocie powoduje jej „przewinięcie” (zmianę indeksowania kolumn lub wierszy). Nie ma to wpływu na poprawność generowania mapy głębi.

Po zakończeniu trzeciego etapu obliczeń, uzyskana macierz jest już właściwą „przestrzenią dysparycji”, czyli zawiera wyznaczone funkcje dopasowania dla każdego punktu obrazu odniesienia.

4.6. Etap 4

Czwarty, ostatni etap wyznaczania mapy głębi polega na znalezieniu minimum funkcji dopasowania dla każdego punktu obrazu. Indeks tego minimum określa o ile należy przesunąć obrazy wejściowe względem siebie, aby uzyskać ich najlepsze dopasowanie (w danym punkcie) i jest on szukaną wartością dysparycji. Rysunek 4.8 przedstawia przykładową funkcję dopasowania z widocznym minimum oraz zaznaczoną wartością dysparycji dla tego minimum (oś pozioma).



Rysunek 4.8: Przykładowa funkcja dopasowania z przestrzeni dysparycji

Kod w Matlabie dla czwartego etapu:

```
for x = 1:w
   for y = 1:h
      [min1, idx1] = min(etap3(x,y,:));
      etap4(x,y) = idx1;
   end
end
```

Dodatkowo na tym etapie możliwe jest wprowadzenie wartości progowej (zmienna `level`) dla znalezionego minimum funkcji dopasowania. Pozwoli to wyeliminować część błędów powstałych w wyniku przesłonięć w obserwowanej scenie. Zmodyfikowany kod powinien wyglądać jak poniżej:

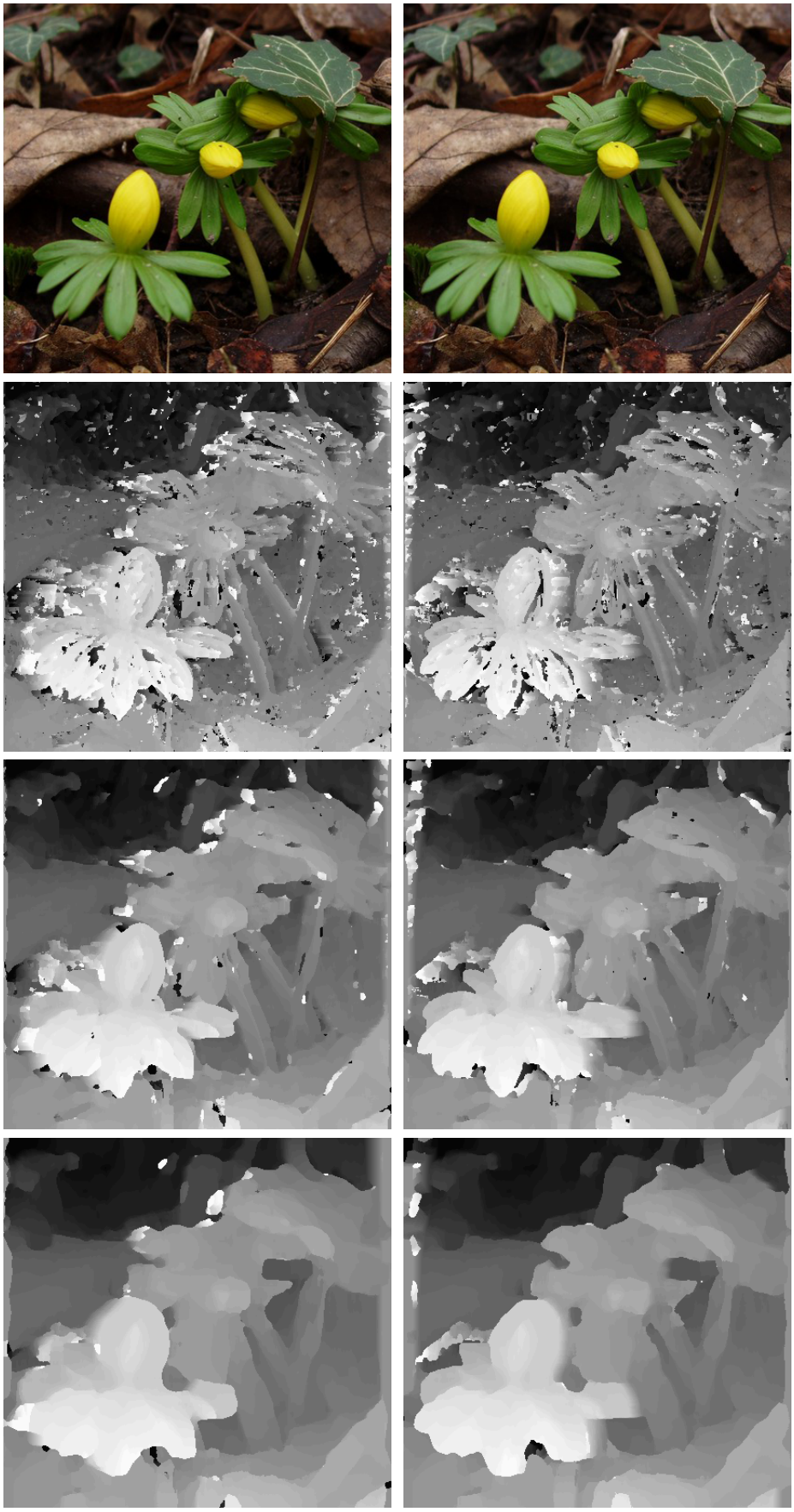
```
for x = 1:w
   for y = 1:h
      [min1, idx1] = min(etap3(x,y,:));
      if(min1 < level)
         etap4(x,y) = 0;
      else
         etap4(x,y) = idx1;
      end
   end
end
```

Oprócz wartości progowej, można zaimplementować jeszcze kryterium kształtu funkcji dopasowania. Polega ono na zapamiętaniu trzech najmniejszych wartości funkcji dopasowania i zatwierdzeniu danego minimum jeśli trzecia najmniejsza wartość jest większa od pierwszej o określony współczynnik (zmienna `ratio`). Wymagane zmiany w kodzie są przedstawione poniżej:

```
for x = 1:w
   for y = 1:h
      [min1, idx1] = min(etap3(x,y,:));   //najmniejsza wartość
      etap3(x,y,idx1) = 100000;         //nadpisanie poprzedniego minimum
      [min2, idx2] = min(etap3(x,y,:));   //druga najmniejsza wartość
      etap3(x,y,idx1) = 100000;         //nadpisanie poprzedniego minimum
      [min3, idx3] = min(etap3(x,y,:));   //trzecia najmniejsza wartość
      if(min1 < level && min3 < min1*ratio)
         etap4(x,y) = 0;
      else
         etap4(x,y) = idx1;
      end
   end
end
```

