AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

Praca dyplomowa

magisterska

System do estymacji mapy głębokości na podstawie stereoskopowych sekwencji wideo dla platformy Android

Imię i nazwisko *Konrad Kurzawski*
Kierunek studiów *Elektronika i Telekomunikacja*
Opiekun pracy *dr inż. Jarosław Bułat*

Kraków, rok 2013

OŚWIADCZENIE AUTORA

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

……..
(Podpis autora)

Spis treści

Wstęp

W niniejszej pracy przedstawiono system do rekonstrukcji przestrzennych struktur na podstawie obrazu pochodzącego za dwóch kamer cyfrowych. System zaimplementowano na platformie Android.

W ostatnich latach, dzięki wzrostowi mocy obliczeniowej procesorów wykorzystujących architekturę ARM pojawia się co raz więcej przykładów wykorzystania cyfrowego przetwarzania obrazów w takich dziedzinach jak robotyka, rozrywka multimedialna czy użytkowe aplikacje na współczesne telefony komórkowe. Rekonstrukcja 3D jest dynamicznie rozwijającą się dziedziną, która znajduje zastosowanie w systemach monitoringu, odtwarzaniu rzeźby terenu na podstawie zdjęć lotniczych czy współczesnych grach konsolowych. Istnieje szereg technik uzyskiwania informacji o przestrzennych strukturach, począwszy od rozwiązań programowych, wykorzystujących co najmniej dwie kamery cyfrowe, aż po implementacje sprzętowe korzystające z właściwości światła strukturalnego bądź systemy TOF (ang. *Time of Flight*).

W pracy dokonano przeglądu i analizy współcześnie stosowanych metod rekonstrukcji 3D, następnie zaimplementowano system stereowizyjny zbudowany z wykorzystaniem średniej klasy kamer internetowych. Dzięki udostępnieniu interfejsu programistycznego przez producenta Xbox Kinect™, chmury punktów uzyskane z tego urządzenia mogły zostać porównane z wynikami otrzymanymi z układu kamer. Ważnym elementem badań była analiza i optymalizacja szybkości działania zbudowanego systemu oraz przeniesienie go na platformę Android w celu zbadania możliwości wykorzystania w systemie wspierającym nawigację osób niewidomych lub ociemniałych. Pokazano metody podtrzymywania kalibracji kamer oraz wskazano konieczność adaptacji do bieżących warunków oświetlenia sceny.

Prezentowana praca składa się z części teoretycznej i części praktycznej. W pierwszym rozdziale wprowadzono model matematyczny układu kamer. Zostały podane zależności pozwalające obliczyć przestrzenne położenie punktów na podstawie dwóch różnych ujęć tej samej sceny. Następnie zaprezentowano metody podtrzymywania kalibracji układu stereowizyjnego oraz przedstawiono autorski algorytm służący do regulacji wybranych parametrów wewnętrznych kamer na podstawie histogramów

obrazów. Trzeci podrozdział pierwszej części pracy zawiera porównanie trzech najpopularniejszych technik rekonstrukcji 3D.

Część praktyczna została podzielona na cztery części. W pierwszej przedstawiono architekturę systemu. Zaprezentowano rozwiązania programistyczne umożliwiające stworzenie aplikacji w języku C++ z użyciem biblioteki OpenCV w środowisku Linux, której szkielet posłużył do opracowania systemu na platformę Android. Drugi podrozdział zawiera opis najważniejszych elementów modułu odpowiedzialnego za podtrzymywanie kalibracji kamer. W trzecim podrozdziale pokazano techniki umożliwiające uzyskanie chmury punktów z urządzenia Kinect oraz obrazów pochodzących z pary kamer. Ostatnia część tego rozdziału zawiera szczegółowy opis komponentów systemu dla platformy Android wraz z uzasadnieniem wyboru modułów sprzętowych i technik programistycznych.

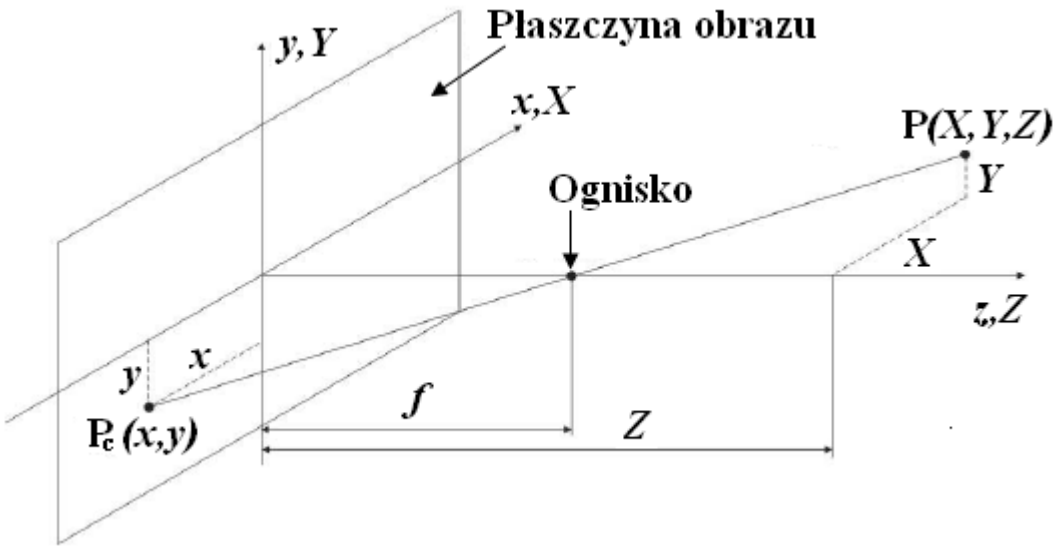
Ostatni rozdział przedstawia wyniki przeprowadzonych badań wraz z analizą dokładności i szybkości działania systemu. W podrozdziale 3.1 pokazano, że nie jest możliwe podtrzymywanie kalibracji kamer bez użycia obiektu wzorcowego, bazując tylko i wyłącznie na sekwencjach obrazu. W kolejnej części zostały zaprezentowane mapy głębokości scen dla scenariuszy wskazanych przez osobę niewidomą jak potencjalne zastosowania systemu. Ostatni podrozdział zawiera analizę złożoności obliczeniowej dla platformy Android. Wskazano najmniej wydajne moduły oraz zaproponowano metody optymalizacji ich działania.

1 Rekonstrukcja powierzchni 3D w systemach stereowizyjnych

W pierwszym rozdziale przedstawiono matematyczny model systemu stereowizyjnego zbudowanego z dwóch kamer cyfrowych. Został wykonany przegląd i analiza najpopularniejszych metod obecnie stosowanych w rekonstrukcji 3D. Pierwszy podrozdział opisuje zależności geometryczne w układzie kamer, zniekształcenia toru optycznego oraz metody kalibracji z użyciem obiektów wzorcowych. W drugim podrozdziale zaprezentowano zakłócenia obrazu wynikające ze sposobu działania kamer cyfrowych oraz budowy systemu stereowizyjnego przeznaczonego do użycia jako okulary wspomagające poruszanie się osoby niewidomej. Zostały przedstawione metody zapobiegania oraz kompensacji tych zakłóceń. Zaprezentowano przykład techniki podtrzymywania kalibracji kamer bez użycia obiektu wzorcowego. Trzeci podrozdział opisuje sposoby uzyskiwania informacji o przestrzennym położeniu punktu na podstawie pary obrazów pochodzących ze skalibrowanych kamer oraz z użyciem kamery TOF i urządzenia Kinect.

1.1 Modele układów kamer

Wyróżnia się dwa podstawowe modele geometryczne kamery cyfrowej[18]. Pierwszym z nich jest model perspektywiczny, który dokładnie odzwierciedla przekształcenia geometryczne wprowadzane przez tor kamery, nie jest on stosowany często w obliczeniach numerycznych, ze względu na złożoność implementacyjną. Wprowadzenie pewnych uproszczeń nie powodujących utraty ogólnych własności modelu perspektywicznego prowadzi do jego uproszczenia – w praktycznych zastosowaniach modelowania przekształcenia homograficznego kamer cyfrowych stosuje się model otworkowy (*pin-hole*). Na rysunkach 1.1 i 1.2 przedstawiono odpowiednio model perspektywiczny i model otworkowy.



Rys 1.1 Perspektywiczny model kamery[1]

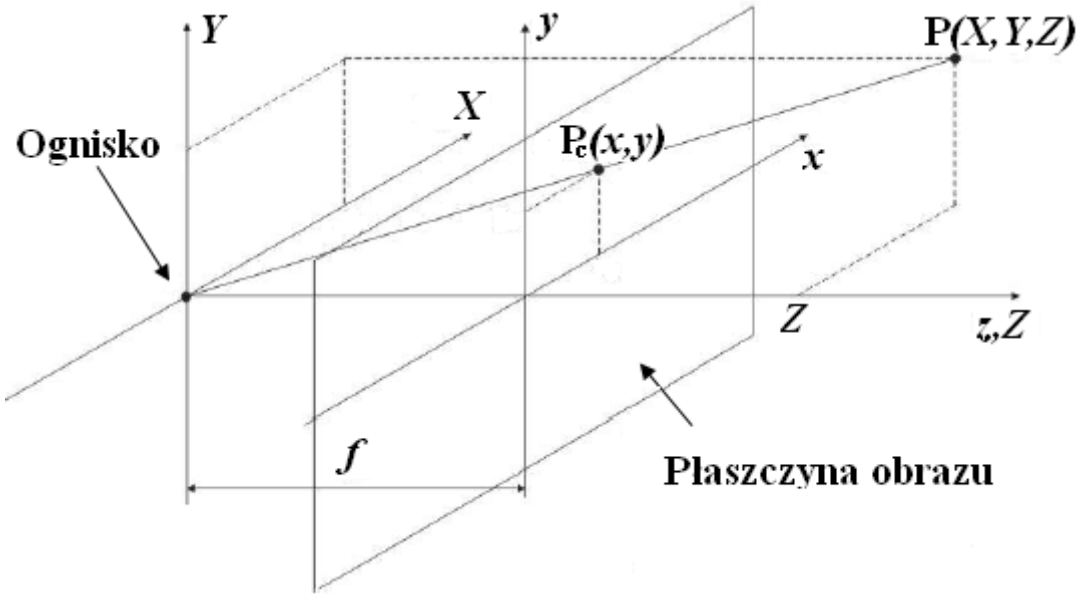
f – ogniskowa kamery,

$\mathrm{P}(X,Y,Z)$ – współrzędne punktu w przestrzeni trójwymiarowej,

$\mathrm{P_c}(x,y)$ – współrzędne punktu na płaszczyźnie obrazu.

W modelu perspektywicznym wprowadzane przekształcenie uwzględnia odwrócenie obrazu o 180 stopni w płaszczyźnie obrazu.

$$\frac{x}{f}=\frac{-X}{Z-f}=\frac{X}{f-Z} \quad , \qquad \frac{y}{f}=\frac{-Y}{Z-f}=\frac{Y}{f-Z} \tag{1.1}$$



Rys. 1.2 Model kamery otworkowej[2]

[1] Rysunek pochodzi z [18], s. 2

Model otworkowy kamery można opisać poniższym równaniem

$$\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x \\ y \\ Z \end{bmatrix} \tag{1.2}$$

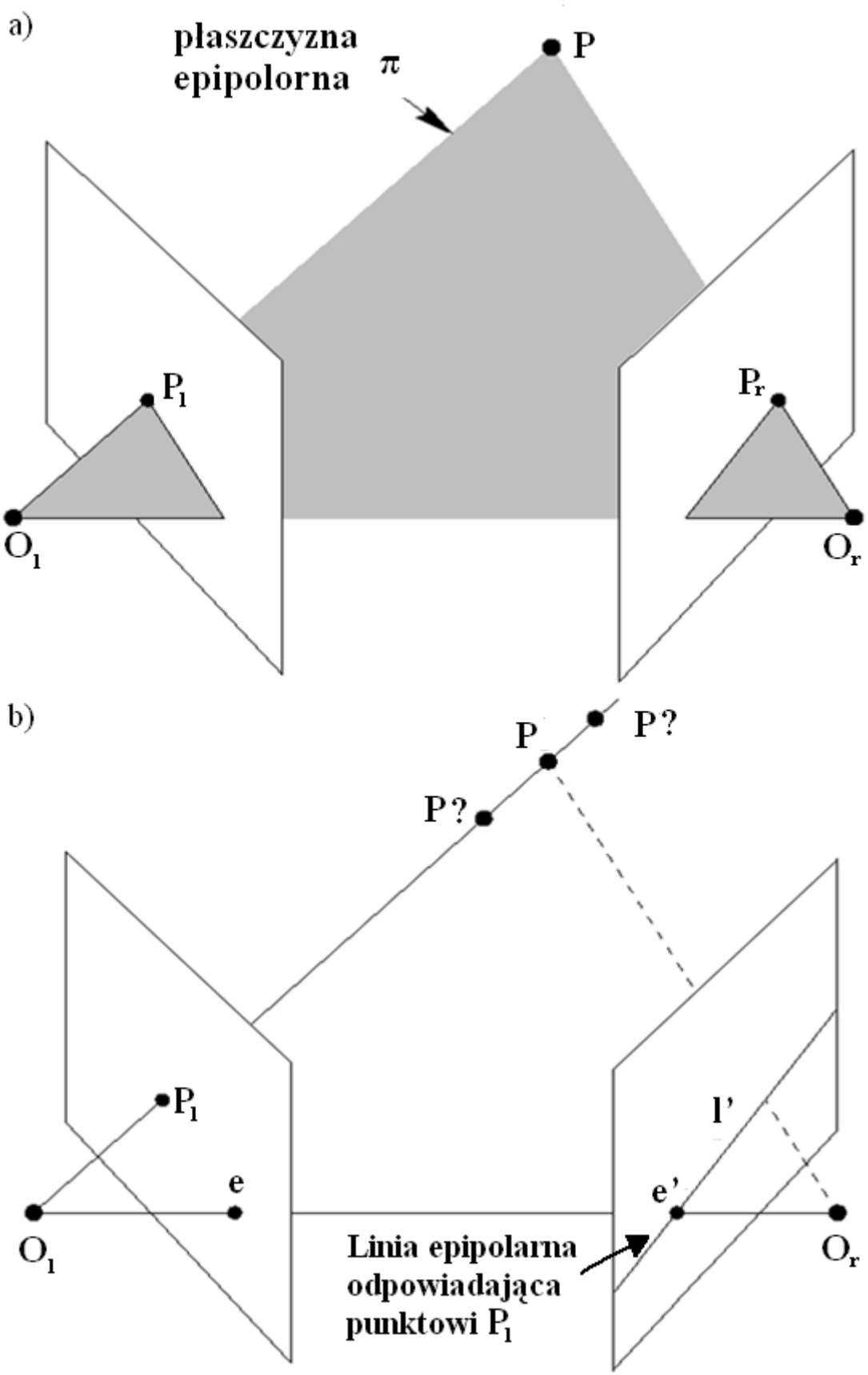
Wprowadzenie niezerowych współrzędnych środka obrazu kamery oraz rozróżnienia długości boku piksela sprowadza (1.2) do następującej postaci:

$$\begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x \\ y \\ Z \end{bmatrix} \tag{1.3}$$

p_x, p_y – współrzędne środka obrazu kamery,
f_x, f_y – ogniskowe kamery w odniesieniu do x i y.

Układ dwóch kamer cyfrowych o znanym wzajemnym położeniu i wyznaczonych parametrach wewnętrznych kamer oraz współczynnikach zniekształceń toru optycznego nazywa się systemem stereowizyjnym. Zasadza działania takiego systemu polega na równoczesnym rejestrowaniu obrazu z dwóch kamer i wyszukiwaniu rzutów tych samych punktów na obrazach lewej i prawej kamery, a następnie korzystając z zależności geometrycznych opisanych w [3], [16] i [19] można obliczyć rzeczywiste współrzędne przestrzenne punktów należących do obserwowanej sceny. Rysunek 1.3 przedstawia uproszczony model geometryczny systemu stereowizyjnego. Płaszczyzna epipolarna wyznaczona przez punkty P , O_l i O_r posłuży do estymacji wzajemnego położenia kamer. Najważniejszą właściwość układu stereowizyjnego zaprezentowano na rysunku 1.3. Znając współrzędne rzutu punktu P na płaszczyznę obrazu lewej kamery (P_l) oraz równanie linii epipolarnej (obliczonej na podstawie rotacji, translacji oraz parametrów wewnętrznych kamer) możliwe jest znalezienie punktu leżącego na linii l' będącego rzutem P na płaszczyznę prawej kamery i na tej podstawie obliczenie położenia przestrzennego P .

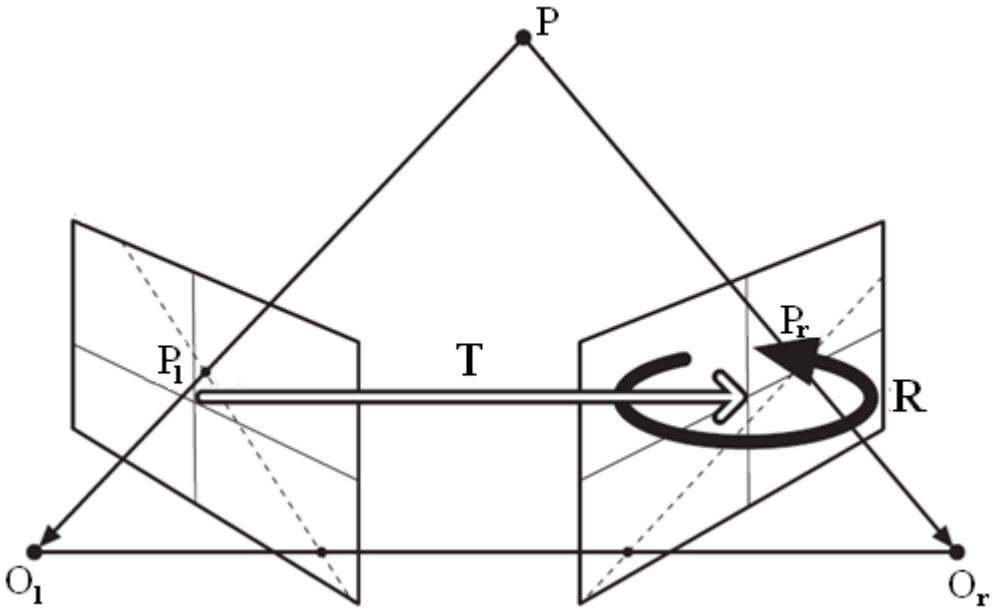
[2] Rysunek pochodzi z [18], s. 8



Rys. 1.3 Układ dwóch kamer - rzut punktu P na płaszczyznę każdej z nich[3]

[3] Rysunek pochodzi z [16], s. 240. Wprowadzono dodatkowe oznaczenia.

W układzie dwóch kamer reprezentowanych modelem otworkowym możliwe jest wyprowadzenie zależności wiążącej współrzędne rzutu tego samego punktu na płaszczyzny obrazów kamer. Pokazana zostanie operacja odwrotna; mając współrzędne punktów na obrazach kamer wyrażone w pikselach oraz macierze wewnętrzne kamer można estymować ich wzajemne położenie. Na rysunku 1.4 przedstawiono uproszczony model układu kamer, który pozwala na zapisanie podstawowych równań wektorowych opisujących system stereowizyjny.



Rys. 1.4 Model geometryczny układ dwóch kamer otworkowych[4]

Korzystając z własności wektora prostopadłego do płaszczyzny:

$$\left(\overline{XA}\right)^T \cdot \bar{N} = 0 \tag{1.4}$$

gdzie:

$\bar{N}$ – wektor prostopadły do płaszczyzny $O_l O_r P$,

X – dowolny punkt płaszczyzny $O_l O_r P$ z którego jest prowadzony wektor prostopadły do niej ($\bar{N}$),

A – dowolny punkt płaszczyzny $O_l O_r P$, różny od X,

$\overline{XA}$ – wektor o początku w punkcie X i końcu w punkcie A,

$(\cdot)$ – iloczyn skalarny.

[4] Rysunek pochodzi z [3], s. 421, wprowadzono dodatkowe oznaczenia

Wektor wodzący z punktu O_l do P_l, oznaczony jako $\overline{O_lP_l}$ oraz wektor $\overline{T}$ należą do płaszczyzny O_lO_rP, a jako wektor prostopadły do niej może posłużyć iloczyn wektorowy $\overline{T} \times \overline{O_lP_l}$. Korzystając z własności (1.4) można zapisać następujące równanie:

$$(\overline{O_lP_l} - \overline{T})^T \cdot (\overline{T} \times \overline{O_lP_l}) = 0 \tag{1.5}$$

Korzystając z własności: $\overline{O_lP_l} = R^T \cdot \overline{O_rP_r} + \overline{T}$, równanie (1.5) przyjmuje postać:

$$(R^T \cdot \overline{O_rP_r})^T \cdot (\overline{T} \times \overline{O_lP_l}) = 0 \tag{1.6}$$

Iloczyn wektorowy $\overline{T} \times \overline{O_lP_l}$ można przedstawić jako iloczyn macierzy i wektora:

$$(\overline{T} \times \overline{O_lP_l}) = S \cdot \overline{O_lP_l} \tag{1.7}$$

gdzie:

$$S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix} \tag{1.8}$$

Wprowadzając macierz E (*essential*): $E = R \cdot S$, równanie (1.6) przyjmuje postać:

$$(\overline{O_rP_r})^T \cdot E \cdot \overline{O_lP_l} = 0 \tag{1.9}$$

Macierz E jest macierzą rzędu drugiego – 9 parametrów zależy od 6 zmiennych (3 od translacji i 3 od rotacji). Macierz E pozwala powiązać położenie przestrzenne punktu P w układach przestrzennych lewej i prawej kamery. Wprowadzając współrzędne homogeniczne punktów w układzie każdej z kamer wyrażone w jednostkach podanych w procesie kalibracji równanie (1.9) można zapisać jako:

$$p_r \cdot E \cdot p_l = 0 \tag{1.10}$$

gdzie:

p_r – homogeniczne ws*p*ółrzędne przestrzenne punktu P w układzie prawej kamery,

p_l – homogeniczne współrzędne przestrzenne punktu P w układzie lewej kamery.

Zależność wiążąca współrzędne homogeniczne punktu na obrazach kamer wyrażone w pikselach (q_r, q_l), $p = M^{-1} \cdot q$, gdzie M to macierz wewnętrzna kamery oraz wprowadzenie macierzy fundamentalnej F zdefiniowanej jako:

$$\mathrm{F} = (\mathrm{M}_\mathrm{r}^{-1})^{\mathrm{T}} \cdot \mathrm{E} \cdot \mathrm{M}_\mathrm{l}^{-1} \tag{1.11}$$

pozwala na zapisanie zależności wiążącej współrzędne (wyrażone w pikselach) tego samego punktu na obrazie lewej i prawej kamery.

$$q_\mathrm{r}^{\mathrm{T}} \cdot \mathrm{F} \cdot q_l = 0 \tag{1.12}$$

Macierz fundamentalna zawiera informacje o wzajemnym położeniu kamer oraz przekształceniach wprowadzanych przez ich tory optyczne, co przy znajomości współrzędnych rzutów punktów na obrazy kamer pozwala na sprawdzenie czy położenie kamer bądź ich parametry wewnętrzne uległy zmianie. Im większa wartość iloczynu $q_\mathrm{r}^{\mathrm{T}} \cdot \mathrm{F} \cdot q_l$, tym większa zmiana nastąpiła w przesunięciu i rotacji lub parametrach wewnętrznych kamer. Przy znanym wzajemnym położeniu kamer (znana jest macierz fundamentalna) oraz współrzędnych punktu na obrazie jednej z kamery, możliwe jest obliczenie położenia tego punktu na obrazie drugiej kamery. Korzystając z (1.12) można obliczyć macierz F znając współrzędne odpowiadających sobie par punktów. Proces estymacji macierzy R i wektora $\overline{\mathrm{T}}$ układu stereowizyjnego na podstawie znajomości macierzy fundamentalnej i parametrów wewnętrznych kamer został przedstawiony w [2].

Obliczenie macierzy E na podstawie F oraz M_r i M_l jest trywialne, natomiast dekompozycja macierzy E na składowe R i S wymaga kilku kroków i nie daje jednoznacznego rozwiązania. Dowolna macierz liczb rzeczywistych o wymiarach 3x3 może być macierzą E (*essential*), wtedy i tylko wtedy gdy jej dekompozycja wg wartości osobliwych przyjmuje następującą postać:

$$\mathrm{E} = \mathrm{U} \cdot \mathrm{diag}(r, s, 0) \cdot \mathrm{V}^{\mathrm{T}} \tag{1.13}$$

gdzie:

$\mathrm{diag}(r, s, 0)$ – macierz diagonalna o wartościach osobliwych równych: r, s i 0 taka, że $\mathrm{r} \geq \mathrm{s}$,

U , V – macierze ortonormalne.

