**Okulografia – śledzenie ruchu gałki ocznej**



Patrycja Śliwińska

Poznań 2018

Spis treści

[1. Uzasadnienie wyboru tematu 3](#_Toc503049071)

[2. Podział prac 5](#_Toc503049072)

[3. Funkcjonalności aplikacji 6](#_Toc503049073)

[4. Wybrane technologie wraz z uzasadnieniem dlaczego? 7](#_Toc503049074)

[5. Architektura rozwiązania (jak jest zbudowane) 8](#_Toc503049075)

[6. Interesujące problemy i rozwiązania ich na jakie się natknęliście 12](#_Toc503049076)

[7. Instrukcja użytkowania aplikacji 17](#_Toc503049077)

[8. Bibliografia 18](#_Toc503049078)

[9. Spis zdjęć 20](#_Toc503049079)

# Uzasadnienie wyboru tematu

Tematem niniejszego projektu jest okulografia (ang. *eye tracking*) polegająca na śledzeniu ruchu gałki ocznej. W związku z tym w ramach projektu został stworzony program rejestrujący ruchy źrenicy badanej osoby.

Na wybór takiego tematu wpłynęło wiele czynników. Niewątpliwie najważniejszym z nich było wykorzystanie okulografii do przeprowadzania wszelkiej maści badań naukowych, zaczynając od różnego rodzaju badań marketingowych a kończąc na pracach badawczych. Sama okulografia swoje początki odnotowuje w XIX w., gdzie w miarę upływu lat, cieszyła się coraz większą popularnością. Jako, że wzrok kształtuje wszystkie nasze zachowania, okulografia odnalazła zastosowanie we wszelkich dziedzinach m. in. reklamie, psychologii, informacji publicznej. Dzięki niej jesteśmy wstanie określić który produkt skupia naszą uwagę, gdzie w pierwszej kolejności kierujemy wzrok po wejściu do danego pomieszczenia oraz czy dana aplikacja/informacja została dobrze zaprojektowania/zlokalizowana w celu szybkiego pozyskania oczekiwanych informacji przez odbiorników.

Obecnie okulografia znajduje zastosowanie nie tylko w badaniach, ale także w różnego rodzaju terapiach, które przyśpieszają powrót do normalnego funkcjonowania. Dodatkowo ta technologia jest wykorzystywana w pewnych badaniach medycznych dzięki którym można szybciej postawić prawidłową diagnozę. Swoje zastosowanie odnajduje m. in. w terapii ADHD, zaburzeniach ze spektrum autyzmu oraz porażeniu mózgowym. Na pomoc ze strony okulografii mogą liczyć również pacjenci cierpiący na dysleksję oraz po udarach i urazach wywołujących zaburzenia prawidłowego funkcjonowania wzroku. Technologia ta jest również wykorzystywana do komunikowania się przez osoby niepełnosprawne. Dzięki okulografii, osoby z porażeniami, mogą w łatwy sposób sterować komputerem oraz korzystać z różnych aplikacji umożliwiających prowadzenie aktywnego życia.

Okulografia nie jest jedynie technologią „dzisiaj”, ale również „jutra”. Wiele dziedzin nauki nadal stoi przed nią otworem. Dużą nadzieję pokłada się w wykorzystaniu okulografii przy oszczędzaniu energii. W obecnych czasach, gdzie energia staje się powoli towarem deficytowym, staramy się ograniczać zużycie energii do minimum. Dzięki okulografii bylibyśmy wstanie odpowiednio sterować parametrami wyświetlaczy, tak aby rozdzielczość elementów, na które w danej chwili spoglądamy, stawałaby się większa, w stosunku do pozostałej części nieobserwowanego wyświetlacza. Dzięki temu, kosztem gorszej jakości obrazu w obszarach nierejestrowanych przez nasz wzrok, bylibyśmy wstanie zaoszczędzić energię a w wyniku tego, wydłużyć czas pracy naszych urządzeń. Z dużą nadzieją spogląda się również na wykorzystanie okulografii w wirtualnej rzeczywistości. Dzięki wykorzystaniu tej technologii w okularach VR można by tworzyć w obserwowanych miejscach, tekstury o wysokiej rozdzielczości, co przełożyłoby się na wzrost wydajności sprzętu i ograniczenie zużycia mocy obliczeniowej urządzenia.