4.7. Wyniki obliczeń

Rysunek 4.9 przedstawia wyniki działania opisanego algorytmu, w zależności od rozmiaru obszaru dopasowania uwzględnianego w obliczeniach. Na górze strony przedstawiona jest wejściowa stereopara a poniżej wyniki obliczeń w dwóch kolumnach. W lewej kolumnie obrazem odniesienia jest lewy obraz stereopary a w prawej prawy. Pierwszy zestaw wyników otrzymano dla obszaru dopasowania 5x5 pikseli, a kolejne odpowiednio dla 11x11 i 21x21 pikseli. Im większy jest obszar dopasowania tym wynikowa mapa głębi jest bardziej jednorodna, niestety jednocześnie maleje dokładność odwzorowania krawędzi obserwowanych obiektów. Łatwo zauważyć błędy powstałe w skutek przesłonięć w obserwowanej scenie. W zależności czy obrazem odniesienia jest prawy czy lewy obraz stereopary błędy te powstają z prawej lub lewej strony obiektów przesłaniających. Skuteczność metod eliminacji tych błędów jest ograniczona a ich efekty przedstawia rysunek 4.10. Górny obraz to niezmieniony wynik obliczeń (dla porównania). Przy wyznaczaniu środkowego wzięto pod uwagę wartość progową na poprawne minimum, a przy dolnym dodatkowo kryterium kształtu minimum. Kryterium kształtu daje zdecydowanie lepsze wyniki, ale odrzucona zostaje też część poprawnych danych. Dodatkowo uwzględnienie tych metod istotnie wydłuża czas obliczeń. Jedynym sposobem na dalszą poprawę jakości mapy głębi jest dostarczenie do algorytmu większej ilości informacji poprzez zastosowanie dodatkowych kamer. Możliwe rozwiązania przedstawione są w następnym rozdziale.



Rysunek 4.9: Wyniki obliczeń (dwie kamery)

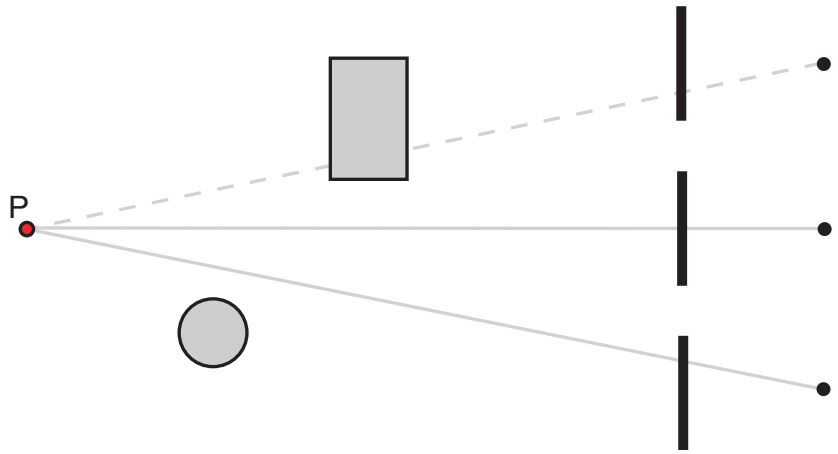
Rysunek 4.10: Poprawa jakości mapy głębi

5. Realizacja układu stereowizyjnego (kilka kamer)

5.1. Trzy kamery

W poprzednim rozdziale przedstawiony został klasyczny system stereowizyjny, wykorzystujący dwie kamery. Przedstawione wyniki pozwalają poznać możliwości tego układy w zakresie rozpoznawania położenia obiektów w obserwowanej scenie. Jednocześnie uwidaczniają się wady takiego rozwiązania, ograniczające dokładność generowanych map głębi. Jedynym sposobem poprawy działania układu stereowizyjnego jest jego rozbudowanie. Dokładając dodatkową kamerę uzyskujemy nowe dane o obserwowanej scenie oraz szansę na poprawę jakości map głębi. Trzecią kamerę najlepiej umiejscowić tak, aby tworzyła ze środkową kamerą układ kanoniczny o takiej samej bazie jak w pierwszym układzie. W ten sposób otrzymujemy potrójny układ kamer, w którym zgodnie z intuicją kamera środkowa będzie kamerą odniesienia, a pozostałe kamery dostarczają niejako symetrycznych obrazów pomocniczych. Dla takiego układu niemal każdy punkt widoczny na obrazie kamery odniesienia, jest widoczny również na obrazie przynajmniej jednej kamery bocznej (rysunek 5.1). Dzięki temu powinniśmy uzyskać istotną poprawę działania algorytmu.

Akwizycja obrazów dla układu z trzema kamerami odbywała się podobnie jak w poprzednim rozdziale (z zastosowaniem aparatu fotograficznego i prowadnicy liniowej). Tym razem wykonywane były trzy zdjęcia przy zachowaniu jednakowych przesunięć w osi prowadnicy. Wartość tych przesunięć uzależniona była od odległości aparatu od obserwowanych obiektów.



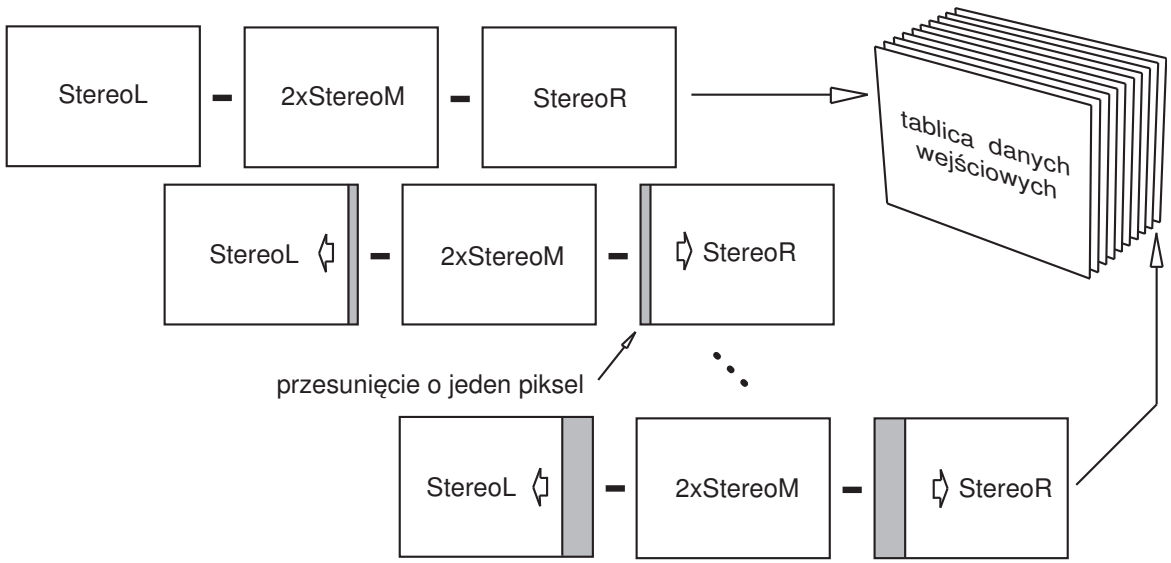
Rysunek 5.1: Rozmieszczenie kamer w układzie potrójnym

5.2. Modyfikacje algorytmu

Uwzględnienie danych z trzeciej kamery wymaga zmian w algorytmie wyznaczającym mapę głębi. Obraz ze środkowej kamery jest obrazem odniesienia, od którego w pierwszym etapie algorytmu należy odjąć odpowiednio przesunięte obrazy z bocznych kamer. W takim przypadku odejmowanie występuje dwukrotnie i aby zachować odpowiednie znaczenie danych wyjściowych, konieczne jest pomnożenie obrazu odniesienia przez dwa. Podobnie jak poprzednio, całe odejmowanie powtarzane jest d_{max} razy, przy czym za każdym razem obrazy z kamer bocznych przesuwane są o kolejny piksel w przeciwnych kierunkach. Graficznie proces ten ilustruje rysunek 5.2. Po zakończeniu wszystkich odejmowań z całej tablicy wyciągana jest wartość bezwzględna. Dzięki modułowej konstrukcji algorytmu dalsze jego etapy mogą pozostać bez zmian.