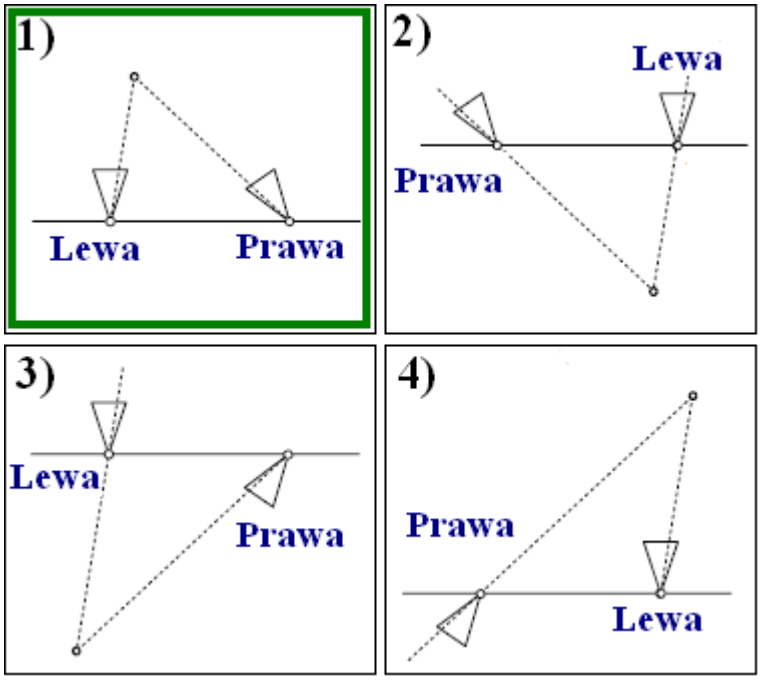
W tabeli 1.1 przedstawiono wszystkie możliwe rozwiązania, z których tylko jedno jest właściwe dla danego układu kamer. Dla uproszczenia zapisu wprowadzono macierz W zdefiniowaną jako:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.14}$$

Tabela 1-1 Możliwe rozwiązania dekompozycji macierzy E na składowe R i $\overline{T}$

1) $R = U \cdot W \cdot V^T$ $\overline{T} = [U_{13}, U_{23}, U_{33}]$	**2)** $R = U \cdot W \cdot V^T$ $\overline{T} = [-U_{13}, -U_{23}, -U_{33}]$
3) $R = U \cdot W^T \cdot V^T$ $\overline{T} = [U_{13}, U_{23}, U_{33}]$	**4)** $R = U \cdot W^T \cdot V^T$ $\overline{T} = [-U_{13}, -U_{23}, -U_{33}]$

Wyprowadzenie powyższych zależności zostało szczegółowo przedstawione w [2] i [16]. Trzy z czterech przypadków odpowiadają sytuacjom, które w rzeczywistym systemie stereowizyjnym nie mają miejsca, czyli np. jedna kamera skierowana jest w przeciwnym kierunku niż druga, bądź takim które łatwo można rozpoznać (zamiana kamer miejscami skutkuje wektorem translacji z przeciwna wartością). Znając przybliżoną, wstępną geometrię układu (np. uzyskaną w procesie wzajemnej kalibracji z użyciem obiektu referencyjnego) możliwe jest wybranie poprawnego rozwiązania. Poszczególne rozwiązania odpowiadające przypadkom z tabeli 1.1 przedstawiono na rysunku 1.5.



Rys. 1.5 Wzajemne położenia kamer odpowiadające dekompozycji macierzy E, poprawne rozwiązanie oznaczono zieloną ramką (patrz **Tabela 1-1**)[5]

Znormalizowany algorytm 8-punktowy zaproponowany przez R. Hartleya[15] jako danych wejściowych wymaga par współrzędnych odpowiadających sobie punktów. Do rozwiązania tego zadania stosuje się szereg technik: poszukiwanie punktów o charakterystycznym otoczeniu, tworzenie deskryptorów a następnie analizę odpowiadających obszarów na drugim obrazie i estymację ich położenia. Biblioteka OpenCV dostarcza kilkanaście algorytmów znajdujących odpowiadające sobie punkty na parze obrazów. Przykłady ich działania zostały przedstawione w drugim rozdziale tej pracy. Posiadając przynajmniej osiem zestawów par odpowiadających sobie różnych punktów o znanych współrzędnych, problem poszukiwania macierzy fundamentalnej można sformułować w następujący sposób:

niech:

$(u, v, 1)$ to współrzędne homogeniczne punktu na obrazie lewej kamery,

$(u', v', 1)$ to współrzędne homogeniczne punktu na obrazie prawej kamery.

Dla każdej pary punktów istnieje dziewięcioelementowy wektor zdefiniowany jako:

$$\bar{a} = \begin{bmatrix} uu' & uv' & u & vu' & vv' & v & u' & v' & 1 \end{bmatrix} \tag{1.15}$$

[5] Rysunek pochodzi z [2], s. 11, wprowadzono tłumaczenia terminów z języka angielskiego na polski

Dodatkowo, macierz F może zostać przedstawiona jako wektor $\bar{\mathrm{f}}$:

$$\bar{\mathrm{f}} = \begin{bmatrix} F_{11} & F_{21} & F_{31} & F_{12} & F_{22} & F_{32} & F_{13} & F_{23} & F_{33} \end{bmatrix} \tag{1.16}$$

Wprowadzając macierz A, złożona z wektorów odpowiadających parom punktów:

$$\mathrm{A} = \begin{bmatrix} \overline{\mathrm{a}_1} \\ \vdots \\ \overline{\mathrm{a}_\mathrm{d}} \end{bmatrix} \tag{1.17}$$

gdzie:

$\overline{\mathrm{a}_\mathrm{i}}$ – wektor zdefiniowany (1.15) odpowiadający i-tej parze punktów,

d – liczba par odpowiadających sobie punktów, $\mathrm{d} \geq 8$.

$$\mathrm{A} \cdot \bar{\mathrm{f}} = \bar{0} \tag{1.18}$$

Macierz F jest zdefiniowana z dokładnością do czynnika skalującego, dlatego przyjmuje się różne formy normalizacji:

1) $\left\| \bar{\mathrm{f}} \right\| = 1$
2) $F_{33} = 1$

W dalszych rozważaniach przyjęto, że F_{33}=1. W przypadku gdy macierz A ma więcej niż osiem wierszy, (1.18) sprowadza się do problemu minimalizacji średniokwadratowej. Aby znaleźć $\bar{\mathrm{f}}$, można dokonać dekompozycji SVD macierzy $\mathrm{A}^\mathrm{T}\mathrm{A}$, w takim wypadku rozwiązaniem będzie wektor odpowiadający najmniejszej wartości osobliwej. Uzyskana w ten sposób macierz F będzie rzędu trzeciego (ze względu na niedokładność estymacji położenia punktów), a poszukiwana powinna być rzędu drugiego. Każdy punkt leżący poza linią epipolarną powoduje, ze w macierzy A znajdują się nieprawdzie równania (wskazujące na błędną translację i rotację układu stereowizyjnego), dlatego w takim przypadku macierz F będzie zależna od więcej niż sześciu parametrów. Zabieg mający na celu wymuszenie niższego rzędu obliczonej macierzy fundamentalnej został przedstawiony poniżej:

$$\mathrm{F} = \mathrm{U} \cdot \mathrm{diag}(r, s, t) \cdot \mathrm{V}^\mathrm{T} \tag{1.19}$$

$$\mathrm{F}' = \mathrm{U} \cdot \mathrm{diag}(r, s, 0) \cdot \mathrm{V}^\mathrm{T} \tag{1.20}$$

F' jest macierzą fundamentalna rzędu drugiego. Do dalszych rozważań zostało przyjęte, że F jest rzędu drugiego, tj. $\mathrm{F} := \mathrm{F}'$. Hartley pokazał, że zadanie sformułowane w sposób przedstawiony powyżej jest źle uwarunkowane numerycznie, co przy niedokładnościach estymacji położenie rzutów odpowiadających sobie punktów prowadzi do otrzymania nieprawdziwych rezultatów. Zastosowanie transformacji współrzędnych homogenicznych punktów polegającej na przesunięciu i skalowaniu zmniejsza współczynnik uwarunkowania numerycznego średnio z 10^{13} do 10^{5} [15]. Taka transformacja może zostać zdefiniowana w następujący sposób:

$$\mathrm{J}_i = \begin{bmatrix} \frac{\sqrt{2}}{d_i} & 0 & \frac{-w}{d_i} \\ 0 & \frac{\sqrt{2}}{d_i} & \frac{-h}{d_i} \\ 0 & 0 & 1 \end{bmatrix} \tag{1.21}$$

d_i – średnia arytmetyczna odległość punktów wybranych do obliczeń od początku układu współrzędnych obrazu dla lewej i prawej kamery, $i \in \{\mathrm{L},\mathrm{R}\}$, w – szerokość obrazu, h – wysokość obrazu.

Przekształcenie jest zdefiniowane oddzielnie dla lewego i prawego obrazu, ponieważ średnia odległość punktów od początku układu współrzędnych może być różna dla tych obrazów. Wprowadzenie transformacji J_L i J_R wymaga zastosowania operacji odwrotnej, wykonywanej po estymacji macierzy fundamentalnej rzędu drugiego, zatem końcowa wartość F wynosi:

$$\mathrm{F}_{final} = \mathrm{J}_\mathrm{R}^{\mathrm{T}} \cdot \mathrm{F} \cdot \mathrm{J}_\mathrm{L} \tag{1.22}$$

W części praktycznej opisano implementację znormalizowanego algorytmu 8-punktowego. Trzeci rozdział prezentuje analizę wyników działania tego algorytmu dla obrazów rzeczywistych i syntetycznych oraz pokazuje problemy niedokładności obliczeń numerycznych w przypadku tego zadania.

1.2 Zakłócenia występujące w systemach stereowizyjnych i metody ich korekcji

1.2.1 Zniekształcenia toru optycznego kamer

Zniekształcenia wprowadzane przez tor optyczny kamery mają znaczący wpływ na obraz uzyskany na matrycy CCD. Główną przyczyną powstawania zniekształceń obrazu jest niedokładność wykonania soczewek oraz ich nieprecyzyjne ustawienie względem matrycy kamery. Pierwsza grupa zniekształceń nosi nazwę radialnych i może być aproksymowana poniższym równaniem [3]:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1+k_1r^2+k_2r^4+k_3r^6 & 0 \\ 0 & 1+k_1r^2+k_2r^4+k_3r^6 \end{bmatrix} \cdot \begin{bmatrix} x_\mathrm{d} \\ y_\mathrm{d} \end{bmatrix} \tag{1.23}$$

gdzie:

x_d, y_d – współrzędne punktu na obrazie kamery posiadającej zniekształcenia radialne,

x, y – współrzędne punktu na obrazie kamery w przypadku gdy nie występują zniekształcenia radialne,

k_1, k_2, k_3 – współczynniki zniekształceń radialnych,

$\mathrm{r} = \sqrt{\mathrm{x}_\mathrm{d}^2 + \mathrm{y}_\mathrm{d}^2}$ – odległość punktu od środka obrazu.

Zniekształcenia styczne pojawiają się gdy matryca obrazu nie jest ułożona prostopadle do osi optycznej soczewek lub gdy osie optyczne kilku soczewek się nie pokrywają. Równanie (1.24) opisuje przekształcenie geometryczne wprowadzane przez ten rodzaj zniekształceń.

$$\begin{aligned} y &= y_d + \left[p_1\left(r^2+2y_d^2\right)+2p_2x_d\right] \\ x &= x_d + \left[2p_1y_d + p_2\left(r^2+2x_d^2\right)\right] \end{aligned} \tag{1.24}$$

gdzie:

x_d, y_d – współrzędne punktu na obrazie kamery posiadającej zniekształcenia styczne,

x, y – współrzędne punktu na obrazie kamery w przypadku braku zniekształceń stycznych,

p_1, p_2 – współczynniki zniekształceń stycznych,

$\mathrm{r} = \sqrt{\mathrm{x}_\mathrm{d}^2 + \mathrm{y}_\mathrm{d}^2}$ – odległość punktu od środka obrazu.

W przypadku nowoczesnych kamer cyfrowych zniekształcenia styczne są taka małe, że można je zaniedbać [3], natomiast w przypadku kamer o szerokokątnych obiektywach pojawiają się silne zniekształcenia radialne [23]. Zniekształcenia toru optycznego mogą zostać skorygowane programowo poprzez obliczanie poprawnej pozycji danego piksela na obrazie kamery na podstawie jego początkowej pozycji i współczynników zniekształceń. Kalibracja kamery, czyli proces estymacji jej wewnętrznych parametrów oraz wektorów zniekształceń najczęściej jest przeprowadzany przy użyciu obiektu referencyjnego o znanym kształcie i wymiarach, co dodatkowo umożliwia dobranie skali w jakiej są obliczane ogniskowe kamer np. $\frac{\text{piksel}}{\text{cm}}$ lub $\frac{\text{piksel}}{\text{mm}}$. W literaturze [3,6,18] najczęściej stosowanym obiektem referencyjnym jest dwuwymiarowa siatka zbudowana z kwadratów przypominająca szachownicę (ze względu na nieskomplikowany opis matematyczny). Do zalet należy zaliczyć także prostotę fizycznego wykonania takiej „szachownicy", która może zostać wydrukowana na kartce papieru. Estymacja współrzędnych punktów charakterystycznych takiego obiektu referencyjnego możliwa jest przy użyciu metod gradientowych z pod-pikselową dokładnością [6].

1.2.2 Zakłócenia spowodowane nieodpowiednim oświetleniem obserwowanej sceny

Zniekształcenia toru optycznego nie są jednymi elementami zakłócającymi poprawną pracę systemu stereowizyjnego. Drugą grupą stanowią czynniki związane z budową i zasadzą działania kamer cyfrowych. Czułość sensora matrycy CCD lub CMOS, jak również czas przez który strumień fotonów pada na sensor ma wpływ na jakość generowanych obrazów. Istnieje szereg parametrów umożliwiających regulację pracy kamery cyfrowej, min. kontrast czy balans bieli kompensujący temperaturę barwową źródła światła. W tej pracy wszystkie operacje na obrazach zostały wykonane w skali szarości, dlatego zdecydowano się ograniczyć analizę wpływu na pracę kamer do dwóch parametrów: wzmocnienia (ang. *gain*) i czasu ekspozycji (ang. *exposure time*). Pozostałe parametry są dobrane empirycznie tak aby zapewnić jak najlepszą jakość obrazu. Kontrola wzmocnienia i czasu ekspozycji kamery ma duże znaczenie w przypadku scen o zmieniającym oświetleniu. Analiza histogramu obrazu w skali szarości pozwala określić czy dane ujęcie było prześwietlone lub niedoświetlone oraz czy jest wystarczająco zróżnicowane. Na tej podstawie można odpowiednio zmieniać parametry kamery. Wzmocnienie kamery jest zdefiniowane jako [24]:

$$\text{gain} \sim \frac{v}{n} \tag{1.25}$$

n – liczba elektronów zgromadzonych na matrycy CCD,

v – jasność piksela.

W przypadku większości kamer cyfrowych wartość *gain* jest nieujemną liczbą całkowitą, co oznacza, że przy małej wartości *gain* potrzeba większej ilości elektronów zgromadzonych w komórce matrycy CCD odpowiadającej danemu pikselowi aby miał on taką samą jasność jak w przypadku większej wartości *gain*. Gdy obserwowana scena jest mocno oświetlona, w celu uniknięcia prześwietleń obrazu korzystanie jest ustawić małą wartość *gain*. Z drugiej strony w przypadku słabo oświetlonych scen mała wartość wzmocnienia spowoduje, że obraz będzie silnie zaszumiony, gdyż liczba elektronów zarejestrowanych w matrycy CCD będzie porównywalna z liczbą elektronów stanowiących stałe wewnętrzne szumy kamery i przetwornika A/C. Zapewnienie odpowiednio dużej ilości ładunku na sensorze kamery wymaga zwiększenia czasu otwarcia migawki, bądź jej apertury obiektywu. W przypadku tanich kamer cyfrowych nie ma możliwości regulacji przysłony, dlatego w dalszej części pracy rozważa się tylko zmianę czasu ekspozycji. Nie może być jednak dowolnie długi – ograniczeniem jest wymagana liczba generowanych klatek na sekundę oraz efekt *rolling shutter*, który powoduje rozmycie obrazu lub zaburzenia kształtów rejestrowanych obiektów podczas poruszania kamerami lub obserwacji dynamicznej sceny. Przykład zniekształconego obrazu, na którym widoczny jest efekt *rolling shutter* zaprezentowano na rysunku 1.6.

Rys. 1.6 Efekt *rolling shutter*: na obrazie widoczne są zniekształcenia śmigła helikoptera[6]

[6] Rysunek pochodzi z [26]

Obliczanie histogramu obrazu na podstawie wszystkich pikseli generuje dodatkowe obciążenie systemu, dlatego można dokonać decymacji – pod uwagę może być brana np. co dziesiątą kolumna i co piaty wiersz obrazu co oznacza, ze do wyznaczenia histogramu będzie brany co pięćdziesiąty piksel. Warto odnotować, że w przypadku scen filmowanych z perspektywy głowy dorosłego człowieka, piksele znajdujące się bliżej górnej krawędzi obrazu będą z reguły jaśniejsze niż pozostałe. Wynika to z faktu, że są to obszary odzwierciedlające niebo, jest to tzw. efekt jasnego nieba [21]. W celu kompensacji tego zjawiska można nadać pikselom współczynniki wagowe zależne od położenia: im bliżej dolnej krawędzi obrazu tym ich większa wartość. W celu uproszczenia analizy, histogram może zostać podzielony na kilka regionów w następujący sposób:

- każdy region ma przypisany identyfikator numeryczny będący liczbą naturalną,
- pierwszy region ma przypisaną wartość identyfikatora równą 0,
- każdy ma region ma przypisany współczynnik wagowy będący jego identyfikatorem zwiększonym o 1, np.: region nr 1 ma wagę 2, region nr 2 ma wagę 3 itd.

Takie podejście umożliwia opisanie danego obrazu jedną liczbą rzeczywistą, na podstawie której można określić czy jest on niedoświetlony bądź prześwietlony [21], co można przedstawić:

$$\mu = \frac{\sum_{i=0}^{r-1}(i+1)x_i}{\sum_{i=0}^{r-1}x_i} \tag{1.26}$$

gdzie:

μ – wartość opisująca średnie oświetlenie sceny (ang. *mean sample value*),

r – liczba regionów w histogramie,

x_i – liczba pikseli leżących w i-tym regionie histogramu z uwzględnieniem wartości współczynników wagowych wynikających z położenia na osi y obrazu.

Obraz zarejestrowany z odpowiednią wartością ekspozycji i wzmocnienia w stosunku do oświetlenie sceny będzie się cechował następującą wartością μ:

$$\mu \in (\frac{r}{2} - \phi, \frac{r}{2} + \phi) \tag{1.27}$$

gdzie:
φ – dopuszczalne bezwzględne odchylenie wartości μ od średniej wartości liczby regionów, czyli przypadku gdy w każdym regionie histogramu jest jednakowa ilość pikseli.

W praktyce oznacza to, że dla $r = 5$ wartości μ powinna oscylować wokół 2,5. W [21] zaprezentowano algorytm do regulacji długości czasu ekspozycji i wzmocnienia w zależności od obliczonej wartości μ. W tej pracy zaproponowano autorski algorytm, będąc rozwinięciem powyższej metody o:

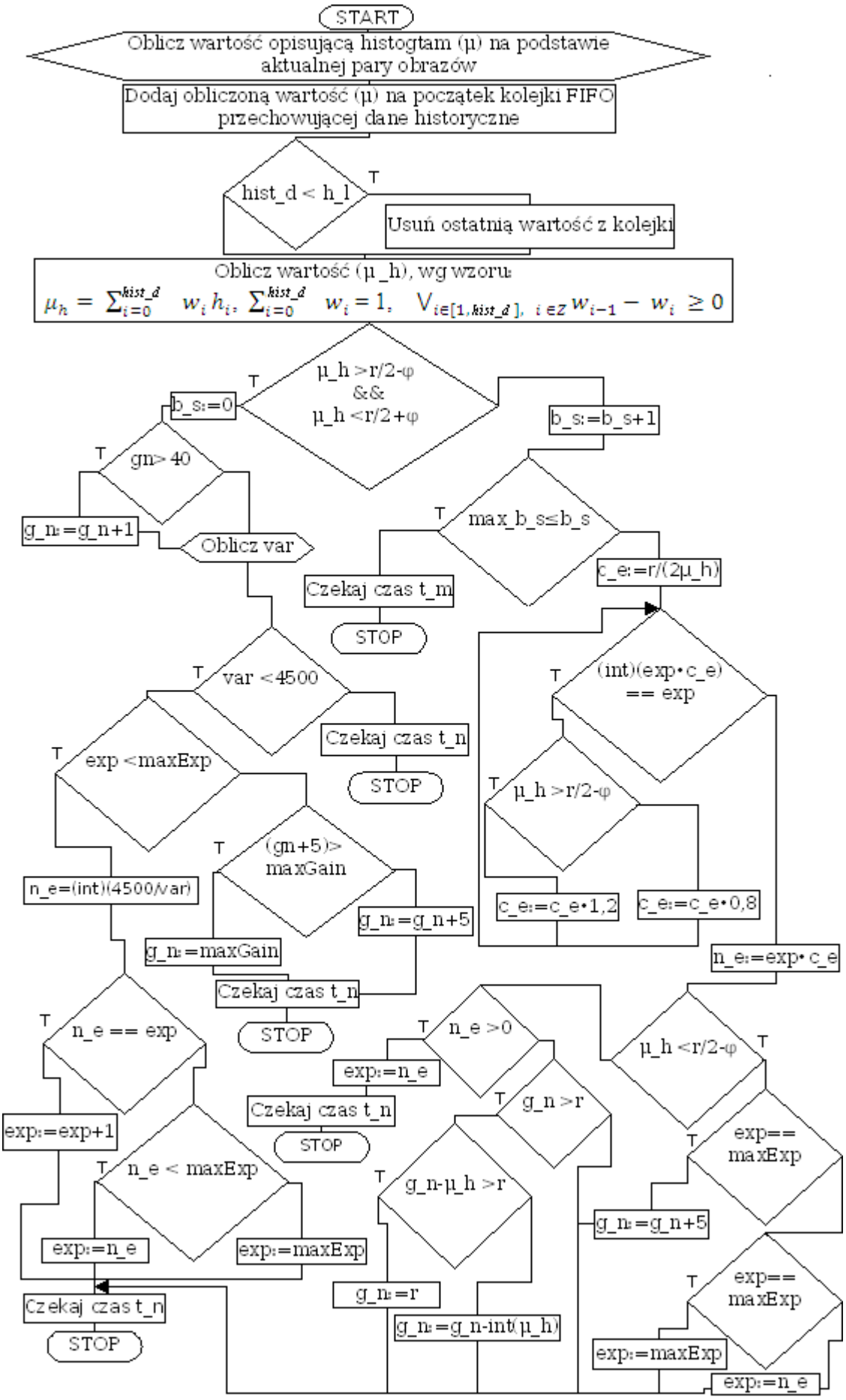
- obliczanie sumarycznego histogramu dla lewej i prawej kamery (μ) oraz podzielenie go na *r* regionów,
- uwzględnienie wartości historycznych (*hist_d*) histogramu aby zapobiegać chaotycznym zmianom wartości *gain* (*gn*) i *exposure* (*exp*) przy krótkotrwałych zmianach oświetlenia sceny (μ_h),
- decymację obrazu przy wyznaczaniu histogramu w celu przyśpieszenia obliczeń,
- uwzględnienie wariancji jasności pikseli (*var*) celem zapobiegania zbytniemu ujednoliceniu obrazu.

Schemat działania tego algorytmu przedstawia rysunek 1.7. Zostały użyte następujące oznaczenia:

- *r* – liczba regionów w histogramie,
- *exp* – obecna wartość *exposure*, wspólna dla obydwu kamer,
- *gn* – obecna wartość *gain*, wspólna dla obydwu kamer,
- μ_h – wartość współczynnika μ z uwzględnieniem wartości historycznych,
- φ – dopuszczalne bezwzględne odchylenie μ_h od wartości $\frac{r}{2}$, wartość wybrana empirycznie tak aby zapewnić akceptowalną jakość obrazu (subiektywna ocena użytkownika systemu),
- b_s – ilość kolejnych par obrazów dla których $\mu\_h > \frac{r}{2} + \varphi$ lub $\mu\_h > \frac{r}{2} - \varphi$,
- *maxExp* – dopuszczalna maksymalna wartość *exposure*, zapewniająca założoną liczbą klatek na sekundę, wartość wyznaczana empirycznie,
- *hist_d* – liczba elementów znajdująca się w kolejce FIFO przechowującej wartości μ_h,
- h_l – maksymalna ilość elementów w kolejce FIFO przechowującej wartości μ_h,

- *var* – wariancja wartości pikseli decymowanych obrazów, obliczona na podstawie tych samych pikseli co wspólny histogram,
- *maxGain* – maksymalna możliwa wartość *gain*,
- *t_n* – odstęp czasu pomiędzy kolejnymi uruchomienia procedury obliczającej i analizującej histogram oraz regulującej wartości *gain* i *exposure*,
- *max_b_s* – maksymalna wartość *b_s*; gdy *b_s* jest równe *max_b_s* oznacza, to że w *max_b_s* iteracjach algorytmu nie udało się ustawić wartości *exposure* i *gain* tak aby zapewnić, że $\mu\_h > \frac{r}{2} + \varphi$ lub $\mu\_h > \frac{r}{2} - \varphi$. Oznacza to, że oświetlenie sceny jest niewystarczające. W takim przypadku regulacja *gain* i *exposure* powinna odbywać się rzadziej, co jest regulowane parametrem *t_m* ze względu na unikanie obciążenia związanego z wyznaczaniem histogramu oraz fakt, że oświetlenie sceny nie zmienia znacząco w odstępach czasu rzędu *t_n*.