Zmodyfikowany kod etapu pierwszego:

```
etap1 = zeros(h,w,3,k,'int32');     //inicjacja tablicy zerami
for d = 1:dmax
   etap1(:,:,:,d) = circshift(StereoR,d-1);
end
for d = 1:dmax
   etap1(:,:,:,d) = etap1(:,:,:,d) + circshift(StereoL,-d+1);
end
for d = 1:dmax
   etap1(:,:,:,d) = etap1(:,:,:,d) - 2*StereoM;
end
etap1 = abs(etap1);     //wartość bezwzględna
```



Rysunek 5.2: Etap 1 algorytmu dla układu trzech kamer

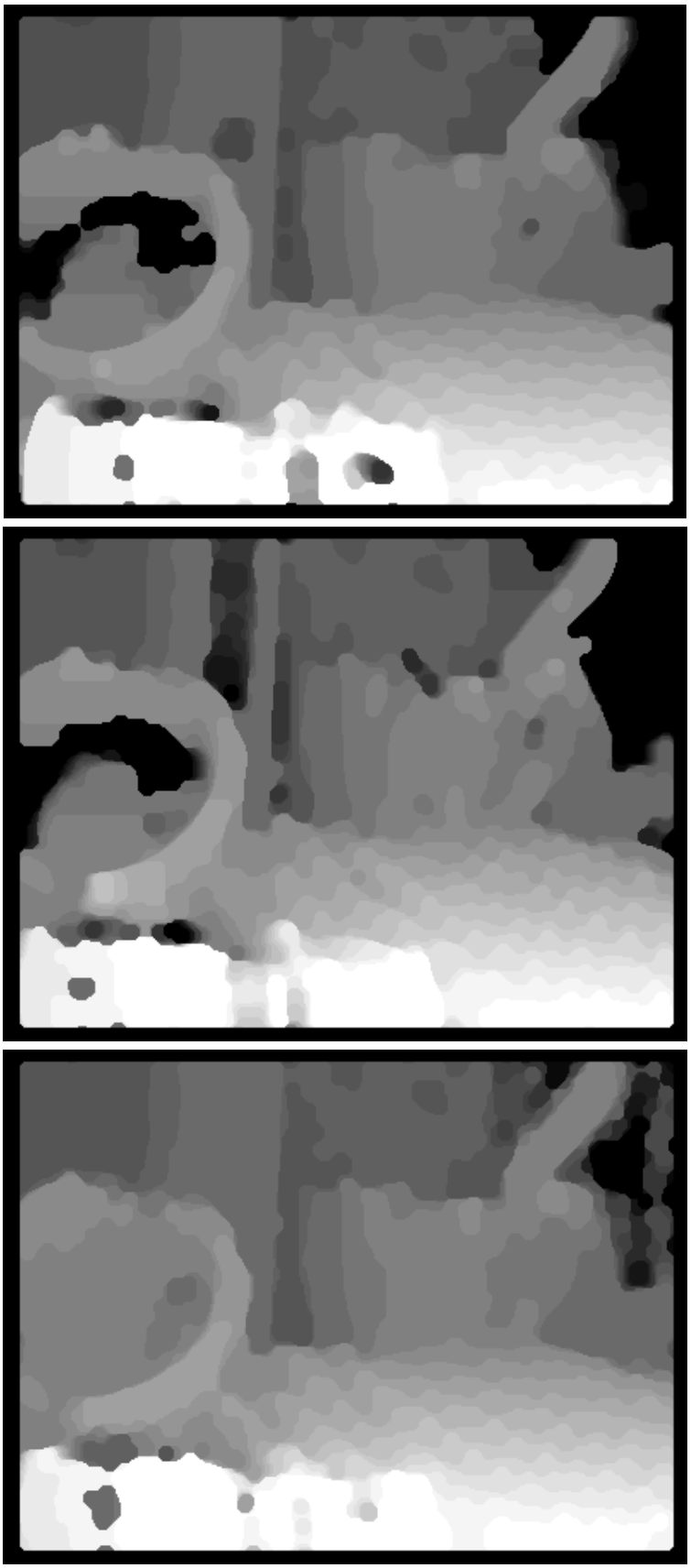
5.3. Wyniki obliczeń

Rysunek 5.5 przedstawia wyniki działania zmodyfikowanego algorytmu. Jako dane wejściowe do obliczeń posłużyły zdjęcia przedstawione na rysunku 5.3. Powiększony obraz odniesienia jest widoczny na rysunku 5.4. Dla porównania na tych samych danych wejściowych, wykonane zostały obliczenia w klasycznym układzie stereowizyjnym (dwie kamery). Obrazem odniesienia był za każdym razem obraz ze środkowej kamery. Dwa pierwsze obrazy przedstawione na rysunku 5.5 pokazują wyniki dla klasycznego układu stereowizyjnego odpowiednio dla kamer: lewej i środkowej oraz środkowej i prawej. Trzeci obraz z rysunku 5.5 przedstawia wyniki dla zmodyfikowanego układu z trzema kamerami.

Można dostrzec istotną poprawę jakości mapy głębi dla układu z dodatkową kamerą. Poprawa widoczna jest także w obszarach o jednolitym kolorze, które ze względy na specyfikę algorytmu stereowizyjnego są dla niego szczególnie wymagające. Wersja algorytmu dla dwóch kamer często nie radzi sobie z takimi obszarami, błędnie wyznaczając dla nich zerową dysparycję. Dla wszystkich trzech obrazów z rysunku 5.5 można zaobserwować zniekształcenia poziomych krawędzi obserwowanych obiektów.

Rysunek 5.3: Przykładowe obrazy wejściowe dla układy z trzema kamerami

Rysunek 5.4: Obraz odniesienia z rysunku 5.3



Rysunek 5.5: Wyniki obliczeń dla obrazów wejściowych z rysunku 5.3

5.4. Pięć kamer

Rozbudowanie układu stereowizyjnego o dodatkową kamerę dało widoczną poprawę w jakości map głębi, istotnie redukując problem przesłonięć w obserwowanej scenie. W celu dalszej poprawy ich jakości, pozostaje do rozwiązania problem degradacji poziomych krawędzi obiektów. Degradacja ta jest spowodowana faktem przemieszczania kamer tylko w płaszczyźnie poziomej. Fragmenty obrazów z kamer przesuwanych w poziomie, ukazujące fragmenty tej samej poziomej krawędzi obiektu, niewiele się od siebie różnią. Powoduje to generowanie błędnych wyników, w dużej mierze zależnych od tła na jakim widziana jest dana krawędź. Analogiczny problem występowałby, dla kamer rozmieszczonych w pionie. W takim przepadku degradacji podlegałyby krawędzie pionowe, a poziome byłby poprawnie odwzorowane. W ten sposób dochodzimy do koncepcji połączenia obu tych rozwiązań w jeden układ zawierający pięć kamer jak na rysunku 5.6. W takim układzie można spodziewać się poprawnego odwzorowania pionowych i poziomych krawędzi oraz dalszej redukcji problemu przesłonięć.

Do akwizycji obrazów w takim układzie zostało wykorzystane pięć kamer internetowych ESPERANZA EC-104. Wykonany został także specjalny statyw pozwalający na ustalenie ich w odpowiedniej pozycji, całość widoczna jest na rysunku 5.6. Maksymalna rozdzielczość uzyskiwanych obrazów to 400x600 pikseli.

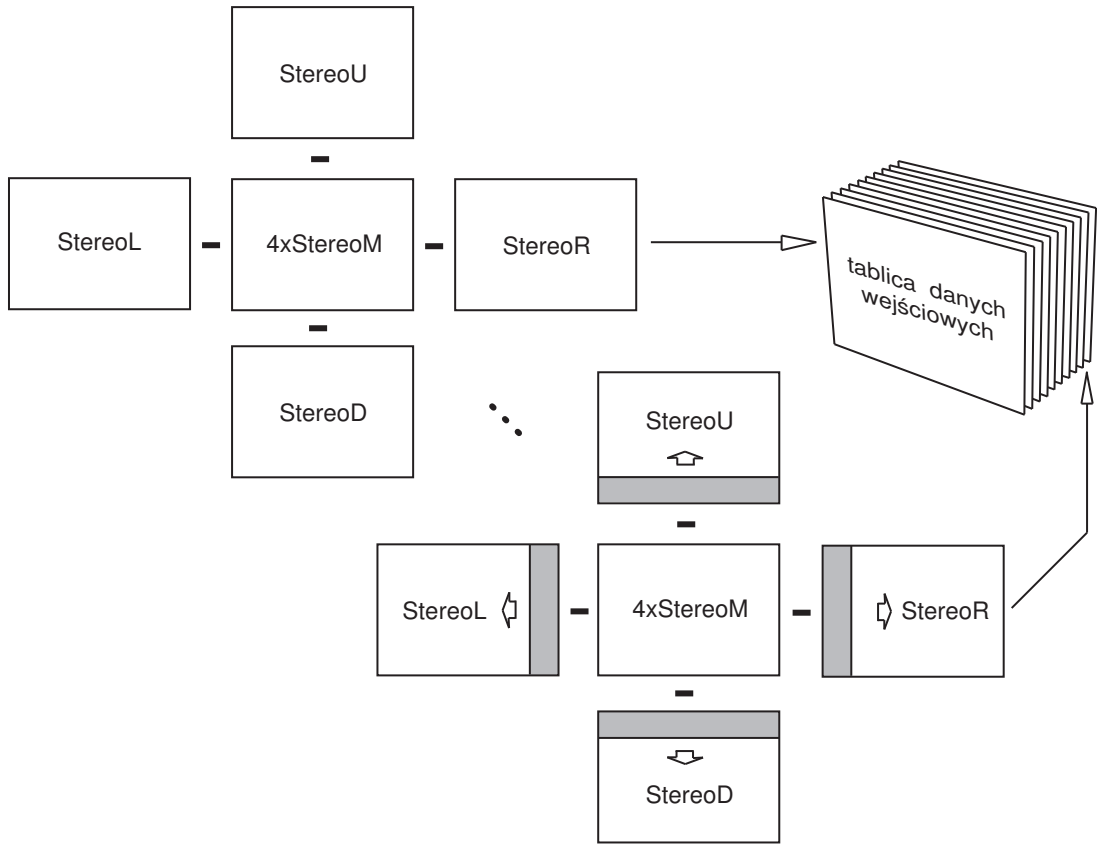
Rysunek 5.6: Statyw wraz z kamerami w układzie krzyżowym

5.5. Modyfikacje algorytmu

Dodanie kolejnych kamer wymaga analogicznych zmian w algorytmie jak w przypadku trzeciej kamery. Tym razem obraz odniesienia mnożony jest razy cztery, a obrazy z kamer pomocniczych przy każdym odejmowaniu przesuwane są w czterech kierunkach zgodnie z rysunkiem 5.7.

Zmodyfikowany kod etapu pierwszego:

```
etap1 = zeros(h,w,3,k,'int32');     //inicjacja tablicy zerami
for d = 1:dmax
   etap1(:,:,:,d) = circshift(StereoR,d-1) + circshift(StereoU,d-1);
end
for d = 1:dmax
   etap1(:,:,:,d) = etap1(:,:,:,d) + circshift(StereoL,-d+1);
end
for d = 1:dmax
   etap1(:,:,:,d) = etap1(:,:,:,d) + circshift(StereoD,-d+1);
end
for d = 1:dmax
   etap1(:,:,:,d) = etap1(:,:,:,d) - 4*StereoM;
end
etap1 = abs(etap1);     //wartość bezwzględna
```