Przyjęto założenie, że początek działania algorytmu na rysunku („START”) jest początkiem każdej iteracji a zakończenie („STOP”) jest jej końcem. Przy takim założeniu zmienna *b_s* jest znana na początku każdej iteracji i może mieć różne wartości. Kolejka FIFO przechowująca wartości μ_h jest również znana na początku działania algorytmu.



Rys. 1.7 Schemat blokowy algorytmu sterującego parametrami: *exposure* i *gain*

W algorytmie można wyróżnić trzy przypadki:

1) Wartość μ_h należy do przedziału $(\frac{r}{2}-\varphi, \frac{r}{2}+\varphi)$, co oznacza, że parametry *gain* i *exposure* są właściwie dobrane do oświetlenia sceny, sprawdzana jest wartość wariancji jasności pikseli – w przypadku zbyt małej wartości zwiększana jest wartość *gain*, co jest równoważne ze zwiększeniem kontrastu obrazów.
2) Wartość μ_h jest większa lub równa $\frac{r}{2}+\varphi$ co oznacza, że obraz jest prześwietlony. W zależności od obecnych wartości *gain* i *exposure* zostaną one odpowiednio zmniejszone.
3) Wartość μ_h jest mniejsza lub równa $\frac{r}{2}-\varphi$ co oznacza, że obraz jest niedoświetlony. W zależności od obecnych wartości *gain* i *exposure* zostaną one odpowiednio zwiększone.

Na schemacie blokowym zostały użyte w kilku przypadkach konkretne wartości liczbowe, które mogą być zmieniane w zależności od modelu kamer.

1.2.3 Zakłócenia spowodowane niewystarczającą synchronizacją kamer

W przypadku dynamicznych scen bądź szybkiego poruszania się układu stereowizyjnego względem otoczenia, konieczne jest zapewnienie najkrótszych odstępów czasu pomiędzy pobieranymi obrazami z lewej i prawej kamery. To zagadnienie jest ściśle związane ze sterownikiem kamer oraz metodą pobierania danych z bufora urządzenia. Istnieją zestawy kamer posiadające sprzętową synchronizację, jednak ze względu na wysoką cenę nie zostały wykorzystane. Opis rozwiązań programistycznych mające na celu jak najlepszą synchronizację kamer zaprezentowano w podrozdziale 2.1.

1.2.4 Zakłócenia spowodowane wycięciem użytecznej części obrazu

Jednym z celów tej pracy była analiza systemów stereowizyjnych pod kątem zastosowań w systemach wspomagających osoby niewidome, dlatego istotnym elementem jest rekonstrukcja struktury 3D jak największej powierzchni terenu, w tym podłoża, co preferuje wybór kamer o szerokokątnych obiektywach. Jak zostało wspomniane w tym rozdziale, obiektywy szerokokątne cechują silne zniekształcenia radialne, których całkowita korekcja może być niewykonalna. W przypadku kamer cyfrowych mogących pracować w różnej rozdzielczości, wartość ogniskowej może się

zmieniać (w zależności od rozdzielczości) ponieważ różna szerokość obrazu może być uzyskiwana operacją kadrowania, a nie skalowania. Aby sprawdzić czy takie zjawisko zachodzi można posłużyć się płaskim obiektem z wyrysowaną podziałką wzdłuż osi poziomej i pionowej. Taki obiekt powinien zostać ustawiony w znanej odległości, prostopadle do osi optycznej kamery a następnie należy zmierzyć zakres odległości widziany na obrazie kamery wzdłuż osi X i Y. Korzystając z prostych zależności trygonometrycznych możliwe jest obliczenie kąta „widzenia" kamery podczas pracy w różnych rozdzielczościach. Przykłady pomiarów kątów „widzenia" dla dwóch modeli kamer internetowych o szerokokątnych obiektywach zostały przedstawione w rozdziale 2, gdzie również porównano uzyskane wyniki z danymi podawanymi przez producentów.

1.2.5 Zakłócenia spowodowane wzajemnym poruszaniem się kamer

Ostatnim rodzajem zakłóceń w systemach stereowizyjnych opisanym w tej pracy jest możliwość wzajemnego poruszania się kamer po wykonaniu wspólnej kalibracji z użyciem wzorcowego obiektu. Zakres ruchu nie jest zazwyczaj duży, jego główną składową jest rotacja, która wynika ze sprężystości materiałów z których jest wykonane mocowanie na kamery. Aby rzuty tego samego punktu na obrazach lewej i prawej kamery leżały nadal na liniach epipolarnych, a w związku z czym była możliwa poprawna rekonstrukcja struktur 3D konieczne jest ponowne obliczenie macierzy opisujących układ stereowizyjny bez użycia obiektu referencyjnego. Istnieje szereg technik umożliwiających estymację rotacji i translacji kamer bez użycia wzorca, większość z nich polega na znalezieniu macierzy fundamentalnej układu i jej dekompozycji na składowe przesunięcia i obrotu. Algorytmy mające na celu znalezienie macierzy F posiadają wspólną część: znalezienie par współrzędnych punktów na obrazie lewej i prawej kamery odpowiadających rzutowi tego samego punktu z przestrzeni 3D [16]. Mając odpowiednią liczbę par punktów stosuje się różne metody mające na celu estymację macierzy fundamentalnej. Do najczęściej stosowanych należą: algorytm 8-punktowy, metody Monte-Carlo (algorytm RANSAC) czy metody minimalizacji średniokwadratowej. W tej pracy opisano i zaimplementowano znormalizowany algorytm 8-punktowy ze względu na stosunkowo nieskomplikowany sposób działania oraz na uzyskiwane wyniki o dobrej dokładności, zbliżonej do konkurencyjnych, bardziej złożonych metod [15], [16].

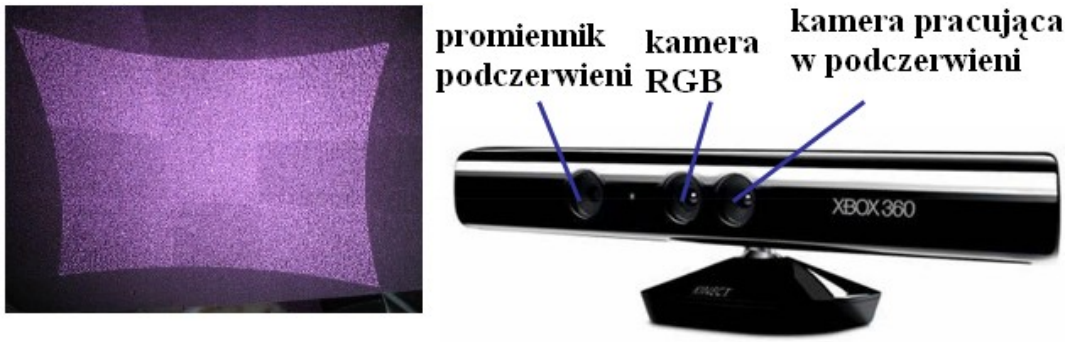
1.3 Metody estymacji mapy głębokości

Mapa głębokości jest obrazem zawierającym rzut chmury punktów na płaszczyznę, której równanie może być dowolnie wybrane, co umożliwia obserwację rzutu danej sceny z różnych perspektyw. Najczęściej wybiera się płaszczyznę przechodzącą przez obiekt rejestrujący obraz, prostopadłą do jego osi optycznej co pozwala na porównanie obrazu rzeczywistego z mapą głębokości. W praktycznych zastosowaniach przyjmuje się najczęściej konwencję, w której rzuty punktów położonych w dużej odległości są oznaczane kolorem ciemniejszym, a leżące bliżej jaśniejszym. Istnieje szereg metod umożliwiających estymację mapy głębokości: z użyciem dwóch bądź więcej kamer cyfrowych, urządzeń korzystających ze światła strukturalnego czy kamer typu TOF. W tym podrozdziale przedstawiono zasadę działania każdej z nich, podano wady i zalety oraz możliwości zastosowania. Przedstawiona zostanie również analiza podstawowych algorytmów do estymacji dysparycji, zdefiniowanej jako różnica współrzędnych rzutów tego samego punktu na obrazie lewej i prawej kamery. Wyznaczenie dysparycji jest niezbędnym krokiem do uzyskania mapy głębokości w układach stereowizyjnych.

Zasada działania systemów typu TOF opiera się na pomiarze czasu między momentem wysłania impulsu światła przez nadajnik, a wykryciem na detektorze odbitego od przeszkody impulsu. Im dalej znajduje się obiekt, którego odległość jest mierzona, tym dłuższy czas propagacji impulsu. Wygenerowanie przebiegów o stromych zboczach i krótkim czasie trwania jest trudne w praktycznej realizacji (nadają się do tego przeważnie lasery), dlatego zastępuje się je sygnałem sinusoidalnym, a w odbiorniku obliczana jest różnica fazy sygnału nadawanego i odbitego. Zastosowanie przebiegów sinusoidalnych pozwala na zbudowanie systemu działającego z różnymi rodzajami nadajników. Wadą takiego rozwiązania jest metoda zredukowania szumów, w której różnica faz sygnałów jest obliczana poprzez całkowanie po czasie, co prowadzi do opóźnień w estymacji mapy głębokości bądź rozmycia związanego z ruchem kamery [4]. Powierzchnie silnie pochłaniające promieniowanie z zakresu podczerwieni mogą nie zostać poprawnie zrekonstruowane. Urządzenia TOF pracują w niskich rozdzielczościach, co może prowadzić do sytuacji, w której obliczone położenie piksela znajdującego się na krawędzi obiektu będzie się znacznie zmieniało w czasie. Do zalety kamer typu TOF należy zaliczyć działanie w czasie rzeczywistym wynikająca z braku

skomplikowanych obliczeń numerycznych niezbędnych do uzyskania informacji 3D oraz bardzo mały wpływ zróżnicowania (tekstury) sceny czy warunków oświetlenia na dokładność i gęstość mapy głębokości [4].

Drugą grupę systemów przeznaczonych do estymacji mapy głębokości stanowią urządzenia, których zasada działania opiera się na właściwościach światła strukturalnego. W skład takiego systemu wchodzi promiennik emitujący światło (przeważnie z zakresu podczerwieni) o określonej strukturze oraz kamera rejestrująca światło z tego samego zakresu. Opcjonalnie stosuje się kamerę RGB, która pozwala na nałożenie odtworzonej chmury punktów na rzeczywisty obraz i wyświetlenie struktury 3D. Przykładem takiego urządzenie jest Xbox Kinect, który znajduje szerokie zastosowanie w grach komputerowych, ale dzięki dostarczeniu przez producenta odpowiednich narzędzi programistycznych pojawiają się przykłady wykorzystania nie tylko dla celów rozrywkowych. W drugim rozdziale tej pracy opisano implementację systemu do estymacji chmury punktów z użyciem urządzenia Kinect, mającym stanowić źródło odniesienia dla wyników uzyskanych z układu stereowizyjnego. Światło emitowane przez promiennik podczerwieni cechuje się specjalną strukturą, która jest unikalna dla pozycji piksela, co przy znajomości wzajemnego położenia promiennika i sensora pozwala jednoznacznie określić położenie przestrzenne danego punktu. W tym celu stosuje się te same zależności co w przypadku systemu stereowizyjnego – tzw. triangulacje, czyli wykorzystanie podobieństwa trójkątów do znalezienie współrzędnej Z (głębokości) punktu. Konieczne jest także użycie algorytmu dopasowującego struktury zarejestrowane przez sensor, aby znaleźć wartość dysparycji dla każdego punktu. Rysunek 1.8 przedstawia zewnętrzną budowę urządzenia Kinect oraz strukturę światła emitowaną przez promiennik na płaską, białą powierzchnię.



Rys. 1.8 Rzut światła strukturalnego na płaską powierzchnię i budowa Kinecta[7]

[7] Rysunek jest połączeniem obrazów z [17] i [4], wprowadzono dodatkowe oznaczenia

Użycie światła strukturalnego z zakresu podczerwieni pozwala zrekonstruować nawet bardzo słabo oświetlone sceny, natomiast czarne czy odbijające, gładkie powierzchnie mogą nie zostać odtworzone ze względu na pochłanianie i rozpraszanie fali emitowanych przez promiennik. Zakłócenia mogą stanowić promienie słoneczne, w których widmie znajdują się składowe z zakresu podczerwieni [17]. Kinect został zaprojektowany do użycia we wnętrzach budynków, dlatego można spodziewać się problemów z estymacją struktur 3D dla scen na otwartej przestrzeni. Przykłady działania tego urządzenia zostały przedstawione w rozdziale trzecim tej pracy.

Trzecią grupę urządzeń do estymacji mapy głębokości stanowią systemy zbudowane z użyciem dwóch lub więcej kamer cyfrowych, którym poświęcona jest ta praca. Po skalibrowaniu kamer i rektyfikacji obrazów (transformacji dzięki której odpowiadające sobie punkty leżą na tej samej wysokości na lewym i prawym obrazie), należy znaleźć dysparycję dla jak największej liczby punktów danego obrazu. Ten problem jest określany w literaturze angielskiej jako *stereo correspondence problem*. Jest to najbardziej złożony obliczeniowo etap rekonstrukcji 3D tą metodą, dlatego ważny jest wybór odpowiednio algorytmu, mając na uwadze takie kryteria jak:

- liczba pikseli dla których znaleziono dysparycję w stosunku do całkowitej liczby pikseli na obrazie (gęstość),
- liczba pikseli dla których znaleziono błędną wartość dysparycji,
- szybkość działania i zużycie zasobów sprzętowych.

W [5] przyjmuje się podstawowe kryterium podziału algorytmów jako gęstość uzyskanej mapy dysparycji. Wyróżnia się metody dające gęste mapy (*dense methods*) i rzadkie (*sparse methods*). Z punktu widzenia tej pracy, estymowana mapa głębokości powinna zawierać jak najwięcej punktów, aby było możliwe rozpoznanie powierzchni terenu, jej nieciągłości oraz potencjalnych przeszkód, zatem dalsze rozważania zostaną zawężone do *dense methods*. Autorzy [22] zaproponowali następujący podział metod dających gęste mapy głębokości:

- metody oparte na korelacji:
  - obliczane jest podobieństwo pomiędzy grupami pikseli ograniczonymi oknem o znanych wymiarach na lewym i prawym obrazie, stosowane są różne metryki, np. L_1 (suma wartości bezwzględnych różnic jasności poszczególnych pikseli na lewym i prawym obrazie) i L_2 (suma

kwadratów różnic jasności poszczególnych pikseli na lewym i prawym obrazie),

- przeszukiwany jest tylko pewien wybrany obszar obrazu, co zmniejsza złożoność obliczeniową,

· metody iteracyjne:

- poszukiwanie mapy dysparycji jest sformułowane jako zadanie optymalizacyjne, najczęściej jest to problem minimalizacji energii,
- do rozwiązania tak sformułowanego zadania można użyć np. algorytmu symulowanego wyżarzania bądź teorii grafów (poszukiwanie minimalnego rozcięcia) [5].

Jako alternatywne metodę do wyżej wymienionych można podać algorytmy obliczające szybką transformatę Fouriera obrazu, a następnie wyznaczane są przesunięcia fazy sygnałów na podstawie czego można w prosty sposób obliczyć dysparycję. Zasada działania i przykłady implementacji takiego systemu zostały opisane w [1].

W [22] dokonano oceny szybkości działania i dokładności wyżej opisanych metod. Okazuje się, że najszybsze są algorytmy oparte na korelacji obrazów z zastosowaniem metryki L_1. Czas obliczenia mapy dysparycji w rozdzielczości 640x480 jest krótszy niż jedna sekunda, zatem to podejście może zostać użyte w systemie działającym w czasie rzeczywistym. Dokładność mapy głębokości (gęstość, ilość błędów) jest mniejsza, jednak czas działania alternatywnych metod (poszukiwanie minimalnego rozcięcia grafu czy algorytm symulowanego wyżarzania) jest o 2-3 rzędy wielkości dłuższy, dlatego w części praktycznej pracy zdecydowano się korzystać z metod opartych na korelacji. Biblioteka OpenCV, z której skorzystano podczas pisania programów, zawiera implementację algorytmów typu *block matching* oraz *graph cuts* [13]. Ze względu na wymaganą pracę w czasie rzeczywistym, zdecydowano się zastosować pierwszy z nich, którego dokładny opis zawarto w [19],[20] i [3]. Trzeci rozdział pracy zawiera analizę szybkości działania implementacji tego algorytmu na procesorach o architekturach ARM i x86, jak również wpływu jakości obrazów wejściowych na wynikową mapę głębokości.

2. Implementacja systemu do rekonstrukcji struktur 3D

W tym rozdziale przedstawiono implementację system do estymacji mapy głębokości na podstawie obrazu z dwóch kamer w systemie Linux i Android. Aplikacja została najpierw wykonana na zwykły komputer w języku C++, korzystając ze szkieletu aplikacji zbudowanego w [19], a następnie za pomocą narzędzi NDK przeniesiono ją na platformę Android. Program został tak napisany aby możliwe było oddzielenie graficznego interfejsu użytkownika wykorzystującego biblioteki Qt4 [12] od części obliczeniowej. Przyjęto założenie, że w środowisku Android zostanie wykorzystana tylko część obliczeniowa, gdyż celem tej pracy było zbudowanie modułu dostarczającego chmurę punktów bądź mapę głębokości do aplikacji wykonującej analizę otoczenia i mającej za zadanie ostrzeganie osoby niewidomej w przypadku wystąpienia zagrożeń.

W podrozdziale 2.1 zostanie omówiona architektura systemu – użyte funkcje i moduły oraz ogólny schemat algorytmu wraz z uzasadnieniem doboru rozwiązań programistycznych i sprzętowych. Następnie zaprezentowana zostanie implementacja algorytmu 8-punktowego jako metody podtrzymywania kalibracji kamer, do której użyto bibliotek umożliwiających przeprowadzanie obliczeń numerycznych z dowolnie wybraną precyzją. Podrozdział 2.3 zawiera opis algorytmu typu *block matching* dostarczonego z OpenCV oraz przedstawia sposób komunikacji z urządzeniem Kinect przy użyciu biblioteki OpenNI. W ostatnim podrozdziale zostanie omówiony sposób działania systemu na platformie Android. Pokazano rozwiązania umożliwiające optymalizację kodu dla architektury ARM oraz wskazano potencjalne problemy związane z błędami działania sterowników poszczególnych urządzeń oraz metody ich naprawy. Każdy z podrozdziałów został wzbogacony o fragmenty kodu implementujące najważniejsze funkcjonalności systemu.

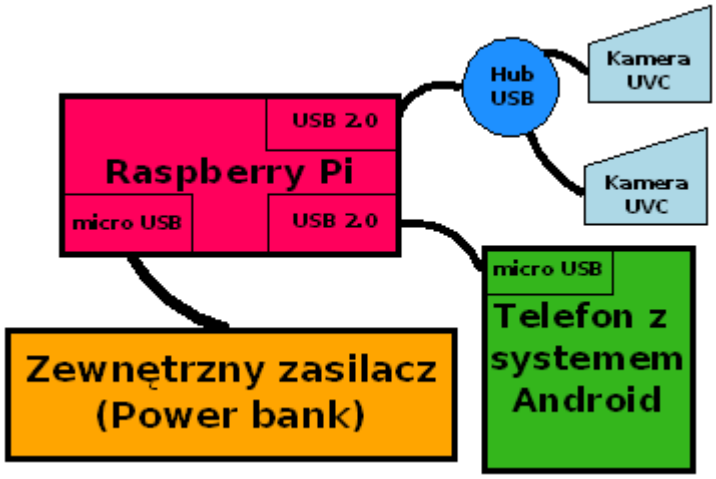
2.1 Architektura systemu

Celem części praktycznej tej pracy było zaprojektowanie i wykonanie systemu do estymacji mapy głębokości dla platformy Android, który ma znaleźć zastosowanie w urządzeniach wspierających poruszanie się osób niewidomych lub ociemniałych. Podstawowym wymaganiem takiego urządzenia są: niewielkie wymiary, mobilność,

praca w czasie rzeczywistym (zapewniająca możliwość swobodnego poruszania się, bez oczekiwania na wynik działania programu) oraz niezawodność uzyskiwanych wyników. System Android został wybrany ze względu na dużą popularność we współczesnych urządzeniach mobilnych, otwarty kod źródłowy oraz wykorzystanie Linuxa w jądrze systemu. Umożliwia to przeniesienie pewnych funkcjonalności między PC a systemem Android, np. dostępu do kamer cyfrowych posiadających sterownik UVC przy użyciu interfejsu V4L2. W jądrze systemu Android domyślnie nie ma sterownika UVC; wynika to z faktu, że podłączanie zewnętrznych kamer do telefonu jest rzadko spotykaną praktyką, zwłaszcza, że często nie posiadają one pełnego kontrolera USB. Do rozwiązania problemu kontrolera USB wykorzystuje się technologię USB OTG (ang. *On The Go*) umożliwiającą pracę urządzenia w trybie *master* w relacji *master-slave* standardu USB. W ramach pracy pobrano źródła jądra systemu Android oraz dokonano ich kompilacji z ustawieniem odpowiednich flag oznaczających dodanie sterownika UVC. Ze względu na trudności z poprawnym uruchomieniem telefonu ze zmienionym jądrem oraz problemami z podłączeniem urządzeń nie wymagających specjalnych sterowników (np. dysków przenośnych) zdecydowano nie podłączać kamer bezpośrednio do telefonu. Dodatkowym czynnikiem motywującym zastosowanie jednostki pośredniczącej w systemie jest możliwość przeniesienia części obliczeń poza telefon. Użycie kamer innych niż kompatybilnych z UVC zostało wykluczone ze względu na brak możliwości regulacji parametrów opisanych w rozdziale pierwszym, dlatego nie zdecydowano na użycie kamer IP. Jako urządzenie będące węzłem pośrednim między kamerami a telefonem wybrano mini komputer Raspberry Pi. Zalety takiego rozwiązania to:

- możliwość uruchomienia systemu Raspbian, pochodnej systemu Debian stworzonej specjalnie pod ten rodzaj urządzeń [11],
- możliwość podłączenia kamer UVC – Raspberry Pi jest wyposażone w kontroler USB oraz sterownik UVC,
- niewielkie wymiary (85 mm x 56 mm x 21 mm) oraz wagę (45 g),
- stosunkowo duże możliwości obliczeniowe (procesor ARM 700MHz z możliwością zwiększenia do 1 GHz), 512 MB RAM współdzielone z GPU,
- niski pobór energii (4-6 W),
- zasilanie przez złącze micro USB, takie samo jak we współczesnych telefonach, co umożliwia podłączenie do tzw. *power banks*.

Następna zaleta wykorzystania Raspberry Pi w projektowanym systemie to niska cena (ok. 30 dolarów) oraz stale rozwijająca się społeczność internetowa użytkowników tego urządzenia. Architektura systemu stereowizyjnego zbudowanego z użyciem Raspberry Pi wymaga uruchomienia aplikacji na dwóch urządzeniach (Raspberry Pi i telefonie z systemem Android) oraz wprowadzenia synchronizacji między nimi, dodatkowo konieczne jest zdefiniowanie interfejsu do komunikacji. Schemat systemu został przedstawiony na rysunku 2.1.



Rys. 2.1 Architektura przenośnego systemu do estymacji mapy głębokości.

Przepływ informacji w systemie jest następujący:

1. Obrazy są cyklicznie pobierane z kamer w procesie uruchomionym na Raspberry Pi.
2. Każda para obrazów jest poddawana korekcji zniekształceń i rektyfikacji.
3. Osobny wątek na Raspberry Pi nasłuchuje zapytań (zdefiniowanej wcześniej sekwencji 3 bajtów) od telefonu na określonym porcie TCP.
4. W przypadku otrzymania zapytania, para przetworzonych obrazów jest wysyłana jako tablica bajtów.
5. W telefonie tablica bajtów jest składana w obraz zgodny z OpenCV, następuje poszukiwanie mapy dysparycji i reprojekcja do chmury punktów.

W części aplikacji uruchamianej na Raspberry Pi wykorzystano program ncat, który umożliwia nasłuchiwania na w ustalonych portach TCP bądź UDP oraz przekierowanie strumieni na standardowe wejście i ze standardowego wyjścia aplikacji. Do podłączenia telefonu wykorzystano technikę USB *tethering* umożliwiającą użycie protokołów z

rodziny Ethernet na interfejsie USB 2.0. Jako protokół sieciowy zastosowano IP, natomiast serwer DHCP jest uruchamiany w systemie Android. Wykorzystanie podłączenia przez USB ma dodatkową zaletę: telefon jest cały czas ładowany, co przy maksymalnym obciążeniu procesora jest niezwykle istotnym aspektem z punktu widzenia czasu działania systemu. W początkowej fazie projektu rozważano podłączenie urządzenia z systemem Android do Raspberry Pi korzystając ze standardu 802.11g, pomimo, że przepływności uzyskane podczas transmisji danych okazały wystarczające, to zrezygnowano z tego rozwiązania ze względu na problemy z utrzymaniem połączenia przez bezprzewodowy adapter USB – co kilkanaście sekund sieć, do której punktem dostępu był telefon przestawała być widoczna, w związku z czym transmisja danych była przerywana. Mając na uwadze planowaną niezawodność, nie można pozwolić na przerwy w działaniu związane z niepoprawną pracą adaptera USB bądź jego sterownika.

Na potrzeby podłączania i kontroli parametrów wewnętrznych kamer została stworzona klasa `Camera`, której każdy obiekt odpowiada jednemu fizycznemu urządzeniu UVC. Jako szkielet do budowy tej klasy wykorzystano aplikację dostępną w [25]. W tabeli 2-1 został przedstawiony fragment pliku nagłówkowego `Camera` wraz z komentarzami.

Tabela 2-1 Fragment pliku nagłówkowego klasy `Camera`

```
struct buffer {
        void * start; //wskaźnik na bufor urządzenia, w celu mapowania go
                      //do pamieci widzianej przez proces
        std::size_t length; //rozmiar obrazu (w bajtach)
};

typedef enum {
        IO_METHOD_READ, IO_METHOD_MMAP, IO_METHOD_USERPTR
} io_method; //metoda dostępu do urządzenia

class Camera {
private:
        void Open();
        void Close();
        void Start();
        void init_userp(unsigned int buffer_size); //zainicalizuj urządzenie
        void init_mmap();                   //jedną z trzech metod
        void init_read(unsigned int buffer_size);  //
        const char *position; // pozycja urzadzenia, w ukladzie dwoch kamer L lub P
        const char *name;  // nazwa pliku reprezentujacgo urzadzenie np (/dev/video1)
        int fps;
public:
         Camera();
        ~Camera();
        Camera(const char *name, int w, int h, int fps, const char *pos);
        int Init(); //zainicjalizuj urzadzenie
        void UnInit(); //zwolnij urzadzenie
        void Stop();   //zatrzymaj prace urządzenia
        int Get();    //pobierz ramke z bufora kamery
        void toIplImage8(); //przetworz ramke z YUY na Y (skala szarości) OpenCV
        int SaveFrame(int index); //zapisz ramke w biezacym katalogu jako poistion_[index].png
        int SaveFrame(char *full_path_name, int index); //zapisz w podanej scieżce.
        float GetFPS(); //pobierz obecną wartość FPS
        int minExposure();  //minimalna możliwa do ustawienia wartosć Exposure
        int maxExposure();  //maksymalna możliwa do ustawienia wartość Exposure
        int lockAutoFocus(); //zablokuj autoFocus kamery
        int getFocus(); //pobierz obecną wartość Focus
        int setFocus(int v ); //ustaw Focus
        int getExposure(); //pobierz wartosc Expsoure
        int setExposure(int v, int w); //ustaw wartość exposure v- czas,
             //w – średnica migawki, niedostępna w większosci urządzeń
};
```

Dane z urządzenia UVC można pobierać na trzy sposoby, jednak podczas testów, podczas których sprawdzono 5 różnych modeli kamer UVC okazało się, że wspierają tylko one jeden z nich: `IO_METHOD_MMAP`. Zasada pracy w tym trybie polega na dołączeniu do przestrzeni adresowej dostępnej dla aplikacji bufora pamięci urządzenia i komunikowania się z nim na zasadzie pamięci dzielonej [10]. Takie podejście pozwala uniknąć nadmiarowego kopiowania z bufora urządzenia do pamięci dostępnej dla aplikacji, ale chcąc modyfikować dane i tak jest koniczne wykonanie ich kopii, w

przeciwnym przypadku mogłoby dojść do zaburzeń pracy urządzenia. W celu kopiowania dużej ilości pamięci wykorzystano funkcję `memcpy`, które pozwala na szybkie kopiowania bloków danych. W klasie `Camera` zaimplementowano funkcje do odczytu i zmiany wartości *exposure*, *gain* oraz do zablokowania *auto-focus,* gdyż do rekonstrukcji 3D niezbędna jest stała ogniskowa kamer. Wynika to z zależności opisujących system stereowizyjny. Macierz reprojekcji pozwalająca wyznaczyć chmurę punktów na podstawie mapy dysparycji jest zależna min. od ogniskowej kamer. Wartość *focus* została wybrana empirycznie tak aby otrzymać użyteczny ostry obraz w zakresie od około 0,5 m do 5-6 m. Wszystkie odwołania do parametrów kamery odbywały się przy pomocy funkcji `ioctl`, przeznaczonej do tworzenia generycznych operacji dla urządzeń I/O. Ustawiania wartości odbywało się poprzez przekazania do `ioctl` struktury składającej się z flagi identyfikującej parametr i pola przechowującego jego wartość, natomiast odczyt polega na przekazaniu wskaźnika do analogicznej struktury i sprawdzeniu odpowiedniego pola po wywołaniu funkcji `ioctl`. Warto zwrócić uwagę, że dane z kamery są pobierane w formacie YUVY co oznacza, że występuję pewien nadmiar informacji, gdyż dalsze przetwarzanie obrazu odbywa się w skali szarości (Y). W tym celu napisano metodę `toIplImage8`, której zadaniem jest odrzucenie składowych chrominancji oraz konwersja danych do formatu kompatybilnego z OpenCV. Tabela 2-2 zawiera kod źródłowy tej metody.

Tabela 2-2 Kod źródłowy metody `toIplImage8` należącej do klasy `Camera`

```
void Camera::toIplImage8() {
    unsigned char *dest = (unsigned char *) (grey_img->imageData);
    int NumPixels=IMAG_SIZE;
    register int i = (NumPixels <<1)-1;
    register int j = NumPixels-1;
    while (i > 0){
       i--;
       dest[j--] = data[i--];
       i--;
       dest[j--] =  data[i--];
     }
}
```

Pole `grey_img` to struktura OpenCV typu `IplImage`, a dane są kopiowane do bufora będącego składnikiem tej struktury. W celu zwiększenia szybkości działania zdecydowano się skorzystać z modyfikatora `register`, będącego „sugestią" dla kompilatora aby dana zmienna była przechowywana w rejestrze procesora, a nie pamięci RAM co znaczenie skraca czas dostępu do niej. Do adaptacji parametrów

kamery do oświetlenia oraz obliczania histogramu wspólnego histogramu kamer została zaprojektowana klasa `StereoController`, której polami są wskaźniki do obiektów klasy `Camera`. Z uwagi na fakt, że nie przewiduje się jednoczesnego użycia więcej niż jednej instancji tej klasy, większość pól i metod posiada modyfikator `static`. W tabeli 2-3 zaprezentowano nagłówek deklarację tej wraz z komentarzami.

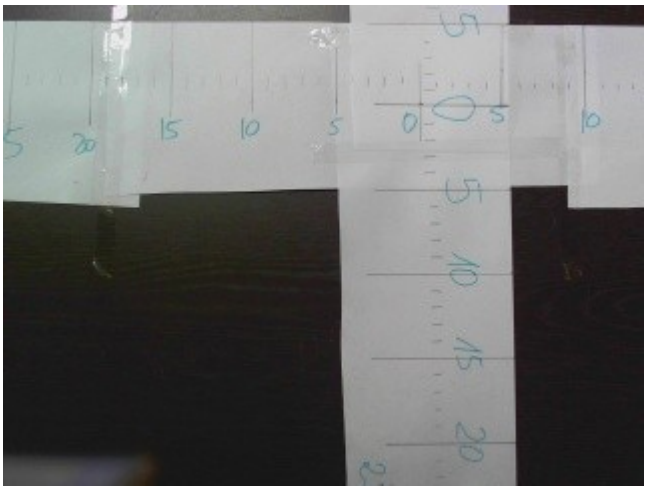
Tabela 2-3 Fragment pliku nagłówkowego klasy `StereoController`

```
class StereoController {
public:
        static void startAnalyzingImages(); //uruchom wątek analizujacy obrazy
        static void stopAnalyzingImages();  //zatrzymaj wątek analizujący obrazy
        static Camera *Lcam; //wskaźnik do obiektu reprezentującego lewą kamerę
        static Camera *Pcam; //wskaźnik do obiektu reprezentującego prawą kamerę
        static char *imgL;  //wskaźniki do kopii danych obrazu z obiektu Lcam
        static char *imgP; //wskaźniki do kopii danych obrazu z obiektu Pcam
        static pthread_t histogram_analyzer; //wątek do obliczania i  analizy histogramu
        static pthread_t calibration_corrector; //wątek do podtrzymywania kalibracji (algorytm 8 pkt)
        static int width;  //szerokść obrazu
        static bool computeHist;  //identyfikator sterujący obliczaniem histogramu
        static int height; //wysokość  obrazu
        static int bad_status_counter; // odpowiada wartości b_s (rozdzial 1.2)
        static int no_of_regions;  //liczba regionów w histogramie
        static int status; //1- hisotgram miesci sie w „normie", 0 -wymaga korekcji
        static int step;  //współczynnik decymacji pikseli (poziomo i pionowo)
        static float sf;  //współczynnik wagowy wartości pikseli
        static float mi_sum ; //obecna wartość mi_sum (u_h)(do wyświetlania w GUI)
        static float images_pix_variance; //wspolna wariancja p. i l. obrazu
        static string directory; //obecny katalog do zapisu danych
private:
        static float CalcSummHist(); //obliczanie wspolnego histogramu
        static void* HistogramLoop(void *ptr); //uruchamia CalcSummHist w
                                               //wątku histogram_analyzer
        static void* controlCalibrationLoop(void *ptr); //podtrzymywanie kalibracji w
                                                        // wątku calibration corrector
        static void CorrectExposure(float current_mi); //ustaw exposure i gain na  podstawie u_h
        static float fi; //dopuszczalne odchylenie od wartości u_h
        static int hh_length; //długość listy przechowującej  u_h
        static float history_weight; // wsp. wagowy dla wartosci historycznych
        static list<float> histogram_history; //lista wartości u_h
        static int limExpforFPS; //max. wartość exposure utrzymująca zadany FPS
};
```

Podstawowym zadaniem klasy `StereoController` jest adaptacja parametrów kamery do aktualnego oświetlenia sceny korzystając z implementacji algorytmu opisanego w rozdziale 1.2 na rysunku 1.7. Wyznaczanie histogramu i jego analiza odbywa się w innym wątku niż główna część programu (związana z pobieraniem obrazu z kamer). Zdecydowano się na takie rozwiązanie ze względu na brak konieczności sprawdzania każdej pary ramek oraz opóźnienie rzędu kilkunastu

milisekund związane z obliczeniami. Aktualna wersja Raspberry Pi (B) posiada tylko jedno CPU, ale w przypadku pojawienia się wielordzeniowych jednostek będzie możliwe uzyskanie przyspieszenia działania aplikacji bez konieczności modyfikacji kodu źródłowego.

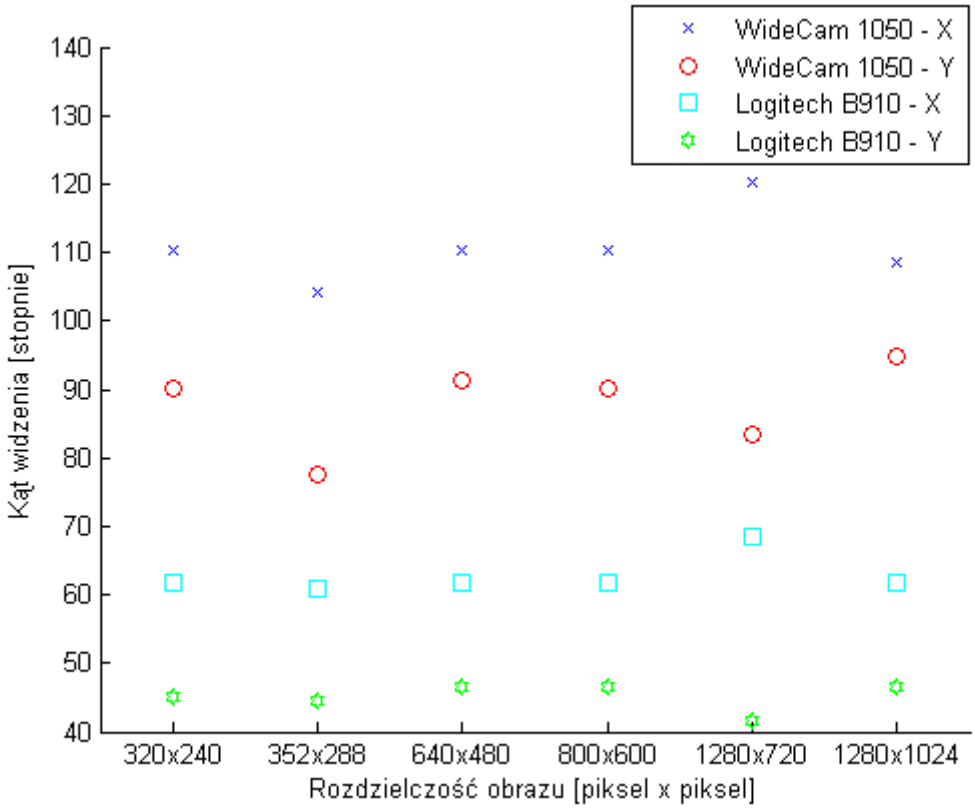
W skład klasy `StereoController` wchodzi struktura typu `pthread_t` o nazwie `calibration_corrector`, w którym uruchamiana jest implementacja znormalizowanego algorytmu 8-punktowego, co zostało opisane w rozdziale 2.2. Pełny kod projektu zarówno w wersji na PC jak i Raspberry Pi z Android został zawarty na płycie CD dołączonej do tej pracy. Wartości parametrów algorytmu adaptującego ustawienia kamer znajdujące się w załączniku zostały wybrany empirycznie dla modelu Logitech B910, który został wybrany spośród czterech innych urządzeń dostępnych w projekcie ze względu na bezbłędne współdziałanie z UVC, wysoką jakość obrazu, szeroki kąt widzenia, niewielki *rolling shutter* oraz możliwość programowego zablokowania funkcji *auto-focus*. W ramach tej pracy wykonano pomiar kąta widzenia dla dwóch modeli kamer, dla których producenci deklarują największe jego wartości. W tym celu skonstruowano planszę testową (rysunek 2.2) ustawioną w znanej odległości i sprawdzono wpływ rozdzielczości pracy kamery na jej kąt widzenia.



Rys. 2.2 Plansza wykonana w celu pomiaru kąta widzenia kamery.

Dokonano analizy w odniesieniu do długości i szerokości obrazu, pomimo faktu, że producenci podają tylko wartość dla osi X. Z punktu widzenia projektu obszar obejmowany przez kamerę wzdłuż osi Y ma duże znaczenie ze względu na estymację kształtu powierzchni znajdującej się w pobliżu użytkownika. Im większy jest kąt widzenia wzdłuż osi Y tym bliżej leżące podłoże będzie zrekonstruowane. Na rysunku

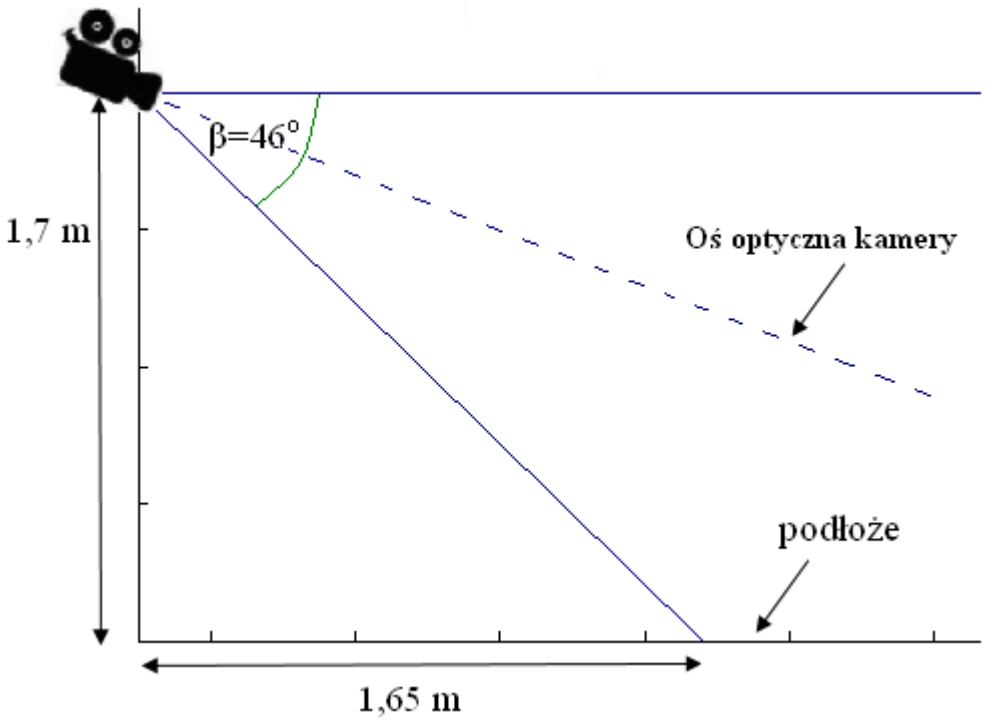
2.3 zaprezentowano zależności kąt widzenia kamer Logitech B910 i WideCam 1050 od rozdzielczości obrazu.



Rys. 2.3 Zależność kątów widzenia kamer od rozdzielczości obrazu

Kamery WideCam 1050 cechują się szerszymi kątami widzenia zarówno wzdłuż osi X oraz Y w każdej rozdzielczości pracy. Deklarowany kąt widzenia przez producenta (120^o) jest osiągany wzdłuż osi X dla rozdzielczości 1280 x 720 Jest to prawdopodobnie nominalna rozdzielczość pracy tego modelu kamery, a proporcje obrazu (1,78 : 1) odpowiadają stosunkowi wymiarów sensora. Dla rozdzielczości o proporcjach ok. 1,33 : 1 kąty widzenia są mniejsze – obraz jest częściowo kadrowany. Dla modelu Logitech B910 zmierzona wartość kąt widzenia wzdłuż osi X dla żadnej z podanych rozdzielczości nie jest równa podawanej przez producenta (71^o). Podobnie jak w przypadku kamer WideCam 1050 największy kąt widzenia w płaszczyźnie horyzontalnej jest uzyskiwany dla obrazu o wymiarach 1280 x 720 jednak nie przekracza on 70 stopni. Kąty w płaszczyźnie wertykalnej różnią się miedzy sobą minimalnie, oscylując w przedziale 41^o – 46^o. Pomimo mniejszych kątów widzenia niż WideCam 1050, ze względu na lepszą jakość obrazu oraz lepszą kompatybilność ze standardem UVC zdecydowano się na wykorzystanie kamer Logitech B910. Przy

założeniu, że kamera znajduje się na wysokości głowy dorosłego człowieka (ok. 1,7 m) i jest skierowana tak jak na rysunku 2.4, możliwe jest zarejestrowanie podłoża znajdującego się w odległości 1,65m od użytkownika. Przy prędkości poruszania się wynoszącej ok. 1 m/s i opóźnieniu przetwarzania informacji równym ok. 500 ms, użytkownik zostanie poinformowany o zbliżającej się przeszkodzie co najmniej sekundę przed potencjalną kolizją.



Rys. 2.4 Zasięg „widzenia kamery" w mobilnym systemie stereowizyjnym przeznaczonym dla dorosłego człowieka

2.2 Przykłady metod podtrzymywania kalibracji kamer

W tym podrozdziale omówiono implementacje algorytmów służących do podtrzymywania kalibracji kamer bez użycia obiektów wzorcowych oraz pokazano autorski sposób implementacji znormalizowanego algorytmu 8-punktowego z użyciem bibliotek MPFR [7] i Eigen [8] w celu przeprowadzenia obliczeń numerycznych z arbitralnie wybraną precyzją.

Metody podtrzymywania kalibracji kamer bez wzorca wymagają dostarczenia współrzędnych par odpowiadających sobie rzutów punktów na lewym i prawym obrazie. Możliwe jest użycie w tym celu algorytmów wykorzystanych do obliczania mapy dysparycji, jednak ze względu na dokładność nie większą niż jeden piksel, szumami związanymi z błędnie dopasowanymi blokami oraz konieczność estymacji macierzy F na podstawie co najmniej 8 par punktów (a mapa dysparycji zawiera ich zbyt dużo – tysiące), zdecydowano się zastosować metody specjalnie dedykowane dla tego zagadnienia. W bibliotece OpenCV zawarto kilkanaście algorytmów przeznaczonych do wykrywania i dopasowywania punktów charakterystycznych z pod-pikselową dokładnością. Warto zwrócić uwagę, że są to metody pozwalające na znalezienie odpowiadających sobie punktów na obrazie poddanym takim transformacjom jak skalowanie czy rotacja, ale kosztem liczby znalezionych punktów. Ich zasada działania opiera się na różnych technikach, ale w każda z nich składa się z dwóch kroków:

1) Znalezienie punktów charakterystycznych na obrazach i wyznaczenie ich deskryptorów (wielowymiarowych wektorów stworzonych na podstawie otoczenia punktu).
2) Dopasowywanie do siebie znalezionych w pierwszym kroku punktów z lewego i prawego obrazu.

W pierwszym kroku często wykorzystuje się narzędzia do detekcji krawędzi, rogów (np. detektor Harrisa) lub metod bazujących na badaniu otoczenia danego punktu. Do najbardziej popularnych algorytmów zaimplementowanych w OpenCV należą:

- *Scale Invariant Feature Transform* (SIFT).
- *Speeded Up Robust Features* (SURF)

- *Maximally-Stable Extremal Region Extractor* (MSER)
- *Binary Robust Invariant Scalable Keypoints* (BRISK)

Wykorzystanie tych metod polega na przekazaniu obrazów jako danych wejściowych, parametrów opisujących próg czułości, założoną liczbę dopasowań oraz metrykę odległości punktów. Po przekształceniu wynik może być zwracany jako struktura typu `std::vector< std::pair<cv::Point2d,cv::Point2d> >`. Przykład użycia metody SURF znajduje się w pliku źródłowym klasy `PositEstimator` na płycie dołączonej do pracy. Metody różnią się czasem działania, ale nie jest to najważniejsze kryterium, gdyż kalibracja nie musi być wykonywana w czasie rzeczywistym, a np. sprawdzana co kilka minut. Dokładność estymacji położenia tych samych punktów może zostać zmierzona jako odległość euklidesowa punktu na jednym z obrazów od linii epipolarnej na tym obrazie obliczonej na podstawie drugiego zdjęcia. Wszystkie metody cechowały się błędem rzędu 0.2 – 3.5 piksela w rozdzielczości 640x480.

W rozdziale pierwszym przedstawiono algorytm estymacji położenia kamer na podstawie macierzy F, zatem proces podtrzymywania kalibracji kamer pokazany w tej części zostanie ograniczony do pokazania implementacji metod obliczania macierzy fundamentalnej. Biblioteka OpenCV dostarcza funkcję `FindFundamentalMat`, która pozwala na kalkulację F jedną z trzech metod [9]:

- RANSAC (*Random Sample Access Consenus*),
- Metoda 8-punktowa (znormalizowana),
- LMeds (*Least Median Squares*).

Wszystkie z tych metod przyjmują jako parametry wektory par punktów odpowiadających sobie, uzyskane np. poprzez użycie SURF, oraz dodatkowe parametry specyficzne dla danej metody, której wyboru dokonuje się poprzez przekazanie odpowiednie flagi. Algorytm RANSAC wybiera losowo 8 par punktów i na tej podstawie obliczana jest wartość początkowa F, następnie wszystkie punkty leżące w odległości większej niż podana (możliwe 1, 2 lub 3 piksele) są odrzucane. Wykonywana jest określona liczba iteracji, w każdej z nich wynik jest ponownie estymowany z losowo dodanej liczby punktów zaklasyfikowanych jako leżące odpowiedni blisko linii epipolarnej (*inliners*). RANSAC jest metodą niedeterministyczną, co w obliczu szumów może prowadzić do otrzymania znacznie różniących się od siebie rezultatów. LMeds jest metodą optymalizacji mającą na celu

wybranie takiej macierzy F, która minimalizuje medianę kwadratów odległości wybranych punktów od linii epipolarnych. Algorytm 8-punktowy został opisany w pierwszej części pracy.