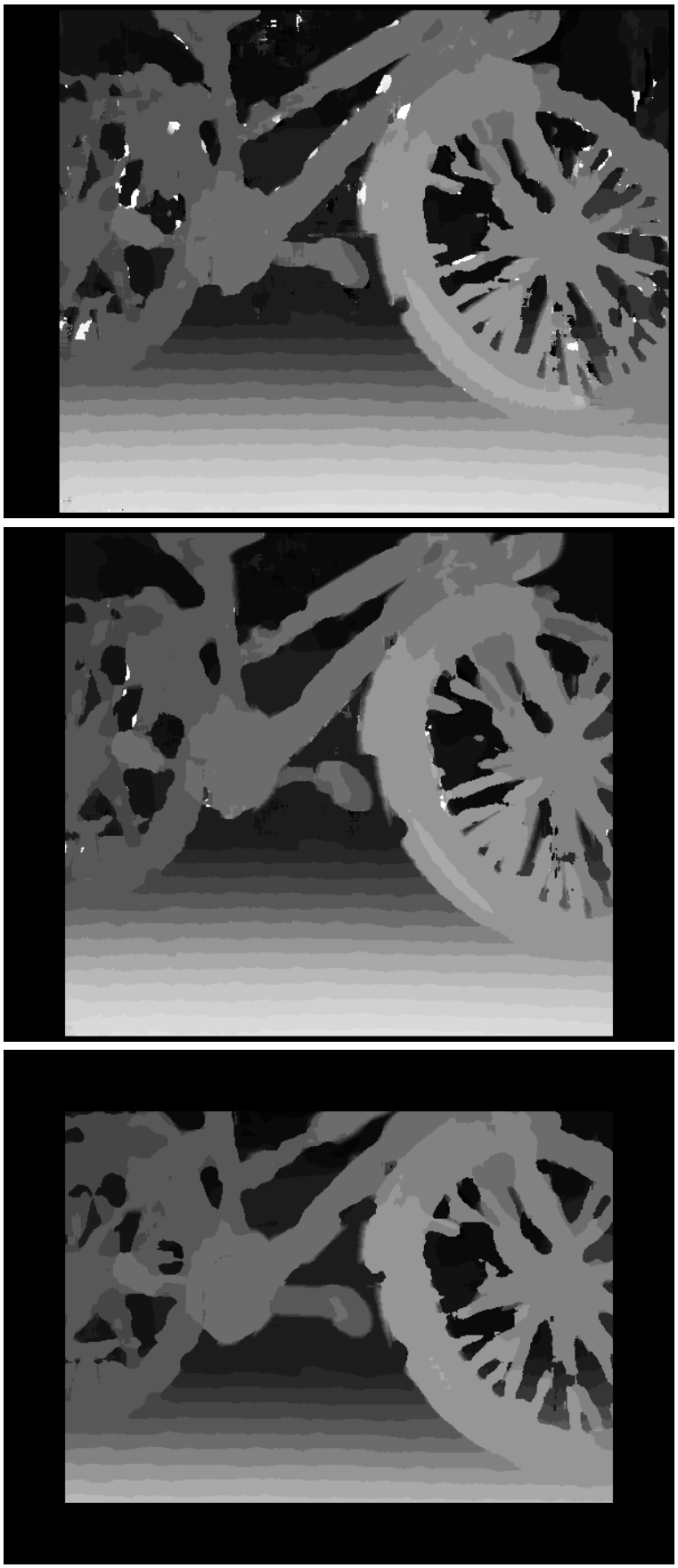


Rysunek 5.7: Etap 1 algorytmu dla układu z pięcioma kamerami

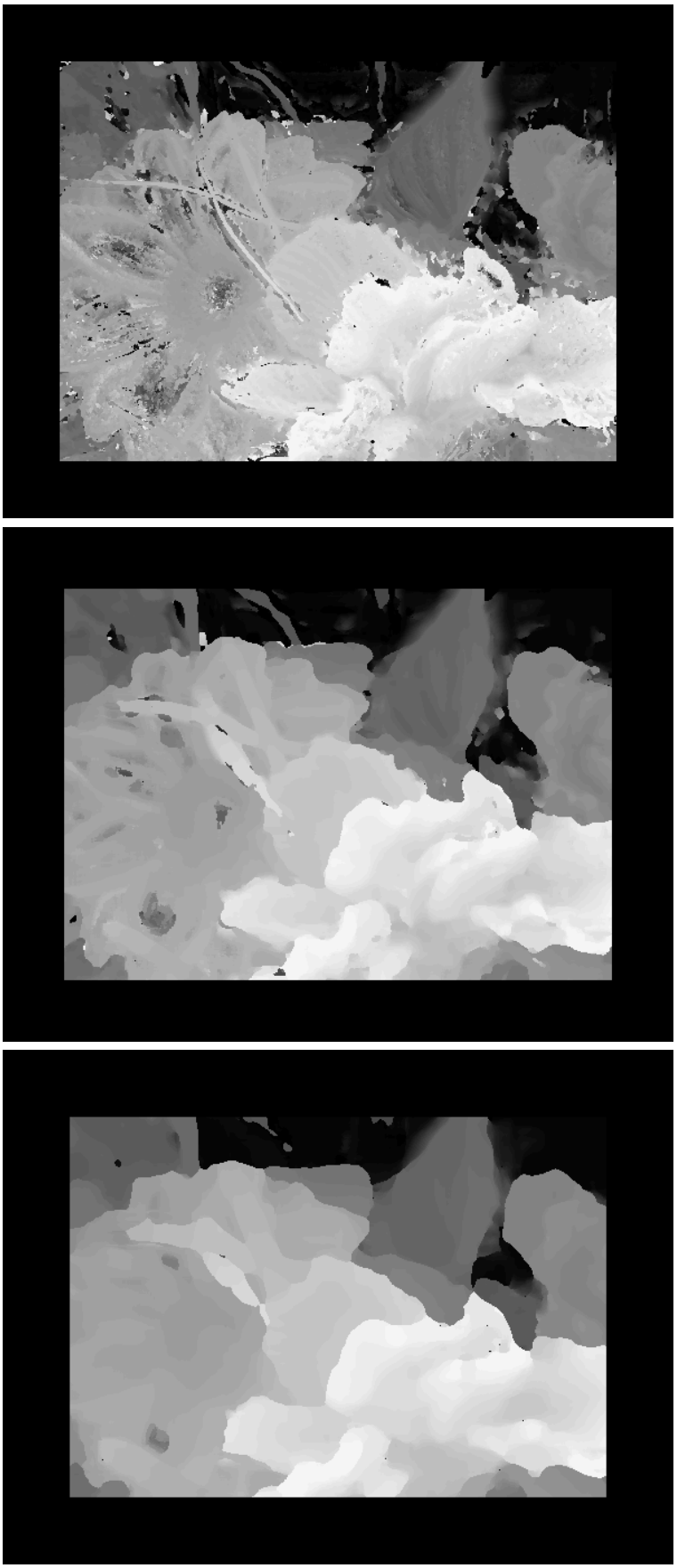
5.6. Wyniki obliczeń

Rysunek 5.9 oraz 5.10 przedstawiają wyniki obliczeń dla zmodyfikowanego układu stereowizyjnego z pięcioma kamerami. Rysunek 5.8 prezentuje obrazy odniesienia odpowiednio dla pierwszej i drugiej serii obliczeń. W pierwszej serii, na tych samych danych wykonano obliczenia dla klasycznego układu stereowizyjnego (górny obraz z rysunku 5.9), oraz dla układu z trzema i pięcioma kamerami (środkowy i dolny obraz z rysunku 5.9). Widoczna jest stopniowa poprawa jakości mapy głębi wraz ze zwiększeniem liczby kamer. Dodatkowo na rysunku 5.9 zobrazowano zmiany wielkości „martwego pola", czyli obszaru obrazu odniesienia dla którego niemożliwe jest wyznaczenie poprawnych wartości dysparycji. Obszar ten jest bezpośrednio zależny od maksymalnego oczekiwanego przesunięcia (d_{max}) oraz w mniejszym stopniu od wielkości otoczenia analizowanego punktu. Na rysunku 5.10 pokazane są wyniki obliczeń w układzie z pięcioma kamerami dla otoczenia o wielkości 3x3, 11x11 oraz 21x21 pikseli. Nie widać znaczących różnic w odwzorowaniu pionowych i poziomych krawędzi.

Rysunek 5.8: Obrazy odniesienia dla obu serii obliczeń



Rysunek 5.9: Wyniki pierwszej serii obliczeń

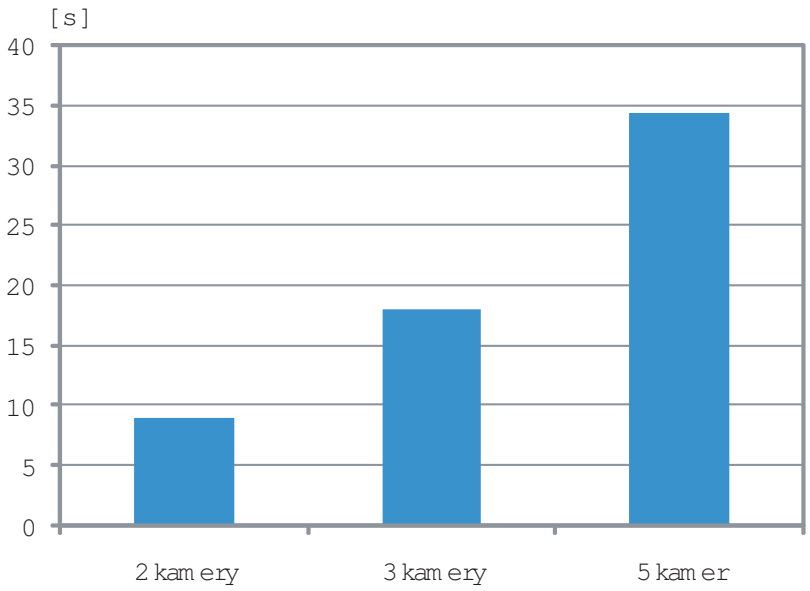


Rysunek 5.10: Wyniki drugiej serii obliczeń

6. Optymalizacja czasu obliczeń

6.1. Optymalizacja kodu Matlab-a

Rozbudowanie układu stereowizyjnego o dodatkowe kamery pozytywnie wpływa na jakość generowanych map głębi, ale jednocześnie znacząco rośnie złożoność obliczeniowa algorytmu. Czas konieczny do przeprowadzenia wszystkich niezbędnych obliczeń powinien być jak najkrótszy, aby cały system był użyteczny. W ramach tej pracy wszystkie obliczenia były wykonywane na komputerze klasy PC z procesorem Intel Celeron 2.8 GHz oraz 3 GB pamięci operacyjnej RAM. Czas potrzebny do przetworzenia jednej ramki obrazu stereowizyjnego o rozmiarze 600x460 pikseli, przy wielkości otoczenia 11x11 i maksymalnym przesunięciu na obrazie 30, wynosił około dziewięciu sekund. Dla większej ilości kamer czas ten jest odpowiednio dłuższy, co obrazuje rysunek 6.1. Tak długi czas przetwarzania jest niekorzystny z punktu widzenia praktycznego zastosowania stereowizji w rzeczywistych układach. Konieczne jest przyspieszenie działania algorytmu, aby umożliwić wyznaczanie przynajmniej jednej klatki obrazu na sekundę. Osiągnięcie tego celu wymaga zastosowania różnych metod, jedną z nich jest optymalizacja kodu pod kątem szybkości jego wykonywania. Niejednokrotnie tą samą funkcjonalność można zaimplementować w różny sposób a mniejsza ilość instrukcji nie zawsze wykonuje się krócej.



Rysunek 6.1: Zależność czasu obliczeń od ilości kamer w układzie

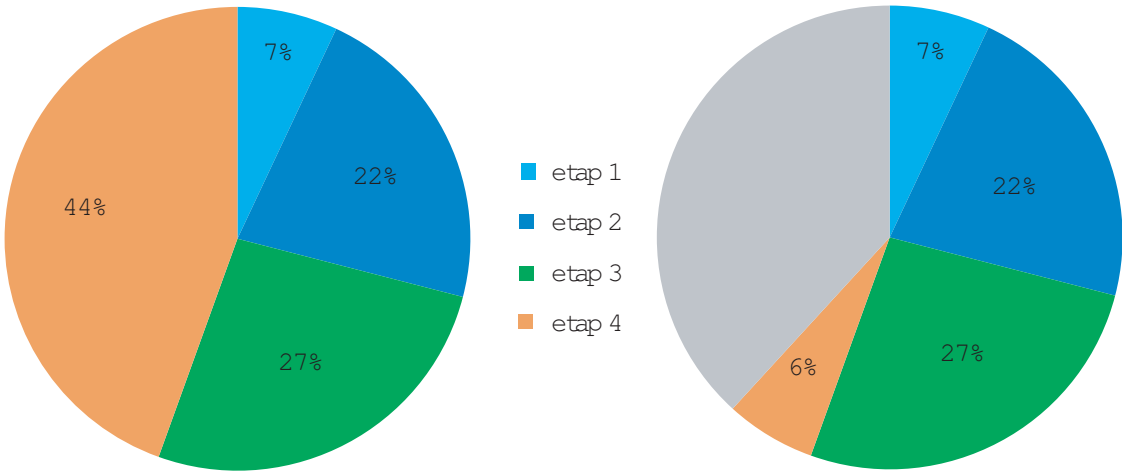
Weźmy na przykład poniższy fragment kody pochodzący z czwartego etapu algorytmu. Wyznacza on wartość dysparycji dla każdego punktu obrazu. Dwie zagnieżdżone pętle wywołują pojedynczą funkcję operującą na małym fragmencie tablicy. Taki zapis jest łatwy do zrozumienia, ale wielokrotne wywoływanie tej samej funkcji i konieczność obsługi indeksacji dwóch pętli sprawia, że jego wykonanie zajmuje niepotrzebnie dużo czasu.

```
for x = 1:w
    for y = 1:h
        [min1, idx1] = min(etap3(x,y,:));
        etap4(x,y) = idx1;
    end
end
```

Powyższy fragment kodu można zastąpić następującym:

```
mini = min(etap3,[],3);
for d = 1:dmax
    etap3(:,:,d) = etap3(:,:,d) + etap4;
    etap4 = etap4 + ((etap3(:,:,d) == mini)*d);
end
```

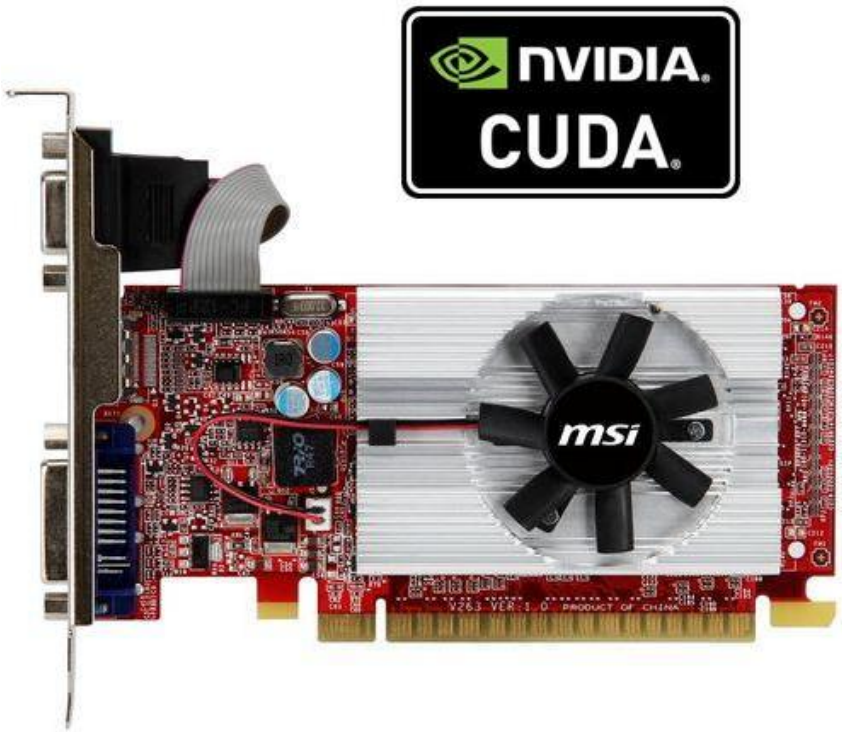
Efekt ich działania jest dokładnie taki sam, ale ten drugi wykona się w czasie blisko dziesięciokrotnie krótszym niż ten pierwszy. Powodem tego jest wykorzystanie funkcji działających od razu na całych tablicach danych a nie na ich pojedynczych elementach oraz ograniczenie indeksowania pętli do niezbędnego minimum. Takie podejście pozwala istotnie zmniejszyć czas obliczeń i sprawia, że moc obliczeniowa komputera jest wykorzystywana bardziej wydajnie. Rysunek 6.2 przedstawia procentowy udział każdego etapu algorytmu przed i po optymalizacji kodu.