Wszystkie metody dawały znacznie różniące się wyniki w porównaniu do uzyskanych w procesie kalibracji z użyciem obiektu wzorcowego zarówno dla położeń odpowiadających sobie punktów estymowanych poprzez dopasowywanie obrazów jak również przy dokładnym podaniu ich współrzędnych na podstawie znajomości równań linii epipolarnych. Okazuje się, że estymacja macierzy F jest zadaniem podatnym na błędy wprowadzane przez skończoną dokładność obliczeń numerycznych [16]. Aby sprawdzić prawdziwość powyższej tezy zaimplementowano znormalizowany algorytm 8-punktowy z wykorzystaniem bibliotek MPFR i Eigen. MPFR zawiera klasy liczb rzeczywistych i zespolonych wraz z definicją operatorów umożliwiających wykonywanie podstawowych działań matematycznych z arbitralnie wybraną przez użytkownika precyzją. Biblioteka Eigen pozwala na przeprowadzanie szeregu obliczeń z zakresu algebry liniowej korzystając z szablonów klas, co oznacza, że możliwe jest np. wykonanie dekompozycji wg wartości osobliwych macierzy o ile składa się ona z obiektów dla których zdefiniowano niezbędne operacje arytmetyczne. W tabelach 2.5, 2.6 i 2.7 przedstawiono wybrane fragmenty kodu metody `run8PointPREC` należącej do klasy `PositEstimator` odpowiedzialnej za estymację macierzy F z określoną przez parametr `precision` precyzją.

Tabela 2-5. Implementacja algorytmu 8-punktowego z wybraną precyzją obliczeń: typy i klasy pomocnicze zdefiniowane w pliku nagłówkowym `PositEstimator`

```
typedef Eigen::Matrix<mpfr::mpreal, Eigen::Dynamic, Eigen::Dynamic> MatrixXXmp; //macierz
                                                                    //liczb typu mpreal
class Point2m{          // punkt 2D o współrzędnych o rozszerzonej precyzji
    public:
       mpfr::mpreal x;
       mpfr::mpreal y;
};
```

Tabela 2-6. Implementacja algorytmu 8-punktowego z wybraną precyzją obliczeń: obliczenie i zastosowanie transformacji normalizacyjnej

```
mpfr::mpreal::set_default_prec(precision); //ustaw precyzję obliczeń
int size = points1.size();
MatrixXXmp m1p = MatrixXXmp::Zero(size, 2); //kontener na  punkty z lewego obrazu
MatrixXXmp m2p = MatrixXXmp::Zero(size, 2);  //kontener na punkty z prawego obrazu
for (int i = 0; i < size; i++) { //wpisz dane do pojemnikow
          mpfr::mpreal tmpx(points1[i].x);  //pobierz dane typu double i konwertuj je do mpreal
          mpfr::mpreal tmpy(points1[i].y);  // pobierz dane typu double i konwertuj je do mpreal
          m1p(i, 0) = tmpx;
          m1p(i, 1) = tmpy;
          m2p(i, 0) = points2[i].x
```

```
		m2p(i, 1) = points2[i].y;
}
MatrixXXmp A = MatrixXXmp::Zero(9, 9);  //macierz będąca iloczynem A^T *A (por. 1.22, 1.23)
MatrixXXmp F0 = MatrixXXmp::Zero(3, 3);  //macierz wynikowa
MatrixXXmp TF = MatrixXXmp::Zero(3, 3);  // macierz transformacji normalizujacej
MatrixXXmp m0c = MatrixXXmp::Zero(1, 2); // wsp. srodka obrazow po  transformacjach
MatrixXXmp m1c = MatrixXXmp::Zero(1, 2); // wsp. srodka obrazow po  transformacjach
mpfr::mpreal scale0 = 0;	//wsp. skalowania (normalizacji) dla lewego obrazu
mpfr::mpreal scale1 = 0;	//wsp. skalowania (normalizacji) dla prawego obrazu
for (i = 0; i < size; i++) {  //oblicz wsp. srodków oraz srednia odleglosc punkow
		mpfr::mpreal x = m1p(i, 0);   // od srodkow
		mpfr::mpreal y = m1p(i, 1);
		m0c(0, 0) += x;
		m0c(0, 1) += y;
		x = m2p(i, 0);
		y = m2p(i, 1);
		m1c(0, 0) += x;
		m1c(0, 1) += y;
}
mpfr::mpreal t = size; // Skalowanie tak aby sr. Odl. punktow od poczatku ukladu wsp. wynosila sqrt(2)
t = 1. / t;		//  1/ilosc punktow
m0c(0, 0) *= t;
m0c(0, 1) *= t;
m1c(0, 0) *= t;
m1c(0, 1) *= t;
for (i = 0; i < size; i++) { // oblicz odległości od środów obrazów
		mpfr::mpreal x = m1p(i, 0) - m0c(0, 0);
		mpfr::mpreal y = m1p(i, 1) - m0c(0, 1);
		scale0 += mpfr::sqrt(x * x + y * y);
		x = m2p(i, 0) - m1c(0, 0);
		y = m2p(i, 1) - m1c(0, 1);
		scale1 += mpfr::sqrt(x * x + y * y);
}
scale0 *= t;
scale1 *= t;
mpfr::mpreal sqrt_2 = mpfr::sqrt(2);
scale0 = sqrt_2 / scale0;		//scale0 = sqrt(2.) / scale0;
scale1 = sqrt_2 / scale1;		//scale1 = sqrt(2.) / scale1;
//A:=At*A , A -macierz utworzona ze wsp. punktow zgodnie z równaniem 1.20
for (i = 0; i < size; i++) {
		mpfr::mpreal x0 = (m1p(i, 0) - m0c(0, 0)) * scale0;
		mpfr::mpreal y0 = (m1p(i, 1) - m0c(0, 1)) * scale0;
		mpfr::mpreal x1 = (m2p(i, 0) - m1c(0, 0)) * scale1;
		mpfr::mpreal y1 = (m2p(i, 1) - m1c(0, 1)) * scale1;
		MatrixXXmp r = MatrixXXmp::Zero(9, 1);
		r(0, 0) = x1 * x0;
		r(1, 0) = x1 * y0;
		r(2, 0) = x1;
		r(3, 0) = y1 * x0;
		r(4, 0) = y1 * y0;
		r(5, 0) = y1;
		r(6, 0) = x0;
		r(7, 0) = y0;
		r(8, 0) = 1;
		for (int j = 0; j < 9; j++)
			for (int k = 0; k < 9; k++) {
				A(j, k) += r(j, 0) * r(k, 0);
			}
		}
}
```

Tabela 2-7 Implementacja algorytmu 8-punktowego z wybraną precyzją obliczeń: wymuszenie rzędu macierzy oraz odwrotna transformacja normalizująca

```
    //dekompozycja SVD   macierzy A
Eigen::JacobiSVD< MatrixXXmp > svd(A, ComputeThinU | ComputeThinV);

MatrixXXmp Di = svd.singularValues();
MatrixXXmp V = svd.matrixV();

F0(0, 0) = V(0, 8); // macierz F0 odpowiada wektorowi
F0(0, 1) = V(1, 8); // własnemu odpowiadającemu
F0(0, 2) = V(2, 8);  //ostatniej (najmniejszej)
F0(1, 0) = V(3, 8);  //wartości własnej
F0(1, 1) = V(4, 8);
F0(1, 2) = V(5, 8);
F0(2, 0) = V(6, 8);
F0(2, 1) = V(7, 8);
F0(2, 2) = V(8, 8);

//wymuś aby F była rzędu 2, por. (1.23, 1.24)
Eigen::JacobiSVD<MatrixXXmp> svdF(F0, ComputeThinU | ComputeThinV);

MatrixXXmp DiagF = svdF.singularValues();
MatrixXXmp NewDiagF = MatrixXXmp::Zero(3, 3);
NewDiagF(0, 0) = DiagF(0, 0);
NewDiagF(1, 1) = DiagF(1, 0);
MatrixXXmp UF = svdF.matrixU();
MatrixXXmp VF = svdF.matrixV();
VF.transposeInPlace();
F0 = UF * NewDiagF * VF;

//Wykonaj odwrotną transformację normalizacyjną
MatrixXXmp T0 = MatrixXXmp::Zero(3, 3);
MatrixXXmp T1 = MatrixXXmp::Zero(3, 3);

T0(0, 0) = scale0;
T0(0, 1) = 0;
T0(0, 2) = -scale0 * m0c(0, 0);
T0(1, 0) = 0;
T0(1, 1) = scale0;
T0(1, 2) = -scale0 * m0c(0, 1);
T0(2, 0) = 0;
T0(2, 1) = 0;
T0(2, 2) = 1;

T1(0, 0) = scale1;
T1(0, 1) = 0;
T1(0, 2) = -scale1 * m1c(0, 0);
T1(1, 0) = 0;
T1(1, 1) = scale1;
T1(1, 2) = -scale1 * m1c(0, 1);
T1(2, 0) = 0;
T1(2, 1) = 0;
T1(2, 2) = 1;

T1.transposeInPlace();
F0 = T1 * F0 * T0;
F0 /= F0(2, 2); //zwrócna macierz F jest wyskalowana wzgledem elementu F(3,3)
return F0;     //tutaj F(2,2) ze wzgledu że pierwszy indeks to 0.
```

W kodzie metody `PositEstimator::run8PointPREC` zmienna *precision* jest przekazywana jako parametr. Współrzędne punktów odpowiadających są przesyłane jako typ *double* a następnie dokonuje się ich konwersji do *mpfr::mpreal*. W implementacji posłużono się klasą *MatrixXXmp* będącą macierzą dwuwymiarową zmiennych typu *mpfr::mpreal*, których precyzja jest ustawiana globalnie, na początku wywołania metody. Zależność pomiędzy dokładnością obliczeń a precyzją (ilością bitów mantysy) została zaprezentowana w rozdziale 3.1

2.3 Estymacja chmury punktów

W tym podrozdziale przedstawiono najważniejsze fragmenty implementacji algorytmów do estymacji chmury punktów z użyciem biblioteki OpenCV oraz pokazano metody dostępu do urządzenia Kinect, które umożliwiają generowanie danych stanowiących referencyjną chmurę punktów danych wyznaczonych z kamer. Zasada działania algorytmu opierającego się na obrazach z kamer jest taka sama w wersji aplikacji na PC i telefon z systemem Android, dlatego zdecydowano się przedstawić jedno wspólne rozwiązanie. Kinect został użyty wyłącznie w celach badawczych i nie stanowi części ostatecznej wersji systemu. Program zaprojektowany na PC umożliwia jednoczesne nagrywanie obrazów z kamer wraz z sekwencją danych z Kinecta, co pozwoliło na porównanie otrzymanych rezultatów.

Biblioteka OpenCV dostarcza implementację algorytmu typu *block matching*, którego dokładna zasada działania została oraz parametry konfiguracyjne zostały opisane w [3] i [19]. Dla każdej pary obrazów poddanych procesowi korekcji zniekształceń toru optycznego oraz rektyfikacji wywoływana jest funkcja przedstawiona w tabeli 2-8:

Tabela 2-8 Przykładowe wywołanie funkcji `cvFindStereoCorrespondenceBM`

```
cvFindStereoCorrespondenceBM(
remappedL,        //wskaźniki do struktury typu IplImage przechowującej obraz z lewej kamery
remappedP,        //wskaźniki do struktury typu IplImage przechowującej obraz z prawej kamery
disp,          // wskaźnik do struktury typu CvMat przechowującej mapę dysparcyji (przed normalizacją)
BMState);   //wskaźnik do struktury zawierającej parametry alg. Block-Matching
```

Funkcja `CvFindStereoCorrespondenceBM` wykonuje operacje *pre–filteringu* obrazów oraz *post–filteringu* obliczonej mapy dysparycji mające na celu zwiększenie jej gęstości i usunięcie jak największej ilości nieprawdziwych dysparycji. Obliczona mapa dysparycji ma pod-pikselową dokładność, jednak ze względu na uproszczenie dalszych obliczeń, kosztem niewielkiej straty precyzji estymacji położenia blisko leżących punktów stosuje się zaokrąglenie do liczb całkowitych. W tabeli 2.9 zaprezentowano przykładowe wywołanie funkcji `cvConvertScale` służącej do konwersji mapy dysparycji.

Tabela 2-9 Przykładowe wywołanie funkcji `cvConvertScale`

```
cvConvertScale(
disp,  // wskaźnik do struktury typu CvMat przechowującej mapę dysparcyji (przed normalizacją)
 vdisp, // wskaźnik do struktury typu CvMat przechowującej mapę dysparcyji (po normalizacji)
1.0 / 16,  // współczynnik skalowania
0); // przesunięcie, stała wartość dodawana do przeskalowanych wartości dysparycji
```

Współczynnik skalowania wynosi $\frac{1}{16}$ ponieważ pod-pikselowa dokładność mapy dysparycji zwróconej przez funkcję `CvFindStereoCorrespondenceBM` to 4 bity. Następnym krokiem procesu estymacji chmury punktów jest wykonanie reprojekcji 2D do 3D. Na podstawie macierzy Q uzyskanej przy obliczaniu rektyfikacji oraz mapy dysparycji możliwe jest odtworzenie współrzędnych przestrzennych każdego punktu posiadającego przypisaną wartość dysparycji. W tabeli 2-10 przedstawiono wywołanie funkcji *cvReprojectImageTo3D*, której zadaniem jest wykonanie wyżej opisanego przekształcenia.

Tabela 2-10 Przykładowe wywołanie funkcji `cvReprojectImageTo3D`

```
cvReprojectImageTo3D(
    vdisp,  // unormowana mapa dysparycji
    Image3D, //wskaźnik do struktury (macierzy) przechowującej chmurę punktów, struktura ma
             // wymiary długosc x szerokość mapy dysparycji, a jej elementy są typu CvPoint3D32f
    Q,       // wskaźnik do struktury typu CvMat przechowującej macierz  reprojekcji Q
   true);    // flaga, której ustawienie powoduje, że dla punktów bez znalezionej dysparycji zwracane
            //jest  położenie (10000,10000,10000)
```

Punkty znajdujące się w dużej odległości od układu kamer, dla których obliczona dysparycja ma wartość minimalną w mapie dysparycji powinny zostać odrzucone. Wynika to z konwencji przetwarzania mapy dysparycji zastosowanej w funkcji `CvFindStereoCorrespondenceBM`: punkty, dla których wartość dysparycji nie mieści się w zakresie zdefiniowanym przez parametry `minDisparity` i `maxDisparities` mają przypisaną wartość `MinDisparity`-1. W przypadku gdy jednostkami odległości są centymetry wartość 10000 (100 metrów) jest wystarczająca gdyż w systemie wspomagającym osoby niewidome powinna być wykonana analiza najbliższego otoczenia, w zakresie do kliku metrów. Warto zauważyć, że przy rozstawie kamer wynoszącym nie więcej niż kilkanaście centymetrów estymacja położenie odległych punktów (np. kilkadziesiąt metrów) jest obarczona dużym błędem ze względu na małe wartości dysparycji, co pokazuje równanie (2.1) [19].

$$\Delta Z = \frac{Z^2}{fb}\Delta d \tag{2.1}$$

gdzie:

ΔZ – niepewność pomiaru odległości,

Z – odległość punktu od płaszczyzny kamer,

f – długość ogniskowej kamer $[\frac{pix}{cm}]$,

Δd – rozdzielczość dysparycji (1 piksel),

b – odległość między początkami układów współrzędnych kamer.

W układzie stereowizyjnym zbudowanych z kamer o obliczonej ogniskowej równej 547,35 $\frac{pix}{cm}$, umieszczonych równolegle do siebie w odległości 10 cm błąd pomiaru współrzędnej Z obiektów znajdujących się 50 cm od układu kamer wynosi 0,5 cm. Estymacja położenia obiektów znajdujących się 2 m od kamer jest obarczona błędem rzędu 7 cm, a dla obiektów znajdujących się w odległości 5 m błąd wynosi 45 cm. Aby poprawić precyzję estymacji położenia daleko leżących punktów można zwiększyć rozstaw kamer, co może spowodować, że blisko leżące obiekty nie będą jednocześnie widoczne na obrazach obydwu kamer, co jest niezbędnym warunkiem dla uzyskania współrzędnych przestrzennych. W celu analizy chmury punktów zawierającej się w konkretnym zakresie odległości można sprawdzać współrzędną Z wszystkich punktów a następnie odrzucać leżące poza tym zakresem. Jest to przydatne podczas sporządzania i skalowania mapy głębokości. W docelowej wersji aplikacji na platformę Android chmura punktów nie jest analizowana. Pełna struktura `Image3D` jest przesyłana kolejką FIFO do procesu w którym zaimplementowano algorytmy do rozpoznawania i wykrywania przeszkód na podstawie chmury punktów.

Aby urządzenie typu Kinect mogło stanowić źródło odniesienia dla układu kamer należy je umieścić jak najbliżej siebie, w celu umożliwienia obserwacji sceny z jak najbardziej zbliżonej perspektywy. W ramach tej pracy wykonano statyw pomiarowy umożliwiający nieruchome umieszczenie kamer Logitech B910 oraz Kinecta. Do budowy został wykorzystany min. statyw pod aparat fotograficzny, całość wraz z zamontowanymi urządzeniami została pokazana na rysunku 2.5.

Rys. 2.5 Statyw pomiarowy z kamerami Logitech B910 i urządzeniem Kinect

Położenie kamer nie może być regulowane, natomiast Kinect posiada programową możliwość regulacji pochylenia, dlatego empirycznie dobrano je tak aby osie optyczne kamer i Kinecta były równoległe.

Do poprawnej pracy urządzenia Kinect pod systemem Linux konieczna była instalacja odpowiednich sterowników OpenNI, które pozwalają na integrację z OpenCV w taki sposób, że Kinect może być obsługiwany jak zwykła kamera cyfrowa, a wybór trybu pobierania danych 3D polega na ustawieniu odpowiedniej flagi w konstruktorze obiektu będącym uchwytem do urządzenia. W aplikacji na PC znajdują się dwie klasy: `kinect_interface` zawierające implementację dostępu do sensora 3D oraz `matrix_io` pozwalająca na serializację uzyskanych chmur punktów. Fragment pliku nagłówkowego klasy `kinect_interface` został przedstawiony w tabeli 2-11.

Tabela 2-11 Plik nagłówkowy klasy *kincet_interface*

```
#define MAX_FRAMES 600  / ile klatek z Kinecta można zarejestrować w jednym nagraniu
using namespace std;
using namespace cv;
class kinect_interface: public QThread {
        //Q_OBJECT
public:
        VideoCapture capture; //uchwyt do Kinecta
        Mat X[MAX_FRAMES]; //tablica macierzy przechowująca ramki z Kinecta
        long int counter;  //licznik zarejestrowanych klatek-chmur punktow
        bool stop; //gdy stop==1 przerwij nagrywania, stop==0 kontynuuj
        QString path; //sciezka do zapisu plikow
        kinect_interface(); //konstruktor, w którym następuje próba połączenia z Kinectem
        void run();  //rozpocznij nagrywania w osobnym wątku
        void dump_data(); //zapisz dane do plików binarnych po skończeniu nagrywania
};
```

Chmura punktów pobrana z urządzenia Kinect ma rozmiar 4 MB, a zapis do pliku tekstowego nie jest możliwy w wymaganym czasie (poniżej 70 ms; 30 ms zajmuje pobranie danych przez interfejs USB). Zapis struktury jako pliku binarnego również nie spełnia kryteriów czasowych, dlatego zdecydowano się na przechowywanie chmur punktów w pamięci RAM a po zakończeniu nagrania ich zrzut na dysk twardy. Takie podejście stwarza ograniczenie długości filmu, przykładowo przy dostępnych 2 GB wolnej pamięci operacyjnej możliwe jest nagranie 500 klatek, czyli 50-sekundowej sekwencji. Do zapisu danych z Kinecta na dysk oraz ich odczytu na potrzeby późniejszej konwersji do plików tekstowych i mapy głębokości napisano dwie metody klasy `matrix_io`: `load` i `save`, które korzystają z pola tej klasy: `path` – wskazuje na katalog w którym jest przechowywana dana macierz, bądź do którego ma zostać zapisana. W tabeli 2-12 przedstawiono fragment implementacji metody `kinect_interface::run`.

Tabela 2-12 Fragment implementacji metody `kinect_interface::run` przeznaczonej do pobierania danych z Kinecta oraz gromadzenia ich w pamięci RAM

```
long int t = 0  ; //zmienna do pomiarow czas pobierania poszczególnych ramek
long int start;  // zmienna do przechowywania chwili rozpoczecia pomiaru czasu
counter = 0; //licznik

// ustaw tryb generacji "głębi"
capture.set(CV_CAP_OPENNI_DEPTH_GENERATOR_REGISTRATION, 1);
Mat c;  //kontener na pojedynczą chmurę punktów
long int avg = 0; //średni czas pobierania ramek
long int s = 0;  //sumaryczny czas pobierania ramek
while (!stop) {
        start = cv::getTickCount();
        capture.grab();  //pobierz dane z urządzenia
        capture.retrieve(c, CV_CAP_OPENNI_POINT_CLOUD_MAP); //wybierz chmurę punktów
        c.copyTo(X[counter]); //skopiuj macierz z chmurą punktów na kolejne miejsce tablicy
        c.release(); //zwolnij tymczasowa macierz c;

        t = (cv::getTickCount() - start) / 1000; //zakończ pomiar czasu

        if (t < 100000 && avg <= 100000){  //odczekaj tyle, aby calkowity
                                           // czas pobierania ramki wyniosl 100 ms
                usleep(100000 - t);
                s += 100000;
        } else {
                s += t;
        }
        avg = s / (counter + 1); //przelicz ponownie średni czas
        if (counter == MAX_FRAMES || stop == 1)
                break; //jeżeli osiągnięto max liczbe ramek – opusc pętlę i zakończ nagrywanie
        counter++;

        }
        stop = 0;
        this->dump_data(); //zapisz dane znajdujace sie w RAM na dysk twardy
}
```

Obraz z kamer jest rejestrowany z częstotliwością 10 klatek na sekundę, dlatego aby zachować synchronizację z Kinectem, po pobraniu chmury punktów, wywoływana jest funkcja `usleep`, która zawiesza działanie programu na taki okres czasu aby całkowity czas iteracji trwał 100 milisekund. W przypadku gdy użytkownik przerwie nagrywanie danych, poprzez naciśnięcie kontrolki ustawiającej wartość zmiennej `stop` na 1, bądź zostanie osiągnięta maksymalna dozwolona ilość zarejestrowanych ramek, następuje opuszczenie pętli pobierającej dane z urządzenia i ich zapis do odpowiedniego katalogu. W trzecim rozdziale tej pracy dokonano porównania map głębokości uzyskanych z Kinecta i zestawu kamer dla tych samych scen.

2.4 Implementacja systemu na platformę Android

W tym podrozdziale zaprezentowano zastosowane rozwiązania programistyczne i sprzętowe w docelowej wersji systemu działającej na telefonie z systemem Android z wykorzystaniem mikro-komputera Raspberry Pi. Dokonano szczegółowego opisu architektury przedstawionej w rozdziale 2.1.

W części systemu znajdującej się na Raspberry Pi cyklicznie pobierane są klatki z kamer z wykorzystaniem opisanych wcześniej klas `Camera` i `StereoController`, w których następuje korekcja zniekształceń i rektyfikacja pary obrazów. Jedna iteracja pętli zajmuje 130-150 ms, co daje szybkość przetwarzania ok. 7-8 ramek na sekundę. Zwiększenie taktowania zegara procesora Raspberry Pi do 1 GHz pozwoliło skrócić ten czas do 100-110 ms, ale urządzenie zachowywało się wtedy niestabilnie, przykładowo dochodziło do utraty połączenia z jedną z kamer, dlatego zdecydowano się pozostać przez domyślnej częstotliwości taktowania urządzenia (700 MHz). W obrębie procesu działają cały czas trzy wątki:

1) główny wątek, w którym uruchomiona jest funkcja `main`,
2) wątek w którym wykonywana jest adaptacja kamer do oświetlania,
3) wątek, który działa jako serwer obrazów dla telefonu.

Zanim zostaną uruchomione wątki 2) i 3), w głównej części program następuje wczytanie parametrów kamer uzyskanych w procesie kalibracji (wykonanym na PC) z plików tekstowych, obliczenie macierzy przekształceń kalibracji i rektyfikacji oraz nawiązanie połączenia z kamerami wraz z ustawieniem początkowych wartość *gain* i *exposure*. Na potrzeby transmisji obrazów do telefonu stworzono klasę `Streamer`, której zadaniem jest nasłuchiwanie zapytań na standardowym wejściu aplikacji i wypisywanie odpowiedzi na standardowe wyjście. Do uruchomienia aplikacji wykorzystano program ncat, umożliwiający odpowiednie przekierowanie tych strumieni, co zaprezentowano w tabeli 2-13.

Tabela 2-13 Skrypt powłoki służący do uruchomienia aplikacji na Raspberry Pi

```
#!/bin/bash
sudo ./repair_uvc.sh #wywołaj skrypt przeładowujący sterownik uvc z odpowiednim opcjami (quirks)
ip=10.42.0.2 # adres interfejsu usb0 RaspberryPi
ncat -lk $ip 6666 | ./bin/program | ncat -lk $ip 7777
```

Stosując się do powyższej konfiguracji klient powinien wysyłać swoje zapytania na adres 10.42.0.2:6666, a nasłuchiwać na adresie interfejsu podłączonego do Raspberry Pi na porcie 7777. Implementacja wybranych metod klasy `Streamer` została przedstawiona w tabelach 2-14 i 2-15.