Rysunek 6.2: Rozkład czasu obliczeń przed i po optymalizacji kodu

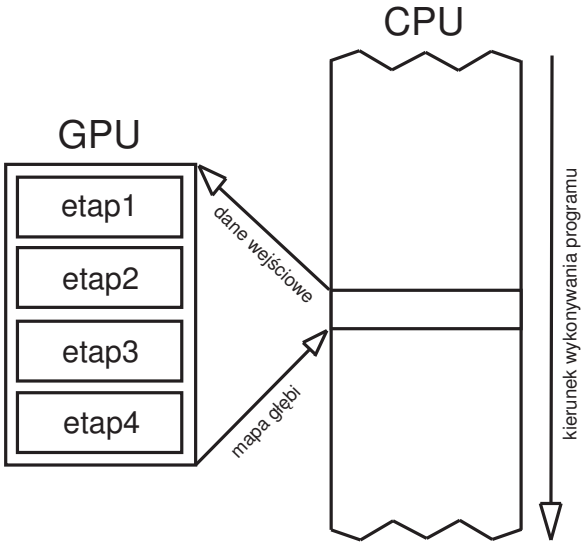
6.2. Obliczenia równoległe z użyciem karty graficznej

Pisanie szybkich algorytmów wymaga szczególnych umiejętności i głębokiej wiedzy o środowisku w jakim się pracuje. Więcej informacji na ten temat dotyczących Matlab-a można znaleźć w [4] oraz jego dokumentacji technicznej. Niezależnie od samej aplikacji, szybkość wykonywania programu jest ograniczona przez możliwości sprzętu, na którym dokonujemy obliczeń. Jednordzeniowy procesor komputera może wykonywać tylko jedno zadanie na raz, przez co nasza aplikacja musi czekać na przedzielenie jej zasobów. Rozwiązaniem tego problemu jest wykonywanie obliczeń w sposób równoległy z wykorzystaniem karty graficznej. Karta graficzna z racji zadania jakie pełni w systemie komputerowym, już od samego początku projektowana jest do pracy wielowątkowej. Przetwarzanie obrazów wymaga dużej ilości prostych, powtarzalnych operacji na dużych tablicach danych. Ilość niezależnych rdzeni w jakie może być wyposażona karta graficzna waha się od kilkudziesięciu do nawet kilkuset. Jednak nie każdej karcie graficznej można zlecić wykonanie obliczeń, niezwiązanych bezpośrednio z wyświetlaniem obrazu na monitorze. Chipset karty musi być wykonany w architekturze wspierającej takie zastosowania. Jedną z takich architektur jest CUDA (ang. Compute Unified Device Architecture) rozwijana przez firmę NVIDIA Corporation. W ramach tej pracy do obliczeń wykorzystywana była karta GIGABYTE GT520 (rysunek 6.3). Karta ta jest wyposażona w 48-rdzeniowy chipset NVIDI-i oraz 1GB pamięci operacyjnej RAM. Komunikacja z płytą główną komputera odbywa się poprzez złącze PCI-express (ang. Peripheral Component Interconnect Express).



Rysunek 6.3: Karta graficzna GIGABYTE GT520

Wykonywanie programu w sposób równoległy różni się od jego tradycyjnego sekwencyjnego wywołania. Do momentu przygotowania wszystkich niezbędnych danych program wykonuje się sekwencyjnie na CPU (procesorze). Następnie dane przesyłane są na GPU (kartę graficzną) wraz z zestawem instrukcji do wykonania w postaci tzw. kernela. Po wykonaniu wszystkich obliczeń na karcie graficznej dane wynikowe przesyłane są z powrotem na CPU gdzie wykonuje się reszta programu (obsługa interfejsu graficznego itp.). Graficznie przedstawia to rysunek 6.4.



Rysunek 6.4: Przebieg wykonywania programu z udziałem GPU

Wewnątrz karty graficznej dane otrzymane z CPU dzielone są na bloki zawierające maksymalnie 1024 elementy. Każdy taki blok może jest wykonywany na jednym rdzeniu chipsetu karty graficznej, czyli jednocześnie przetwarzane jest 48 bloków danych. Kolejność przetwarzania bloków może być dowolna i zależy od wewnętrznej organizacji pamięci karty graficznej. W przypadku obrazów wejściowych o rozdzielczości 600x460 pikseli przyjęto rozmiar jednego bloku na 30x23 piksele (czyli każdy blok zawiera 690 elementów). Łącznie do przetworzenia będzie wtedy 400 bloków danych.

Każdy etap algorytmu stereowizyjnego wymaga wiedzy o różnych fragmentach obrazu, dlatego niemożliwe było napisanie kernela dla karty graficznej który wykonywałby wszystkie etapy jednocześnie. Konieczne było napisanie osobnych kerneli dla każdego etapu algorytmu. Wykonanie kodu kolejnego kernela rozpoczyna się dopiero po przetworzeniu wszystkich bloków danych z poprzedniego. W ten sposób uzyskana została synchronizacja niezbędna do uzyskania poprawnych wyników (map głębi). Pełne kody źródłowe kerneli dla każdego etapu algorytmu można znaleźć w dodatku B. Dodatkowe informacje o równoległym wykonywaniu obliczeń z użyciem pakietu Matlab można znaleźć w [6].

6.3. Porównanie wyników

Dla porównania skuteczności przedstawionych metod przyspieszania wyznaczania map głębi, przeprowadzone zostały obliczenia porównawcze na takim samy zestawie danych. Wszystkie wersje aplikacji wyznaczały mapę głębi dla stereopary o rozmiarach 600x460 pikseli, wielkości otoczenia 11x11 i maksymalnym przesunięciu na obrazie równym 30 pikseli. Wyniki zestawione są w tabeli 6.1. Dla układów z większą ilością kamer czasy te będą oczywiście proporcjonalnie dłuższe.

Wersja aplikacji	czas obliczeń [s]
standardowy kod Matlab-a (CPU)	15,2
zoptymalizowany kod Matlab-a (CPU)	9,1
obliczenia równoległe (GPU)	0,6

Tablica 6.1: Porównanie czasów obliczeń dla różnych wersji aplikacji

Uzyskane poprawa szybkości obliczeń jest zadowalająca i w pełni rekompensuje wysiłek jaki włożono w optymalizację kodu algorytmu oraz dostosowanie go do wykonywania w sposób równoległy. Dalsze rozwijanie tego podejścia, wraz z zastosowaniem nowych bardziej wydajnych kart graficznych, pozwoliłoby z pewnością na wyznaczanie gęstych map głębi w czasie zbliżonym do rzeczywistego.

7. Podsumowanie

Stereowizja jest bardzo ciekawą i obiecującą metodą rozpoznawania położenia obiektów w przestrzeni. Nie jest to jednak metoda pozbawiona wad. W ramach tej pracy przeprowadzona została analiza problemów związanych ze specyfiką algorytmu seterowizyjnego oraz możliwości poprawy jakości generowanych map głębi. Modyfikacje samego algorytmu pozwalają na odrzucenie części błędów pojawiających się w mapach głębi, jednak ich źródło znajduje się już w samej idei stereowizji dwukamerowej. Jedynym skutecznym sposobem wyeliminowania tych błędów jest zwiększenie liczby kamer w układzie do trzech. Uzyskuje się przez to dodatkową informację o obserwowanej scenie. Ta dodatkowa informacja, pozwala na prawidłowe wyznaczenie odległości obserwowanych obiektów w obszarach niedostępnych dla klasycznej stereowizji dwukamerowej. Zwiększenie liczby kamer do pięciu i ustawienie ich w układzie krzyżowym pozwala na dalszą redukcję błędnych wyników oraz na poprawę odwzorowania poziomych krawędzi, które były zniekształcane ze względu na przesuwanie kamer tylko w płaszczyźnie poziomej.

Zwiększanie ilości kamer przyczynia się do poprawy jakości map głębi, ale jednocześnie zwiększa złożoność obliczeniową algorytmu. Większa ilość obliczeń, powoduje nieuchronnie wydłużenie czasu jaki jest na nie potrzebny. Zbyt długi czas obliczeń znacznie ogranicza możliwości praktycznego zastosowania stereowizji, dlatego w ostatnim rozdziale pracy przedstawiono metody pozwalające przyspieszyć wyznaczanie map głębi. Na szczególną uwagę zasługuje tu metoda obliczeń równoległych z zastosowaniem karty graficznej, dzięki której możliwe jest wyznaczanie map głębi w czasie poniżej sekundy.

Idea stereowizji oparta jest na funkcjonowaniu ludzkiego wzroku. Dzięki temu posiada wiele jego zalet, ale dziedziczy również jego wady. Problemy mogą wystąpić we wszystkich sytuacjach, w których człowiek także może mieć problem z rozpoznaniem rzeczywistej odległości od obiektów. Dotyczy to szczególnie przedmiotów całkowicie przeźroczystych (np szyby), jak i powierzchni odbijających światło (np lustro). Z tych fundamentalnych powodów, układ stereowizyjny raczej nigdy nie będzie samodzielnym modułem a jedynie uzupełnieniem większego systemu rozpoznawania obrazu.

Bibliografia

[1] J.-Y. Bouguet. *Camera Calibration Toolbox for Matlab*. `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`.

[2] B. Cyganek. *Komputerowe przetwarzanie obrazów trójwymiarowych*. Akademicka Oficyna Wydawnicza EXIT, Warszawa 2002.

[3] L. Di Stefano, M. . Marchionni, and S. Mattoccia. A pc-based real-time stereo vision system. *Machine Graphics & Vision*, vol. 13, no. 3:197–220, 2004.

[4] P. Getreuer. *Writing Fast MATLAB Code*. http://www.ee.columbia.edu/marios/matlab/WritingFastMATLABCode.pdf, 2006.

[5] Y. Mao, S. Soatto, J. Kosecka, and S. S.S. *An invitation to 3-D vision: from images to geometric transformations*. Springer-Verlag, 2004.

[6] MathWorks. *Parallel Computing Toolbox - User's Guide*. http://www.mathworks.com/products/parallel-computing/, 2011.

[7] K. Muhlmann, D. Maier, J. Hesser, and R. Manner. Calculating dense disparity maps from color stereo images, an efficient implementation. *International Journal on Computer Vision*, vol. 47, no. 1-3:79–88, 2003.

[8] P. Pełczynski and P. Strumiłło. *Szybkie wyznaczanie głębi w scenie trójwymiarowej ze stereoskopii*. XIII Konferencja Sieci i Systemy Informatyczne, 2005.

[9] D. Rzeszotarski, P. Strumiłło, P. Pełczynski, B. Wiecek, and A. Lorenc. System obrazowania stereoskopowego sekwencji scen trójwymiarowych. *Elektronika: prace naukowe*, nr 10:165–184, 2005.

[10] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal on Computer Vision*, vol. 47, no. 1-3:7–42, 2002.

Dodatek A

Zawartość płyty CD:
...

Dodatek B

Kody źródłowe kerneli dla poszczególnych etapów algorytmu:

Etap 1:

```
__global__ void etap1(const int* in1,
                const int* in2,
                int* out,
                int height,
                int width,
                int k)
{
   int row = blockIdx.x * blockDim.x + threadIdx.x;
   int col = blockIdx.y * blockDim.y + threadIdx.y;
   int all = height*width;    int allx2 = all*2;
   int allx3 = all*3;   int idx1 = col*height+row;
   int idx2 = idx1+all;
   int idx3 = idx1+allx2;
   for(int d=0; d<k; d++) {
      int dCol = col-d;
      int idxd = dCol*height+row;
      int dxallx3 = d*allx3;
      if(dCol >= 0) {
         out[dxallx3+idx1] = abs(in2[idxd] - in1[idx1]);
         out[dxallx3+idx2] = abs(in2[idxd+all] - in1[idx2]);
         out[dxallx3+idx3] = abs(in2[idxd+allx2] - in1[idx3]);
      }
   }
}
```

Etap 2:

```
__global__ void etap2(const int* in,
                      int* out,
                      int height,
                      int width,
                      int k,
                      int m)
{
   int row = blockIdx.x * blockDim.x + threadIdx.x;
   int col = blockIdx.y * blockDim.y + threadIdx.y;
   int all = height*width;
   int idx = col*height+row;
   int kx3 = k*3;
   for(int d=0; d<kx3; d++) {
      int dxall = d*all;
      int dxallplusrow = dxall+row;
      int didx = dxall+idx;
      for(int i=0; i<m; i++) {
         out[didx] = out[didx] + in[dxallplusrow+(col+i)*height];
      }
   }
}
```

Etap 3:

```
__global__ void etap3(const int* in,
                      int* out,
                      int height,
                      int width,
                      int k,
                      int m)
{
   int row = blockIdx.x * blockDim.x + threadIdx.x;
   int col = blockIdx.y * blockDim.y + threadIdx.y;
   int all = height*width;    int allx2 = all*2;
   int allx3 = all*3;   int idx = col*height+row;
   for(int d=0; d<k; d++) {
      int didx = d*all+idx;
      int dall3idx = d*allx3+idx;
      for(int i=0; i<m; i++) {
         int dall3idxi = dall3idx+i;
         out[didx] = out[didx] + in[dall3idxi]+in[dall3idxi+all] +
                 + in[dall3idxi+allx2];
      }
   }
}
```

Etap 4:

```
__global__ void etap4(const int* in,
                      int* out,
                      int height,
                      int width,
                      int k)
{
   int row = blockIdx.x * blockDim.x + threadIdx.x;
   int col = blockIdx.y * blockDim.y + threadIdx.y;
   int all = height*width;
   int idx = col*height+row;
   int mini = in[idx];
   out[idx] = 0;
   for(int d=1; d<k; d++) {
      int didx = d*all+idx;
      if(in[didx] < mini) {
         mini = in[didx];
         out[idx] = d;
      }
   }
}
```