Tabela 2-14 Metoda w której wykonywana jest obsługa zapytań o parę obrazów

```
void* Streamer::listenForRequests(void *ptr) {
      char buff[3];  // przechowuje zapytanie od klienta
      int readB=0 ; //liczba wczytanych bajtów
      while (1) {
          readB =read(fileno(stdin), buff, 3); //wczytaj 3 bajty z stdin
          if (readB==3 && buff[0]=='o' && buff[1]=='k'){  //jezeli otrzymano „ok”
              loadImgBuffer(); //skopiuj dane z kamer do bufora na parę obrazów
              dataToStdOut();  //wypisz bufor (tablicę bajtów) z obrazami na stdout
          }
          usleep(20000); //nasłuchuj zapytan co 20 ms / można zmienic
          readB=0; //wyzeruj licznik wczytanych bajtow
      }
}
```

Tabela 2-15 Połączenie obrazów w jedną tablicę bajtów

```
void Streamer::loadImgBuffer(){
    //midd – wskaźnik na „środek” bufora imgBuffer
   //imgBuffer – wskaźnik do tablicy char przechowującej parę obrazów
  //Img_length – rozmiar obrazu w bajtach (640x480)
   memcpy(imgBuffer,L_Ptr,Img_length);  // skopiuj lewy obraz
   midd=&imgBuffer[Img_length];  //przesun wskaznik w buforze
   memcpy(midd,P_Ptr,Img_length); //skopiuj prawy obraz
}
```

Rozmiar jednej klatki wysłanej z Raspberry Pi do telefonu wynosi 600 KB, co oznacza, że do pracy z docelową szybkością (10 klatek/s) konieczne jest połączenie o przepływności co najmniej 48 Mbit/s, co stanowi zaledwie 10 % maksymalnej przepływności standardu USB 2.0. Należy jednak pamiętać, że oprócz transmisji danych do telefonu z tego samego kontrolera USB korzystają dwie kamery pobierające nieskompresowany obraz w formacie YUVY, co generuje znaczne, stałe w czasie obciążenie jednostki SoC.

Aplikacja na platformę Android została napisana w języku C++, a wszystkie niezbędne biblioteki wchodzą w skład systemu operacyjnego, bądź zostały skompilowane statycznie [14]. Takie podejście zwiększa rozmiar pliku wynikowego, ale pozwala w prosty sposób go przenosić między urządzeniami bez konieczności każdorazowej instalacji niezbędnych modułów. Uruchamianie programu może odbywać

się poprzez emulator terminala lub korzystając z aplikacji typowej dla platformy Android, która może wywołać plik wykonywalny. Możliwe jest także zdalne uruchomienie pliku wykonywalnego korzystając z serwera SSH zainstalowanego na telefonie. Podobnie jak w przypadku części systemu dla Raspberry Pi w skład procesu wchodzą trzy wątki:

1) główny wątek programu w którym wykonywana jest rekonstrukcja 3D,
2) wątek, w którym uruchamiana jest metoda pobierająca dane z Raspberry Pi,
3) wątek, który działa jako serwer chmur punktów dla procesu odpowiedzialnego za ich analizę i wykrywanie przeszkód.

W wątku 2) wywoływane są funkcje umożliwiające odczyt określonej liczby bajtów z gniazda, które składa się z adresu interfejsu IP *usb0* telefonu i portu 7777. Następnie otrzymana tablica bajtów jest odpowiednio dzielona i konwertowana do dwóch struktur typu `IplImage`. Aby zapewnić spójność przetwarzanych danych zastosowano mechanizm wzajemnego wykluczania (`mutex`), który umożliwia synchronizację wątków, co zostało przedstawione w tabeli 2-16.

Tabela 2-16 Synchronizacja odczytu obrazów z użyciem mechanizmu `mutex`

```
//1 - kopiowanie zwartości buforaz danymi wczytanymi z RaspberryPi,   wątek 2)

pthread_mutex_lock(&mutex1);  //zablokuj mutex
memcpy(tmpL, buff, WIDTH * HEIGHT);   //skopiuj dane z bufora na parę obrazów do tymczasowego
                                       //kontenera na lewy obraz (tmpL)
midd = &buff[WIDTH * HEIGHT];          // znajdź wskaznik na srodek bufora w celu skopiowania
                                       // prawego obrazu
memcpy(tmpP, midd, WIDTH * HEIGHT); // skopiuj dane z bufora do kontenera na prawy obraz
pthread_mutex_unlock(&mutex1);         //zwolnij mutex
```

```
//2 – kopiowanie danych z tymaczasowych buforów do struktur obrazów typu IplImage ,   wątek 1)

pthread_mutex_lock( &mutex1 );      // zablokuj mutex
 memcpy(L_read->imageData, tmpL, WIDTH * HEIGHT); //skopiuj dane z tym. bufora do obrazu L.
 memcpy(P_read->imageData, tmpP, WIDTH * HEIGHT); //skopiuj dane z tym. bufora do obrazu P.
 pthread_mutex_unlock( &mutex1 );  //zwolnij mutex
```

Odczyt bądź zapis danych do/z struktur `tmpL` i `tmpP` jest wykonywany tylko w dwóch fragmentach programu umieszczonych w tabeli 2-16, co zapewnia że tylko jeden z wątków 1), 2) będzie w danej chwili czasu modyfikował albo wczytywał te dane. Analogiczne rozwiązanie zostało opracowane w celu zapewnienie spójności chmur punktów przesyłanych do aplikacji dokonującej ich analizy. Aby zapewnić jak największą szybkość przekazywania dużej ilości danych (rozmiar struktury przechowującej położenie przestrzenne punktów jest stały i wynosi 3,5 MB) zdecydowano się na stworzenie kanału komunikacyjnego między procesami z

wykorzystaniem dwóch nazwanych łączy typu FIFO. Pierwsze z nich służy do przesyłania żądań. Wysłane są 4 bajty (liczba typu `int`), po ich odczytaniu chmura punktów jest wpisywana na początek drugiego łącza FIFO. Ze względu na systemowe ograniczenie maksymalnej ilości bajtów wpisanej jednym wywołaniem funkcji `write` do FIFO, dane są przesyłane blokowo aż do wpisania ich założonej liczby.

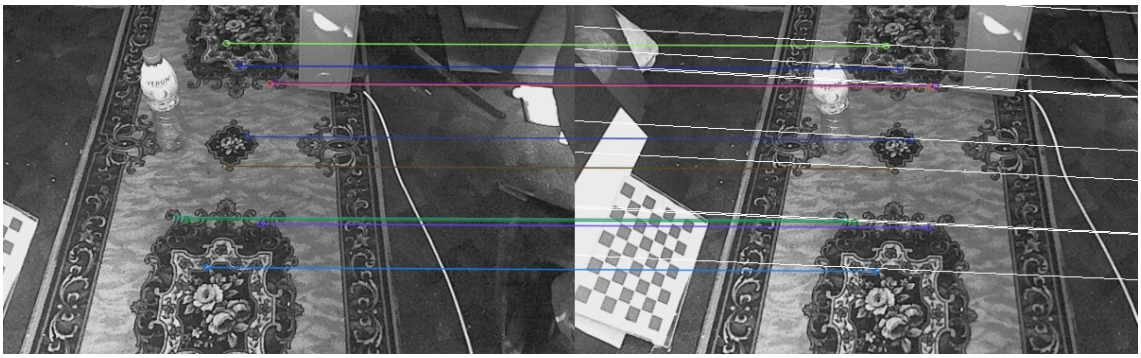
Implementacja metod rekonstrukcji 3D opisanych w podrozdziale 2.3 nie jest specyficzna dla platformy Android, dlatego nie będą one omawiane w tej części. Rozwój biblioteki OpenCV został zapoczątkowany w oddziałach firmy Intel, w związku z tym wiele algorytmów zostało zoptymalizowanych pod architekturę procesorów tego producenta. Wykorzystanie blokowego przetwarzania danych i instrukcji koprocesora umożliwia wykonanie wybranych operacji kilkukrotnie szybciej niż w przypadku aplikacji napisanej bez wykorzystania rozwiązań dedykowanych dla danej platformy sprzętowej. Przykładem optymalizacji kodu dla architektury x86 jest funkcja `CvFindStereoCorrespondenceBM`: użycie techniki SIMD (*Single Instruction Multiple Data*) polegającej na wykorzystaniu struktur i metod SSE2 pozwala na kilkukrotne przyśpieszenie obliczeń (z ok. 350 ms do ok. 50 ms na procesorze Intel Core Duo 2 GHz). Współcześnie produkowane telefony i urządzenia mobilne posiadają procesory ARM, których CPU nie zawiera SSE2. Istnieje implementacja SIMD na tą platformę (NEON), ale nie jest tak rozbudowana jak jej odpowiednik dla x86. Biblioteka OpenCV jest obecnie w fazie rozwoju dla urządzeń mobilnych, dlatego w najbliższych latach spodziewane jest zwiększenie szybkości działania algorytmów na procesorach ARM. Korzystając z obecnej implementacji funkcji `CvFindStereoCorrespondenceBM` z zastosowaniem maksymalnej optymalizacji kodu przez kompilator (flaga –O3) na średniej klasy telefonie z jednordzeniowym procesorem Qualcom Snapdragon (ARMv7) o taktowaniu 1 GHz obliczenia trwają około 800 ms, jednak należy pamiętać, że zastosowanie wielordzeniowej jednostki o wyższej częstotliwości taktowania powinno znacząco skrócić czas wykonywania tego algorytmu. W funkcji `cvReprojectImageTo3D` przeprowadzana jest duża liczba operacji na liczbach zmiennoprzecinkowych, dlatego pomimo braku optymalizacji dla architektury x86 i ARM, czas wykonywania na drugiej z nich jest kilkukrotnie większy ze względu na brak jednostki wspomagającej ten typ obliczeń (FPU). Analiza wydajności całego systemu dla platformy Android została przedstawiona w podrozdziale 3.3.

3. Analiza wyników

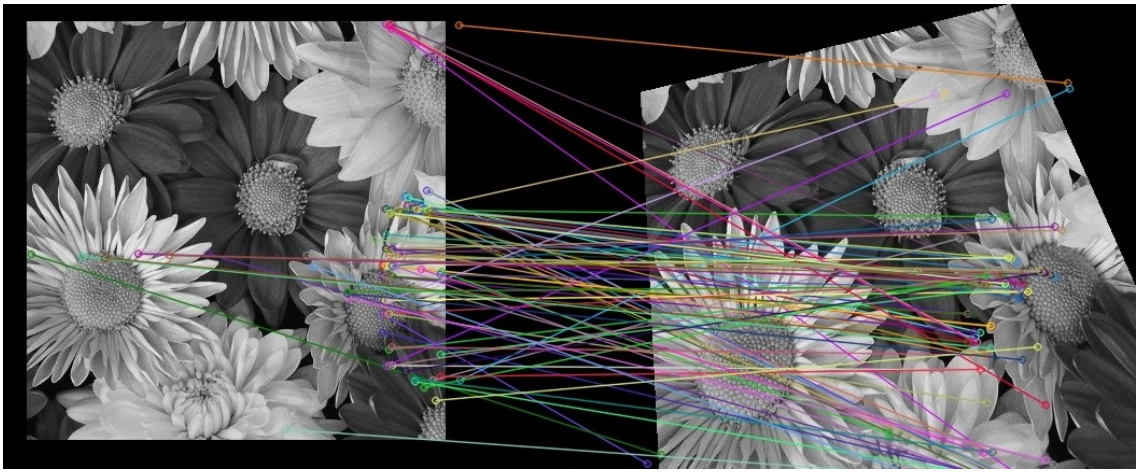
Poniższy rozdział przedstawia analizę dokładności przeprowadzonych pomiarów oraz wydajności poszczególnych elementów systemu stereowizyjnego opisanego w drugim rozdziale. W podrozdziale 3.1 pokazano przykłady działania metod podtrzymywania kalibracji kamer dla obrazów rzeczywistych i syntetycznych. Dokonano analizy wpływu precyzji obliczeń numerycznych na dokładność estymacji macierzy fundamentalnej układu oraz translacji i rotacji. Wyznaczono maksymalny błąd estymacji współrzędnych odpowiadających sobie punktów, który pozwala na odtworzenie macierzy F z taką samą dokładnością jak w procesie kalibracji z użyciem obiektu wzorcowego. Podrozdział 3.2 zawiera porównanie map głębokości uzyskanych w systemie stereowizyjnym i za pomocą urządzenia Kinect oraz analizę czynników wpływających na dokładność otrzymanych wyników. Pokazano również jak cechy obrazów z kamer oraz wybrane parametry algorytmu *block matching* opisanego w rozdziale 2.3 wpływają na gęstość mapy głębokości i liczbę błędnie zrekonstruowanych punktów. Ostatni podrozdział zawiera porównanie wydajności poszczególnych komponentów systemu na platformę Android oraz propozycje ich rozwoju bądź modyfikacji w celu poprawy efektywności pracy.

3.1 Analiza procesu podtrzymywania kalibracji kamer

Proces podtrzymywania kalibracji kamer składa się z dwóch kroków: wyznaczenia bieżącej wartości macierzy fundamentalnej, a następnie znając niezmienne w czasie parametry wewnętrzne kamer, obliczenia nowej macierzy rotacji i translacji kamer. W podrozdziale 2.2 przedstawiono algorytm 8-punktowy, który został wykorzystany w procedurze podtrzymującej kalibrację kamer. Rysunki 3.1 i 3.2 zawierają przykładowe pary obrazów z zaznaczonymi odpowiadającymi sobie punktami dla rzeczywistej i syntetycznej sceny. Do wygenerowania obrazu obserwowanego z różnych, znanych perspektyw wykorzystano zewnętrzny skrypt w języku MATLAB umożliwiający symulację toru optycznego kamery o zadanych parametrach.



Rys. 3.1 Obraz lewej i prawej kamery z odpowiadającymi sobie punktami. Białym kolorem oznaczono linie epipolarne



Rys. 3.2 Para syntetycznych obrazów z zaznaczonymi odpowiadającymi sobie punktami.

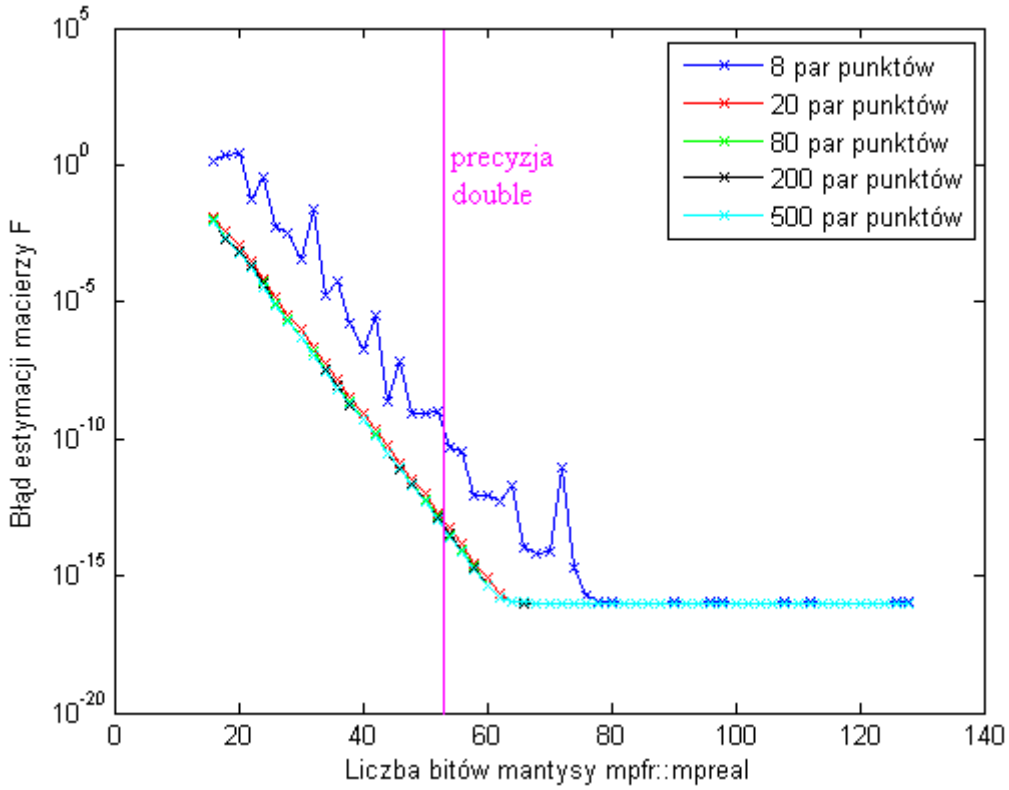
Na rysunku 3.2, niektóre punkty zostały błędnie dopasowane ze względu na duże podobieństwo kliku fragmentów lewego obrazu. Ten problem występuję również w przypadku obrazów rzeczywistych, gdy kamery są skierowane na obiekty o regularnej, powtarzającej się teksturze.

Ze względu na procedurę estymacji położenia kamer wykorzystującą precyzję `double` wyniki były obarczone błędem względnym większym niż 100 %, zdecydowano się sprawdzenie wpływu precyzji obliczeń numerycznych na dokładność obliczenia F. W tym celu wybierano losowo, z rozkładem równomiernym współrzędne punktów na lewym obrazie oraz współrzędną x punktu na prawym obrazie, a następnie wykorzystując F uzyskaną podczas kalibracji z użyciem obiektu wzorcowego

wyznaczano współrzędną *y* punktu na prawym obrazie, tak aby spełnione było równanie (1.12). W przypadku precyzyjnych obliczeń powinno być możliwe dokładne odtworzenie macierzy fundamentalnej. Jako miarę błędu przyjęto:

$$e = \frac{1}{9}\sqrt{\sum_{i=1}^{3}\sum_{j=1}^{3}\left(\frac{F_{ref(i,j)} - F_{(i,j)}}{F_{ref(i,j)}}\right)^2} \tag{3.1}$$

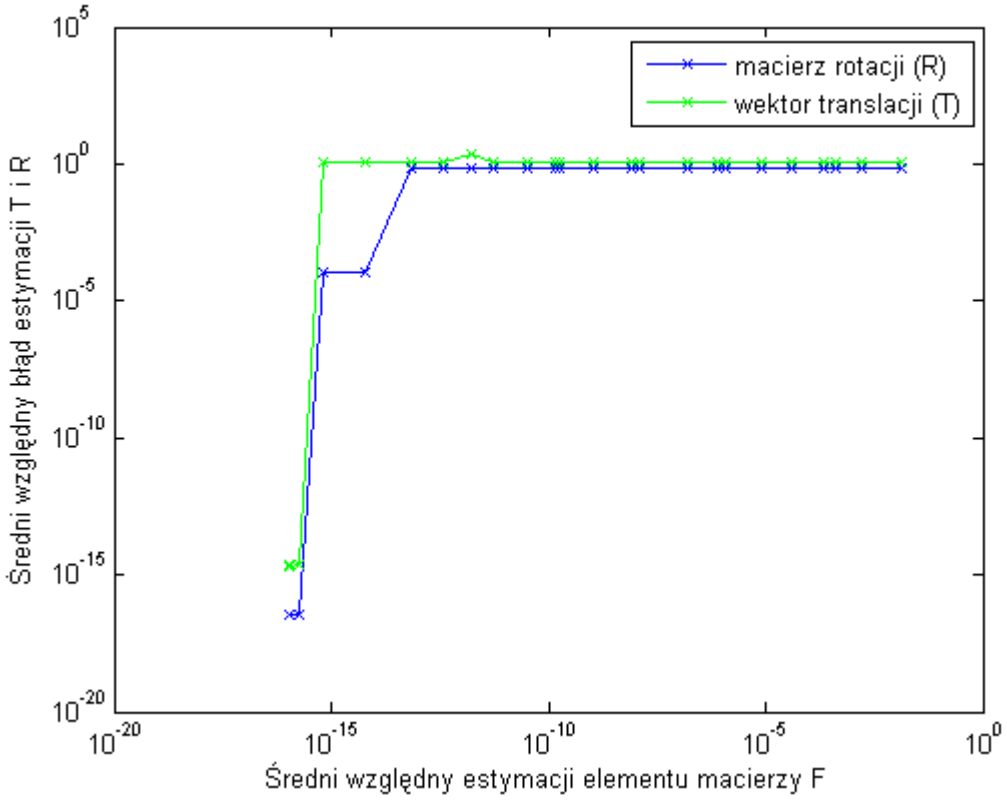
Zdecydowano się na obliczenie względnego błędu estymacji poszczególnych elementów macierzy F ze względu na fakt, że stosunek wartości najmniejszego elementu do największego jest bardzo duży. Podczas estymacji macierzy F dla kilku różnych położeń kamer wynosił on ok. 10^{-10}. Wpływ ilości bitów mantysy zmiennej typu `mpfr::mpreal` na wartość błędu dla różnej liczby par odpowiadających sobie punktów przedstawiono na rysunku 3.3. Wynik został uśredniony dla 50 iteracji.



Rys. 3.3 Wpływ ilości bitów mantysy na dokładność obliczeń macierzy fundamentalnej

W procesie kalibracji z obiektem wzorcowym macierz F została wyznaczona z dokładnością do 16 miejsc po przecinku, dlatego największa uzyskana dokładność wynosi ok. 10^{-16}. W przypadku obliczania macierzy fundamentalnej na podstawie 8 par

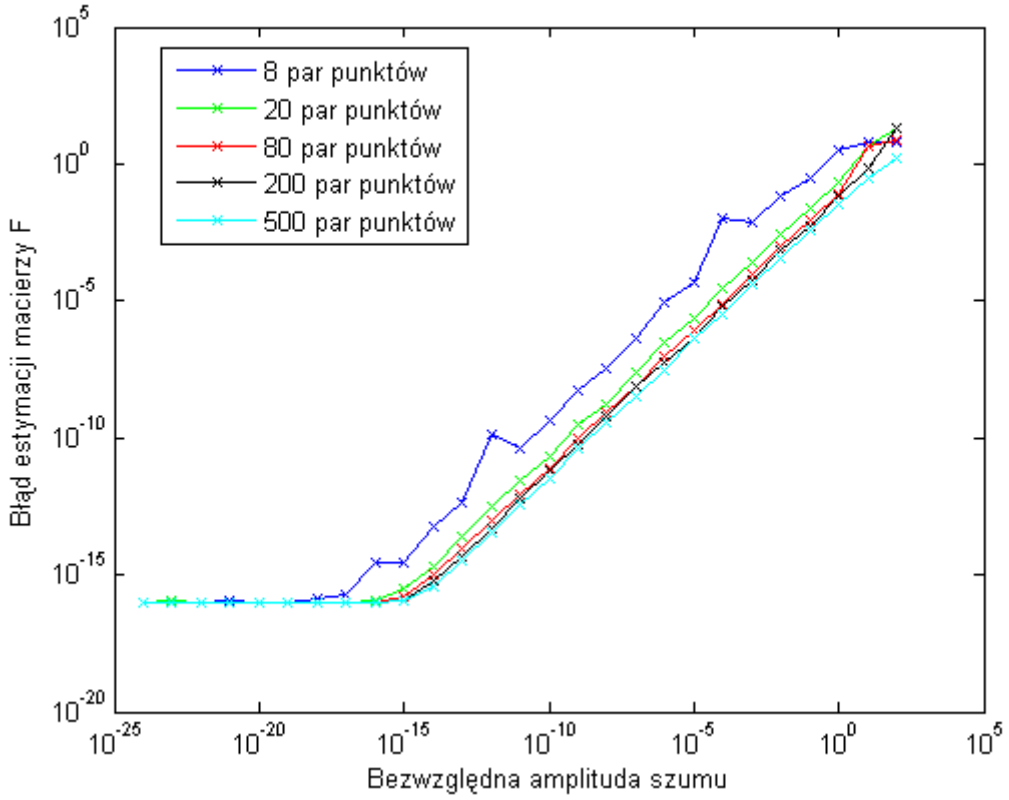
punktów wpływ niedokładności jest większy niż w pozostałych przypadkach przedstawionych na wykresie – wymagane jest przynajmniej 80 bitów mantysy. Dla 20 i więcej par punktów błąd estymacji F zmienia się nieznacznie wraz ze wzrostem liczby par. Do osiągnięcia największej dokładności w tym przypadku potrzeba co najmniej 65 bitów mantysy, czyli więcej niż posiada typ `double` (53). Aby sprawdzić jak błąd estymacji macierzy fundamentalnej wpływa na błąd wyznaczenia macierzy rotacji i wektora translacji wykonano pomiar niepewności obliczenia T i R (wyznaczony analogicznie do błędu estymacji F opisanego równaniem 3.1)w funkcji błędu F. Rezultaty przedstawiono na rysunku 3.4.



Rys. 3.4 Wpływ dokładności obliczeń macierzy F na błąd wyznaczenia T i R

Dekompozycja macierzy F na składowe T i R jest podatna na zakłócenia wartości jej elementów: gdy średni błąd estymacji elementów F wynosi 10^{-12} to średnie błędy wyznaczenia elementów T i R wynoszą ok. 10^{-1}–10^{0}. Wymagana jest dokładność rzędu 10^{-16} aby precyzyjnie odtworzyć macierze translacji i rotacji uzyskane w procesie wspólnej kalibracji z użyciem obiektu wzorcowego. Nawet nieznaczne błędy wyznaczenia F rzędu 10^{-15}–10^{-14} powodują znaczny przyrost niedokładności obliczenia elementów T i R .

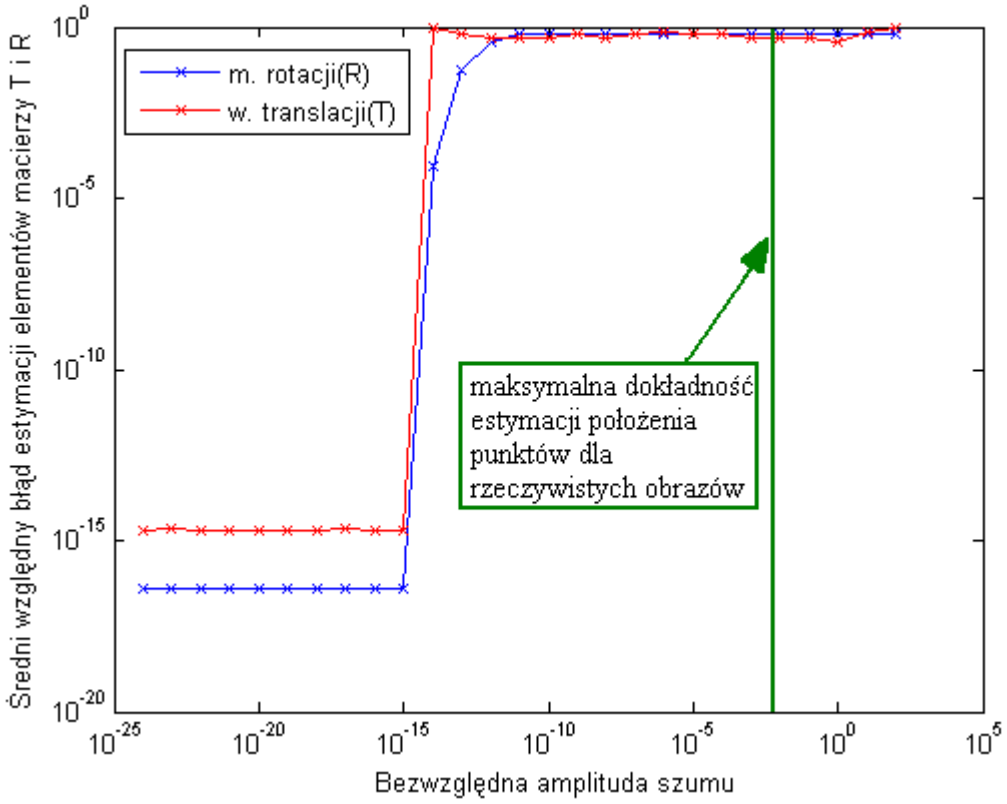
Położenie odpowiadających sobie punktów jest znane z dokładnością ok. 0.5 piksela, co przy rozdzielczości 640 x 480 oznacza błąd względny rzędu 10^{-2}. Aby sprawdzić czy taka dokładność pozwoli na odtworzenie macierz F z największą możliwą dokładnością (10^{-16}) wykonano następujący eksperyment. Położenie punktów na prawym obrazie wynikające z geometrii epipolarnej zostało zakłócone w następujący sposób: do składowych *x* i oraz *y* został dodany szum o rozkładnie równomiernym oraz różnych wartościach amplitudy i wartości średniej równej 0. Wpływ amplitudy szumu na błąd estymacji F opisany (3.1) został pokazany na rysunku 3.5. Obliczenia zostały przeprowadzone z wykorzystaniem liczb zmiennoprzecinkowych o 128 bitach mantysy. Dla każdej wartości amplitudy szumu (w postaci 10^{-k}, $-25 \leq k \leq 2$, gdzie k jest całkowite) wykonano pomiar 50 razy a wynik uśredniono.



Rys. 3.5 Wpływ niedokładności estymacji położenia odpowiadających sobie punktów na dokładność estymacji macierzy fundamentalnej układu

Z rysunku 3.5 wynika, że nawet przy dużej ilości par odpowiadających sobie punktów, aby otrzymać założoną precyzję estymacji F, konieczna jest znajomość ich położenia z dokładnością względną rzędu 10^{-15}. W przypadku rzeczywistych bądź syntetycznych obrazów uzyskane wyniki (błąd względny rzędu 10^{-2}) pozwalają na obliczenie macierzy

fundamentalnej z błędem zdefiniowanym (3.1) równym 10^{-3}, co mając na uwadze zależności wynikające z rysunku 3.4 pozwala stwierdzić, że niemożliwe jest odtworzenie wzajemnego położenia kamer wyłącznie na podstawie obrazu z kamer, bez obiektu odniesienia. Błąd względny estymacji zarówno macierzy translacji i wektora rotacji wynosi 10^{-1}–10^{0}, co sprawia, że procedura kalibracji kamer bez wykorzystania wzorca jest bezużyteczna. Na rysunku 3.6 zaprezentowano wpływ względnego błędu położenia punktów charakterystycznych na prawym obrazie na niedokładność obliczenia T i R.



Rys. 3.6 Wpływ niepewności estymacji położenia odpowiadających sobie punktów na dokładność estymacji wektora translacji i macierzy rotacji

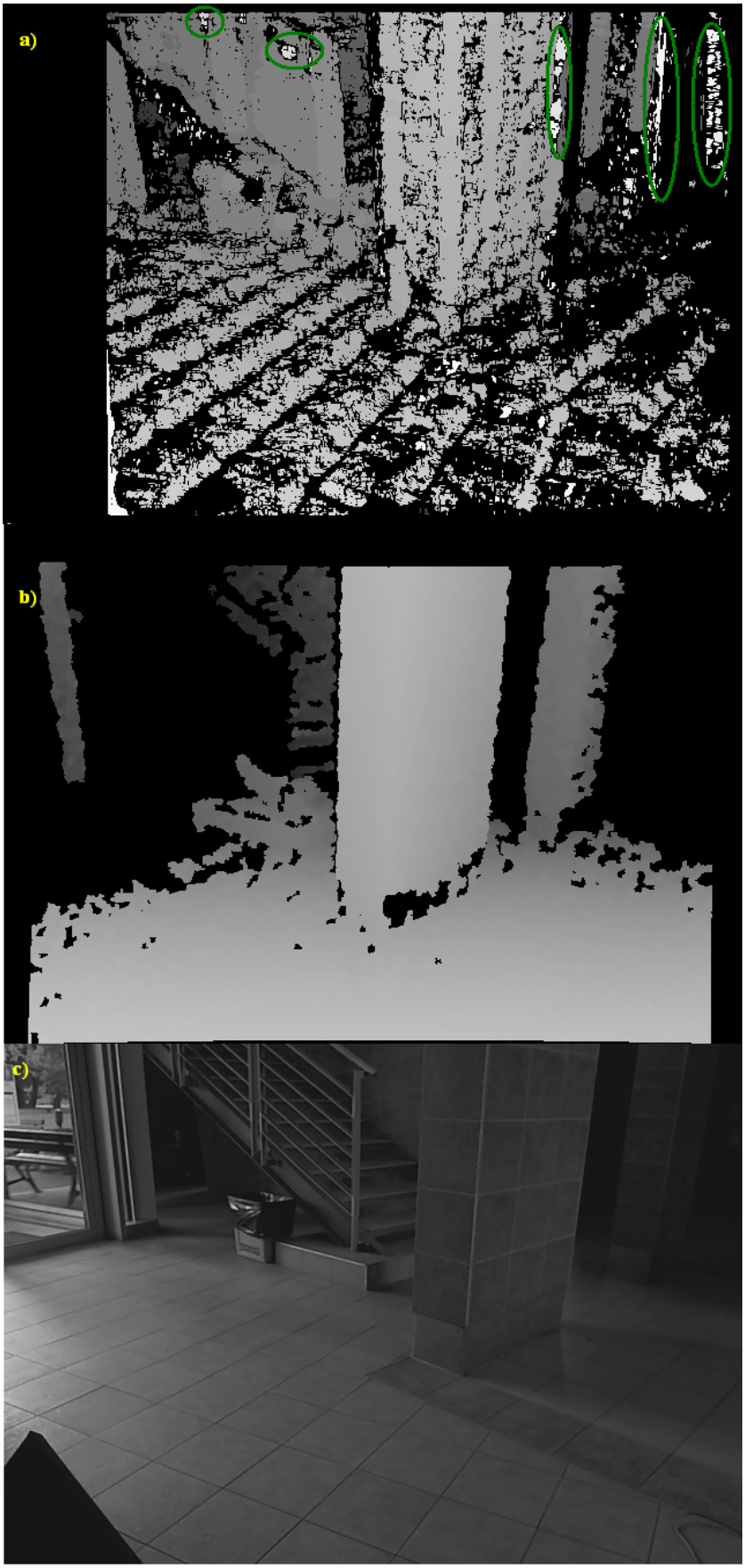
Pomimo wykorzystania do obliczeń zmiennych o 128-bitowej precyzji nie jest możliwe dokładne wyznaczenie macierzy rotacji i wektora translacji. Okazuje się, że aby uzyskać względną niedokładność mniejszą niż 10^{-1} konieczna jest znajomość położenia odpowiadających sobie punktów na lewym i prawym obrazie z błędem względnym co równym co najwyżej 10^{-15}; jest to zadanie niewykonalne zarówno dla obrazów rzeczywistych i syntetycznych. Oznacza to, że nie jest możliwe podtrzymywanie kalibracji kamer bez wykorzystania obiektu referencyjnego.

3.2 Analiza dokładności otrzymanych wyników

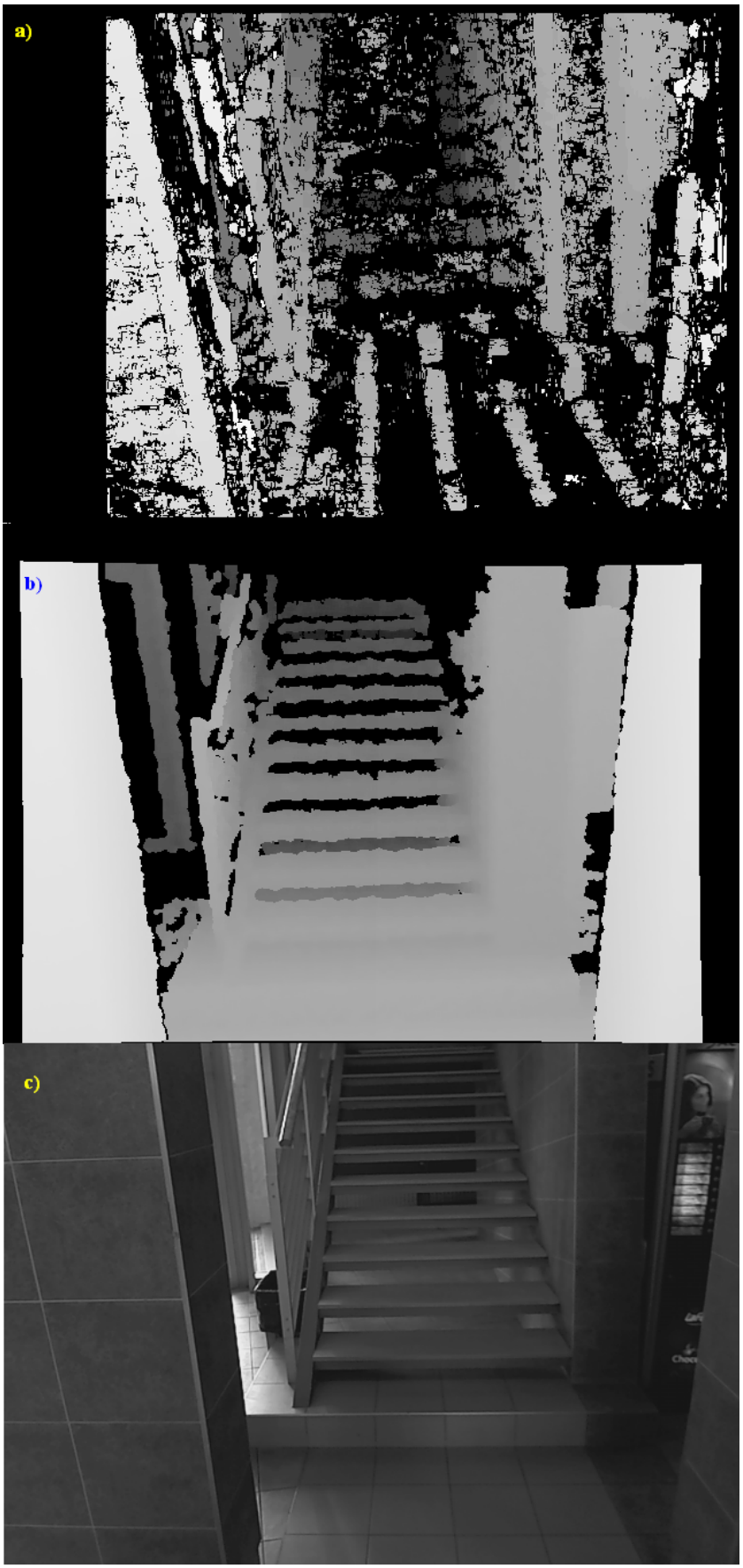
W ramach tej pracy dokonano serii nagrań wg scenariuszy opisujących potencjalne zagrożenia wskazane przez osobę niewidomą podczas poruszania się na zewnątrz i wewnątrz budynków. Zostały pokazane przykładowe mapy głębokości uzyskane z układu kamer i urządzenia Kinect oraz zidentyfikowano czynniki wpływające na ich dokładność i gęstość.

Okazuje się, że nawet przy niedużym nasłonecznieniu praca Kinecta jest na tyle zakłócana, że niemożliwa jest rekonstrukcja 3D – uzyskuje się bardzo rzadką chmurę punktów, a odtworzone wartości są w znacznej części nieprawdzie. Błędne działanie urządzenia wynika z faktu, że w promiennik podczerwieni jest zakłócany światłem słonecznym o zbliżonej częstotliwości. Rysunek 3.7 prezentuje wnętrze budynku zrekonstruowane przy użyciu kamer i Kinecta oraz obraz z lewej kamery zestawu kamer. Na mapie głębokości uzyskanej z kamer oznaczono zielonym kolorem obszary, dla których błędnie odtworzono położenie przestrzenne. Wynika to z niskiej jakości tekstury w tych rejonach, ale na tyle dużej aby nie zostały pominięte. W przypadku zwiększenia progu jakości tekstury (parametr `textureThreshold`) mapa głębokości będzie zawierała mniej punktów, ale zostaną one poprawnie zrekonstruowane. Obszary o jednolitej kolorystyce, bez krawędzi czy rogów nie są praktycznie rekonstruowane. Wyniki uzyskane z Kinecta są dokładniejsze, ale w lewym górnym rogu obrazu tylko nieliczne punkty zostały odtworzone. Jest to rezultat działania promieni słonecznych padających przez szklane drzwi.

Na rysunku 3.8 przedstawiono mapę głębokości dla schodów prowadzących w górę. Pomimo wystarczającego oświetlenia, niska jakość tekstury stopni spowodowała, że nieliczne punkty są zrekonstruowane, natomiast poręcze i ściany zostały dobrze odtworzone ze względu na dużą ilość punktów charakterystycznych.

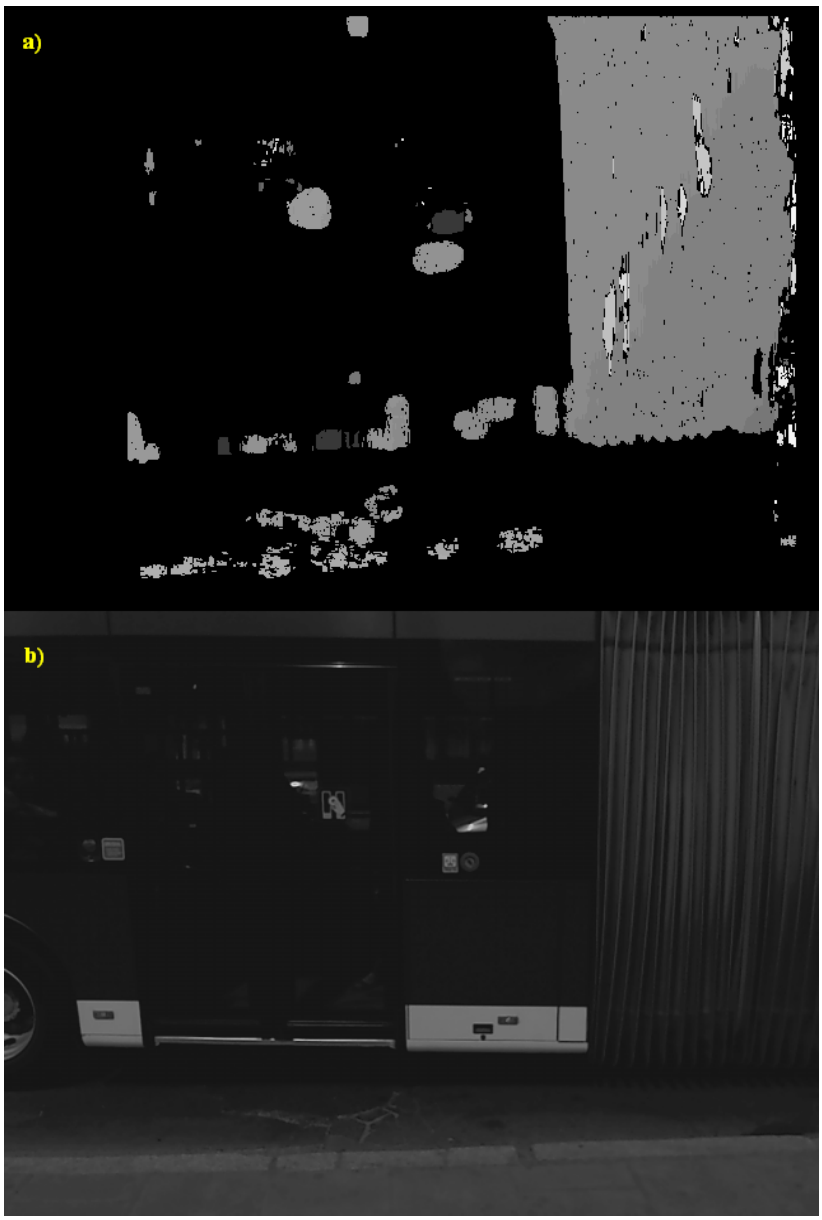


Rys 3.7 Mapa głębokości uzyskana z: a) układu kamer, b) urządzenia Kinect, c) obraz z lewej kamery zestawu kamer



Rys 3.8 Mapa głębokości uzyskana z: a) układu kamer, b) urządzenia Kinect, c) obraz z lewej kamery zestawu kamer

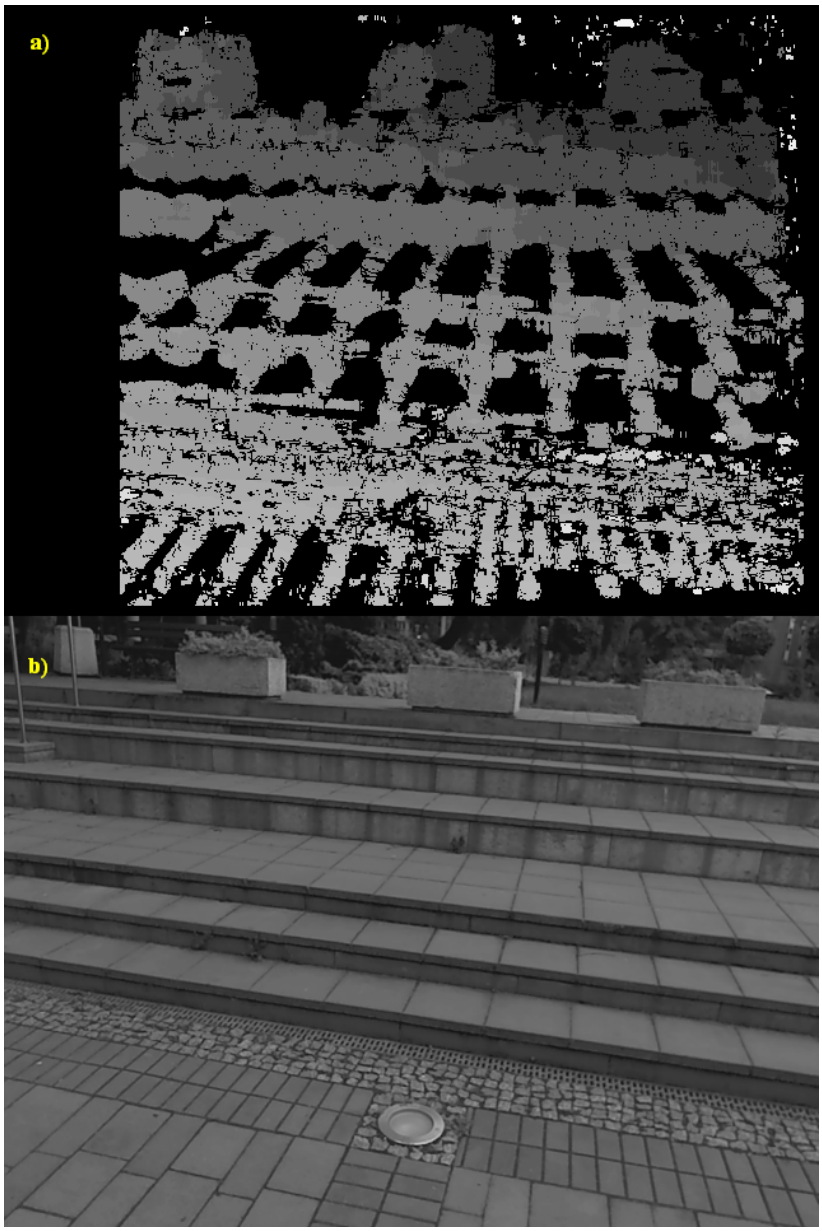
Dla scen na zewnątrz budynków nie zostały załączone mapy głębokości z urządzenia Kinect ze względu na nieprawidłowe działanie. W tym przypadku Kinect nie mógł stanowić odniesienia dla wyników pochodzących z systemu stereowizyjnego. Na rysunku 3.9 zaprezentowano fragment jednego ze scenariuszy wskazanych przez niewidomą osobę: autobus podjeżdżający na przystanek, identyfikacja położenia drzwi oraz sprawdzenie czy są otwarte.



Rys 3.9 Mapa głębokości uzyskana z układu kamer (a)), obraz z lewej kamery (b))

Drzwi i szyby są wykonane ze szkła, co powodowało liczne odbicia i prześwity, dlatego nie było możliwe zrekonstruowanie tych obszarów, natomiast przegub autobusu

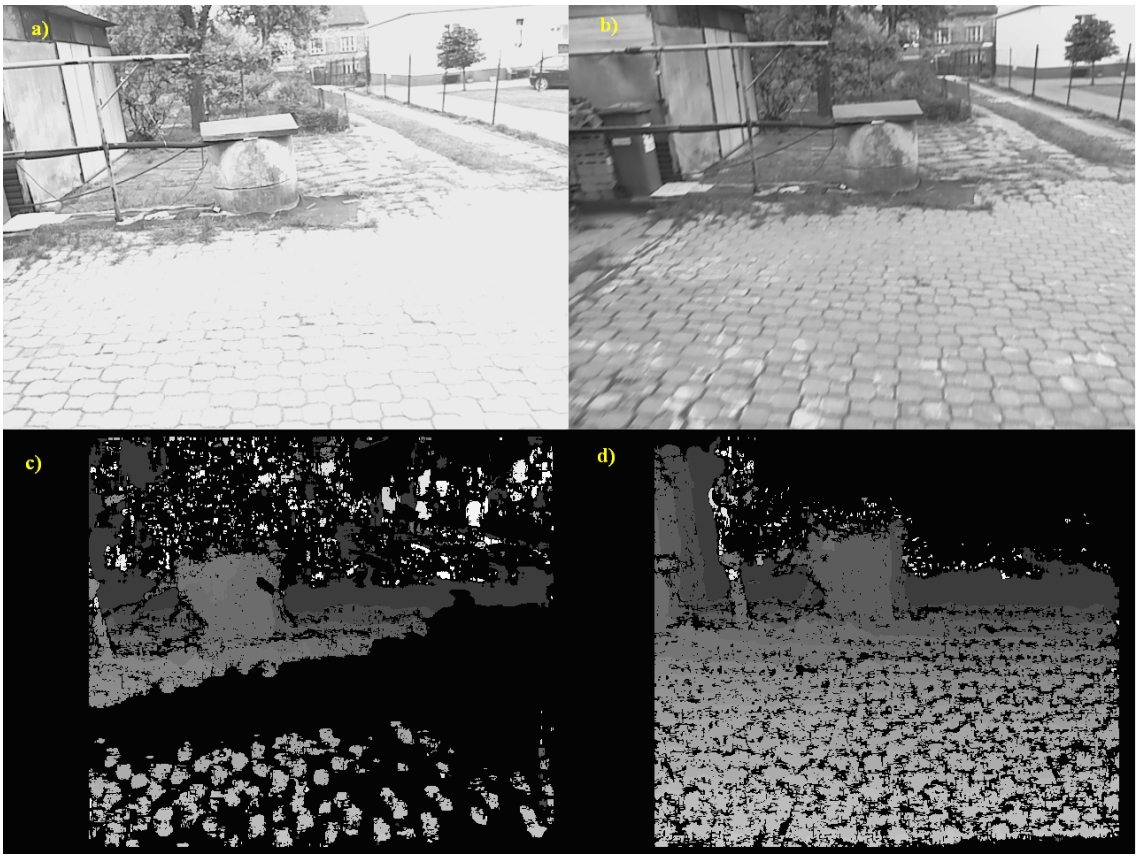
posiadający zróżnicowaną teksturę został poprawnie odtworzony. Realizacja tego scenariusza nie powiodła się. Na podstawie uzyskanej chmury punktów nie jest możliwa identyfikacja drzwi pojazdu. Na rysunku 3.10 zaprezentowano fragment schodów w górę na zewnątrz budynku i mapę głębokości uzyskaną z kamer.



Rys 3.10 Mapa głębokości uzyskana z układu kamer (a)), obraz z lewej kamery (b))

Schody oraz podłoże zostały wykonane z elementów o różnorodnej teksturze, dużej liczbie krawędzi oraz rogów, więc było możliwe ich dokładne odtworzenie, jednak płaskie i jednolite powierzchnie, np. płyty chodnikowe nie zostały w pełni zrekonstruowane, możliwa była tylko estymacja ich konturów, niemniej jednak taka dokładność jest wystarczająca do oszacowania i analizy geometrii otoczenia. Aby

pokazać jak istotnym czynnikiem jest adaptacja kamer do warunków oświetlenia, wykonano nagranie tej samej sceny przy dużym nasłonecznieniu w przypadku ustawienia małej (0,05) i dużej (0,35) wartości parametru φ algorytmu regulującego wartości *gain* i *exposure* układu. Dla φ = 0,05 dopasowanie *gain* i *exposure* nie było możliwe w taki sposób aby system był stabilny, czas ekspozycji zmieniał się chaotycznie a wartość μ oscylowała w przedziale od ok. 1,9 do 3,1 nigdy jednak nie osiągając dopuszczalnych wartości (2,45-2,55) na dłużej niż okres między pobieraniem ramek z kamer. Wynikiem gwałtownych zmian wartości μ były naprzemiennie niedoświetlone i prześwietlone sceny, co wpływało niekorzystanie na uzyskaną mapę głębokości pomimo wysokiej jakości tekstury poszczególnych jej elementów. Rysunek 3.11 zawiera porównanie obrazów i map głębokości dla dobrze i źle dobranych parametrów *gain* i *exposure.*



Rys 3.11 Obraz i mapa głębokości przy niewłaściwej adaptacji kamer do oświetlenia: a) , c) oraz przy poprawnej adaptacji b), d).

Prześwietlenie sceny spowodowało spadek jakości tekstury uniemożliwiający rekonstrukcję podłoża oraz pojawienie się niepoprawnie odtworzonych punktów dla daleko położonych obiektów (prawa, górna część obrazu).

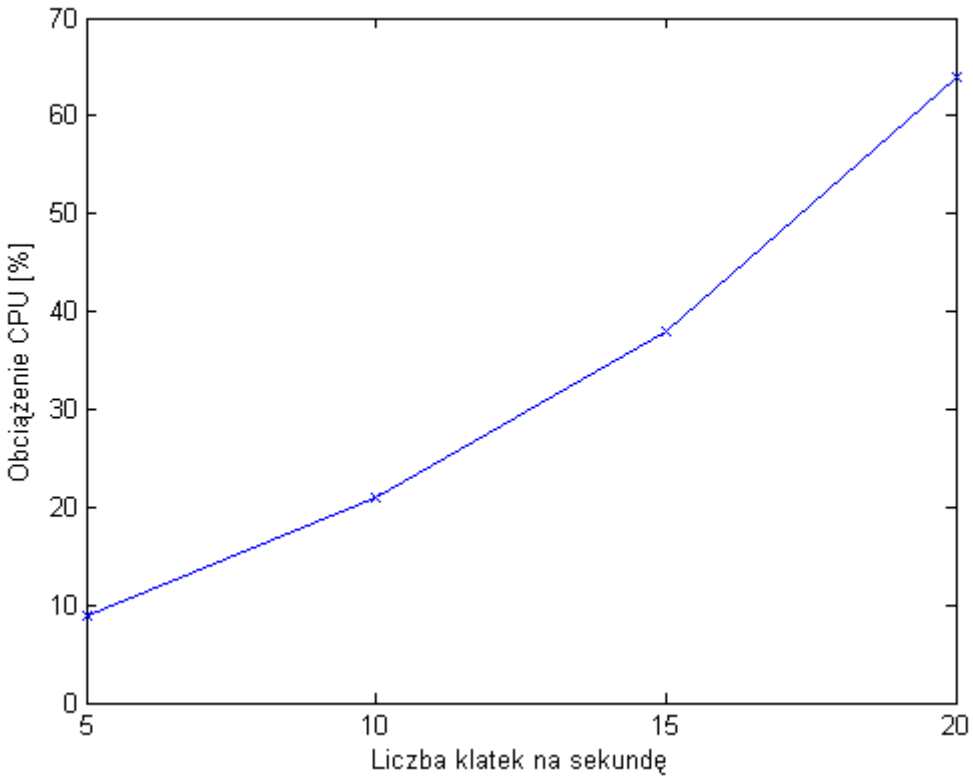
Na podstawie wykonanych pomiarów można sformułować następujące wnioski:

- Kinect nadaje się do pracy wyłącznie w zamkniętych pomieszczeniach, bez dostępu światła słonecznego,
- Odpowiednia adaptacja parametrów kamery ma bardzo duże znaczenie dla gęstości i dokładności mapy głębokości,
- Sceny zawierające obiekty o słabej, jednorodnej strukturze są rekonstruowane w niewielkim stopniu, głównie odtworzone zostają ich krawędzi i rogi,
- Szklane powierzchnie wprowadzają silne zakłócenia do chmury punktów spowodowane odbiciami światła, widzianymi w różny sposób z lewej i prawej kamery,
- System stereowizyjny nadaje się do pracy zarówno w budynkach jak i w otwartych przestrzeniach.

3.3 Analiza złożoności obliczeniowej

System wykonany na platformę Android składa się z kilku modułów, a wydajność każdego z nich wpływa w innym stopniu na ostateczną szybkość działania systemu oraz wprowadzane opóźnienia. W tym podrozdziale dokonano porównania czasu działania najważniejszych komponentów systemu, zostały wskazane „wąskie gardła" oraz propozycję poprawy wydajności każdego z nich.

Jako pierwszy element systemu stanowiący duże obciążenie można wyróżnić część odpowiedzialną za pobieranie obrazów z kamer UVC. Jednoczesne pobieranie 2 obrazów w rozdzielczości 640 x 480, bez kompresji, w przestrzeni YUYV, z szybkością 10 klatek na sekundę co generuje wykorzystanie pasma równe około 70 Mbit/s oraz duże zużycie CPU urządzenia Raspberry Pi. Na rysunku 3.12 przedstawiono zależność procentowej zajętości CPU od liczby klatek na sekundę pobieranych z kamer. Korzystając z dwóch kamer nie było możliwe uzyskanie więcej niż 20 klatek na sekundę, pomimo przeładowania modułu `uvcvideo` z parametrami pozwalającymi na oszacowanie przepustowości przez sterownik a nie kamerę, która może żądać wyższej przepustowość niż jest rzeczywiście potrzebna. Warto mieć na uwadze, że standard USB 2.0 pozwala aby synchroniczne transmisje zajmowały nie więcej niż 80 % maksymalnej przepustowości pojedynczego hosta. Okazuje się, że obciążenie CPU rośnie szybciej niż liniowo wraz ze wzrostem liczby klatek na sekundę pochodzących z dwóch kamer USB. Pomimo teoretycznej zajętości łącza wynoszącej około 140 Mbit/s nie było możliwe uzyskanie większych przepływności. Zajętość procesora przy 20 klatkach na sekundę była bardzo duża (70 %) pomimo braku obciążeń pochodzących od innych aplikacji. Mając na uwadze, że konieczne jest wykorzystanie mocy obliczeniowej do usuwania zniekształceń toru optycznego i rektyfikacji obrazów, zdecydowano się na pracę kamer z szybkością 10 ramek na sekundę.



Rys 3.12 Zależność obciążenia CPU Raspberry Pi od liczby klatek na sekundę pobieranych z dwóch kamer UVC w rozdzielczości 640x480, w formacie YUVY

Przekształcenie usuwające zniekształcenia toru optycznego i dokonujące rektyfikacji może zostać wykonane jednocześnie, jednak z uwagi na obecność tylko jednego rdzenia w procesorze Raspberry Pi zdecydowano się na wykonanie tej procedury dla lewego i prawego obrazu w tym samym wątku. Okazuje się, że przy 10 klatkach pobieranych z kamery czas przetwarzania obrazów wynosi około 130 milisekund, co daje około 8 przetworzonych ramek na sekundę, dlatego zastosowanie wyższego FPS nie poprawia ostatecznej wydajności, a może ją pogorszyć poprzez wprowadzenie dodatkowego obciążenia CPU.

Pobieranie wstępnie przetworzonych par obrazów z Raspberry Pi na telefon odbywa się dzięki wykorzystaniu techniki *Ethernet over USB* opisanej w drugim rozdziale tej pracy. Okazuje się, że przepływności uzyskiwane dla telefonu HTC HD2 wynoszą nie więcej niż 25 Mbit/s, generując przy tym obciążenie CPU nie większe niż 15 %. Tak niska wydajność nie jest powodowana przez dużą zajętość Raspberry Pi. Podczas kopiowania dużych plików między tymi urządzeniami maksymalne osiągane

przepływności były ok. 10 % większe niż w przypadku wymiany obrazów opartej na programie ncat, którego zasadę działania przedstawiono w rozdziale drugim tej pracy. Jeden zestaw skalibrowanych obrazów zajmuje 600 KB, co przy osiąganych szybkościach transmisji danych rzędu 25 Mbit/s pozwala przesłać ok. 6 ramek na sekundę. Wykorzystanie bezprzewodowych standardów z rodziny 802.11 b/g pozwala na uzyskanie takich samych przepływności, ale ze względu na problemy z utrzymaniem połączenia w dłuższym okresie czasu zrezygnowano z wykorzystania tego rozwiązania. Jedynym sposobem mogącym polepszyć wydajność jest wykorzystanie innego modelu telefonu, z szybszym procesorem i bez ograniczeń przepływności takich jak w modelu HTC HD2.

Proces rekonstrukcji 3D na podstawie zrektyfikowanych obrazów składa się z wywołania po sobie trzech funkcji: `CvFindStereoCorrespondenceBM`, `cvConvertScale`, `cvReprojectImageTo3D`. Obciążenie procesora telefonu podczas działania każdej z nich było bliskie 100%. Czas działania tych funkcji jest zależny wielu czynników, ale okazuje się, że czas działania dwóch ostatnich nieznacznie zmienia się w czasie, pomimo wykorzystania map głębokości o skrajnie różnych parametrach jako danych wejściowych. Tabela 3-1 przedstawia czasy działania funkcji `CvFindStereoCorrespondenceBM` wyrażone w milisekundach w zależności od parametrów `NumberOfDisparities` (liczba przeszukiwanych wartości dysparycji) oraz `SADWindowSize` (rozmiar okna porównywania grup pikseli). Kolorem czerwonym oznaczono te wartości, dla których empiryczna ocena jakości mapy głębokości jest nieakceptowana.

Tabela 3-1 Zależność czasu działania `CvFindStereoCorrespondenceBM` w zależności od rozmiaru okna porównywania i liczbie stopni swobody wartości dysparycji

NumerOfDisparities/ SADWindowSize	**32**	**48**	**64**	**80**	**96**	**128**	**140**
11	259 ms	299 ms	362 ms	515 ms	700 ms	926 ms	1048 ms
13	325 ms	377 ms	488 ms	674 ms	821 ms	1097 ms	1275 ms
17	384 ms	491 ms	656 ms	**770 ms**	995 ms	1203 ms	1460 ms
19	473 ms	530 ms	711 ms	852 ms	1089 ms	1314 ms	1607 ms
21	531 ms	612 ms	793 ms	964 ms	1234 ms	1450 ms	1720 ms
23	597 ms	757 ms	904 ms	1117 ms	1330 ms	1512 ms	1812 ms

Zielony kolor oznacza najkrótszy czas działania funkcji `CvFindStereoCorrespondenceBM`, w którym możliwe jest uzyskanie mapy głębokości o akceptowalnej dokładności i gęstości. Uzyskany czas działania na telefonie (770 ms), jest około 12 razy dłuższy niż czas wykonywania się tego samego algorytmu na komputerze o procesorze z dwukrotnie większym taktowaniem, ale w architekturze x86. Dzięki zastosowaniu zbioru instrukcji SSE2 było możliwe uzyskanie tak dużego przyśpieszenia działania: ta sama procedura skompilowana w wersji bez SSE2 wymaga ok. 300 ms (pomimo zrównoleglenia części obliczeń), czyli niewiele ponad dwa razy szybciej niż na telefonie z procesorem w architekturze ARM. Tymczasowym rozwiązaniem poprawiającym wydajność implementacji `CvFindStereoCorrespondenceBM` w systemie Android jest wykorzystanie wielordzeniowego procesora o wyższym taktowaniu, tak aby na potrzeby tej funkcji możliwe było użycie kliku jednostek obliczeniowych. Ze względu na rozwój urządzeń mobilnych zbudowanych na procesorach ARM, można spodziewać się, że w najbliższej przyszłości algorytmy optymalizowane pod architekturę x86 (np. techniki SSE) zostaną dostosowane do architektury ARM (np. NEON). Wykonanie funkcji `cvConvertScale` oraz `cvReprojectImageTo3D` zajmuje po około 50 milisekund. Tak długi czas działania wynika z wykonywania serii operacji na liczbach zmiennoprzecinkowych, co w przypadku braku koprocesora FPU znacznie wydłuża czas obliczeń. Jako propozycję poprawy wydajności tych metod można zaproponować konwersję liczb typu `float` do `int`, ale kosztem utraty części informacji i nakładu związanego z rzutowaniem typów. Kolejnym rozwiązaniem mogącym skrócić czas działania tych funkcji jest implementacja algorytmu wielowątkowego mnożenia macierzy pod warunkiem wykorzystania CPU umożliwiającego jednoczesne działanie przynajmniej dwóch wątków.

Sumaryczny czas działania trzech funkcji niezbędnych do uzyskania chmury punktów wynosi 870 ms na klatkę, co oznacza wydajność systemu rzędu 1,2 FPS. Dopóki nie osiągnie się szybkości tej części rzędu 6 FPS, optymalizacja części związanych z pobieraniem danych z Raspberry Pi i wstępnym przetwarzaniem obrazów z kamer jest bezzasadna. Obecnie pojawia się na rynku urządzeń mobilnych coraz więcej producentów oferujących telefony czy tablety z procesorami z rodziny x86 (np. Intel Atom) działające z systemem Android, dlatego jedną z szans na poprawę wydajności algorytmów do rekonstrukcji 3D może być wykorzystanie takiego

urządzenia.

Wnioski końcowe

W przedstawionej pracy zaprezentowano matematyczny model układu stereowizyjnego oraz dokonano przeglądu i analizy współczesnych technik rekonstrukcji 3D. Zostały zaimplementowane algorytmy służące do kalibracji kamer oraz do uzyskania struktur przestrzennych na podstawie dwóch różnych rzutów tej samej sceny.

Główną część pracy stanowiło zaprojektowanie architektury mobilnego systemu i implementacja algorytmów pierwotnie napisanych dla komputera klasy PC. Została wykonana analiza kilku rozwiązań programowo-sprzętowych umożliwiających zbudowanie przenośnego systemu do estymacji mapy głębokości wraz z uzasadnieniem wyboru poszczególnych komponentów. W skład zbudowanego mobilnego systemu stereowizyjnego wchodzą: dwie kamery z interfejsem USB, mikrokomputer Raspberry Pi i telefon z systemem Android. Ze względu na brak wbudowanych akumulatorów w urządzeniu Raspberry Pi, do zasilania systemu konieczne jest użycie zewnętrznego źródła energii, tzw. *power bank*. Jest to nieduży, przenośny akumulator dedykowany do ładowania telefonów i innych urządzeń mobilnych. Jego wykorzystanie nie zwiększa znacząco masy czy wymiarów całości układu, a dodatkowo pozwala na ładowanie telefonu, który przy niemal 100% obciążeniu rozładowałby się w bardzo krótkim czasie. Kolejnym krokiem rozwoju systemu mogłoby być wykonanie okularów zawierających wbudowane kamery wykorzystane podczas badań i dopasowanie ich kształtu w taki sposób umożliwiający swobodne użycie przez osobę niewidomą.

Zaimplementowany został algorytm 8-punktowy służący do estymacji macierzy fundamentalnej systemu stereowizyjnego, która może posłużyć do podtrzymywania kalibracji kamer. Aby zminimalizować bądź wykluczyć błędy wprowadzane przez niedokładność obliczeń numerycznych, wykorzystano bibliotekę MPFR pozwalającą na użycie liczb zmiennoprzecinkowych o arbitralnie wybranej liczbie bitów mantysy. Zostało sprawdzone, że w przypadku estymacji macierzy fundamentalnej na podstawie par odpowiadających sobie punktów wykorzystanie zmiennych typu `double` o 53 bitach mantysy nie jest wystarczające do otrzymania dokładnego wyniku. Dekompozycja macierzy fundamentalnej na składowe translacji i rotacji jest zadaniem podatnym na błędy: niewielka zmiana parametrów macierzy fundamentalnej powoduje znaczne różnice w otrzymanych macierzach przesunięcia i obrotu. Aby otrzymać

dokładną informację o położeniu kamer w układzie stereowizyjnym, konieczna jest znajomość elementów macierzy fundamentalnej z błędem względnym nie większym niż 10^{-14}, gdy wynosi ona 10^{-12} to wyznaczone macierze T i R są obarczone kilkunastoprocentowym błędem.

W pracy zaprezentowano autorski algorytm służący do regulacji wzmocnienia i czasu ekspozycji kamery oraz pokazano jak istotnym czynnikiem jest adaptacja kamery do oświetlenia sceny. Niedoświetlone bądź prześwietlone obiekty nie są rekonstruowane ze względu na niską jakość tekstury. Dzięki implementacji tego algorytmu możliwa jest rekonstrukcja scen zarówno scen wewnątrz jak i na zewnętrz budynków. Uzyskane mapy głębokości pozwalają stwierdzić, że wykorzystanie algorytmów typu *block-matching* pozwala na dokładne zrekonstruowanie obiektów o zróżnicowanej teksturze, natomiast dla płaskich i jednorodnych struktur możliwe jest odtworzenie tylko ich konturów. W większości wykonanych nagrań uzyskane wyniki pozwalały na rekonstrukcję struktury otoczenia umożliwiającą rozpoznanie potencjalnych przeszkód i zagrożeń. W zbudowanym układzie, przy odległości między kamerami równej 10 cm, obliczone położenie punktów znajdujących się ok. 5 metrów od zestawu kamer jest obarczone 10 % błędem, natomiast dla obiektów znajdujących się ok. 70 cm od zestawu kamer nie jest wymagana obecnie uzyskiwana dokładność rzędu kilku milimetrów. W celu bardziej precyzyjnej estymacji położenia punktów znajdujących się w większych odległościach od użytkownika należałoby zwiększyć rozstaw kamer, mając na uwadze fakt, że budowa systemu nie może przeszkadzać w swobodnym poruszaniu się użytkownika.

Analiza wydajności poszczególnych elementów systemu pozwoliła na zidentyfikowanie najwolniej działających modułów, znalezienie przyczyny niewystarczającej szybkości działania oraz sformułowanie propozycji optymalizacji systemu. W ramach tej pracy wykonano prototyp mobilnego układu stereowizyjnego, którego szybkość przetwarzania wynosi 1,2 FPS. Dzięki wykorzystaniu wydajniejszych urządzeń oraz optymalizacji wybranych algorytmów na architekturę ARM polegających np. na wykorzystaniu technik SIMD możliwy będzie wzrost szybkości przetwarzania do 5-6 FPS co powinno być wystarczającym wynikiem aby pomóc w nawigacji osobom niewidomym.

Literatura

[1] Ahlvers U., Zoeler U., Rechmeier S., *FFT-based disparity estimation for stereo images coding*, International Conference On Image Processing, 2003

[2] Bartoli A., Olsen S.I., Wykłady z przedmiotu: *3D Computer Vision*, wykład p.t. *Camera Motion From Two-View Relationships*, *http://isit.uclermont1.fr/~ab/Classes/DIKU-3DCV2/*

[3] Bradski G. , Kaehler A., *Learning OpenCV*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

[4] Castaneda V., Novab N., *Time-of-Flight and Kinect Imaging*, wykład w ramach przedmiotu: *Kinect Programming for Computer Vision,* Technische Universität München, 2011

[5] Cyganek B., Siebert J. P., *An introduction to 3d computer vision techniques and algorithms*, 2009 John Wiley & Sons, Ltd

[6] Datta A., Kim J., Kanade T., *Accurate Camera Calibration using Iterative Refinement of Control Points*, Workshop on Visual Surveillance (VS) 2009, październik 2009

[7] Dokumentacja biblioteki GNU MPFR, *http://www.mpfr.org/mpfr-current/mpfr.pdf*

[8] Dokumentacja biblioteki Eigen, *http://eigen.tuxfamily.org/dox/*

[9] Dokumentacja biblioteki OpenCV: *http://opencv.willowgarage.com/documentation/cpp/index.html*

[10] Dokumentacja interfejsu V4L2, *http://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single*

[11] Dokumentacja systemu Raspbian, *http://www.raspbian.org/RaspbianDocumentation*

[12] Dokumentacja biblioteki Qt, *http://doc.qt.nokia.com/4.7/index.html*

[13] Droppelmann S., Hueting M. Latour S., Van der Veen M., *Stereo Vision using the OpenCV library*, czerwiec 2010

[14] Gargenta M., *Getting native with NDK*, *https://marakana.com/s/post/1342/GettingNativeWithNDK.pdf*

[15] Hartley R., *In the defense of 8-point algorithm*, IEEE transaction on pattern analysis and machine intelligence, vol. 19, no. 6, czerwiec 1997

[16] Hartley R., Zisserman A., *Multiple View Geometry in computer vision*, Cambridge University Press 2000, 2003

[17] Hoeim D., *How The Kinect Works,* wykład w ramach przedmiotu: *Computational Photography*, University of Illinois, 2011

[18] Kielczewski M., *Wykład dot. kalibracji kamer*, Politechnika PoznańskaWydział Informatyki ,Katedra Sterowania i Inżynierii Systemów, 2011

[19] Kurzawski K., praca inżynierska: *Program komputerowy do pomiaru odległości w przestrzeni 3D na podstawie wielu sekwencji 2D*, Kraków, styczeń 2012

[20] Ndhlovu T. , praca magisterska*: An investigation into stereo algorithms: An emphasis on local-matching*, Cape Town, marzec 2011

[21] Nourani-Vatani N., Roberts J., *Automatic Camera Exposure Control*, Proceedings of the 2007 Australasian Conference on Robotics & Automation, Australia grudzień, 2007

[22] Szeliski R., Zabih. R, *An Experimental Comparison of Stereo Algorithms*, ICCV '99 Proceedings of the International Workshop on Vision Algorithms: Theory and Practice, 1999
[23] Thirtala S. R., Pollefeys M. , *The Radial Trifocal Tensor: A tool for calibrating the radial distortion of wide-angle cameras*, CVPR '05 Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005
[24] Witryna www: *http://www.mirametrics.com/tech_note_ccdgain.htm*, marzec 2013
[25] Witryna www: *http://code.google.com/p/libcam/*
[26] Witryna *http://en.wikipedia.org/wiki/File:Jamtlands_Flyg_EC120B_Colibri.JPG*

Dodatek A. Zawartość dołączonej płyty CD-ROM

Poniżej przedstawiono strukturę katalogów dołączonej płyty CD-ROM:

/src_PC
Katalog zawierający kody źródłowe i nagłówki wszystkich napisanych przez autora tej pracy klas dla systemu dla PC (z Qt4), oraz inne niezbędne pliki projektowe.

/src_Android_Pi
Katalog zawierający kody źródłowe i nagłówki wszystkich napisanych przez autora tej pracy klas dla system dla platformy Android i Raspberry Pi, oraz inne niezbędne pliki projektowe.

/run
Plik zawierający skrypty uruchomieniowe programów dla Android i Raspberry Pi

/reports
Katalog zawierający obrazy w oryginalnych rozdzielczościach, których miniatury wykorzystano w rozdziałach 2 i 3 oraz inne obrazy pochodzące z nagrań wykonanych w ramach tej pracy

/Zawartość.txt
Plik tekstowy, w którym znajduje się opis zawartości płyty CD-ROM.

/Praca_mgr_Konrad_Kurzawski.pdf
Dokument *.pdf* zawierający treść pracy dyplomowej.

Dodatek B. CD-ROM