# Podział prac

Projekt podczas realizacji podzielono na kilka etapów:

* budowa odpowiedniej aparatury do rejestracji obrazów,
* kalibracja kamery górnej (przymocowanej na czole),
* zapis rejestrowanych obrazów do plików .avi,
* wykrywanie gałki ocznej oraz zczytywanie jej pozycji do pliku .csv,
* skalowanie kamer względem osoby realizującej badanie,
* wizualizacja otrzymywanych wyników.

Ze względu na jednoosobowy zespół projektowy, całość prac została zrealizowana przez autora projektu (z pominięciem budowy aparatury wykorzystywanej do rejestrowania obrazów).

# Funkcjonalności aplikacji

Program ten został stworzony do celów badawczych. Głównym zadaniem aplikacji jest śledzenie ruchu gałki ocznej badanej osoby. Dzięki niej mamy możliwość zarejestrowania miejsc, na które spogląda badana osoba w trakcie przeprowadzanego badania. Niniejsza aplikacja umożliwia:

* rejestrowanie obrazu z obu kamer,
* zapisywanie pozycji źrenicy ze wskazaniem numeru klatki,
* wskazywanie obiektów, na które spoglądała w danej chwili badana osoba.

# Wybrane technologie wraz z uzasadnieniem dlaczego?

Aplikacja została stworzona w języku C/C++. W trakcie realizacji projektu skorzystano z następujących bibliotek:

* OpenCV,
* fstream.

Język C/C++ został wybrany ze względu na najlepszą znajomość przez autora projektu oraz ogólnie szeroko dostępną dokumentację.

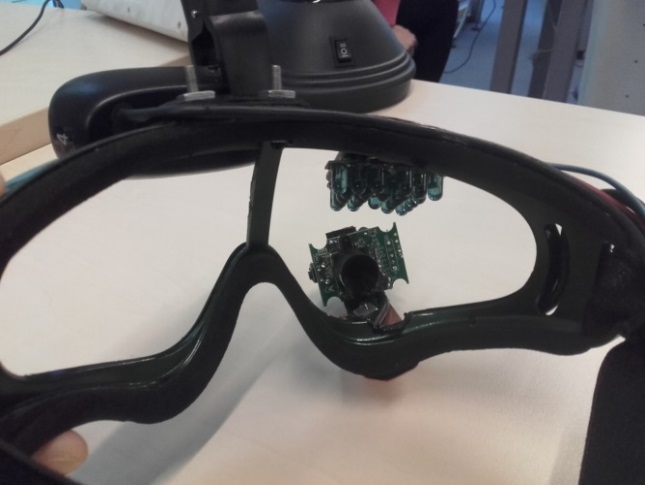
Biblioteka OpenCV jest wykorzystana do obsługi kamer. Dzięki niej w łatwy sposób możemy przechwycić strumień danych oraz dokonywać wszelkiej edycji obrazu w czasie rzeczywistym. Natomiast biblioteka fstream została użyta w celu uzyskania możliwości operowania na plikach.

# Architektura rozwiązania (jak jest zbudowane)

Prace nad projektem rozpoczęły się od budowy urządzenia służącego do rejestrowania obrazów. Zdjęcia aparatury przedstawiają: Zdjęci 1. oraz Zdjęcie 2



Zdjęcie 1. Widok z przodu



Zdjęcie 2. Widok z tyłu

W skład aparatury pomiarowej wchodzą dwie kamery przymocowane do oprawek okularów. Jedna z nich została umieszczona na górze okularów. Dzięki niej uzyskujemy obraz otoczenia, który jest obserwowany przez badanego. Druga z kamer znajduje się w prawej dolnej części okularów. Jest ona pozbawiona filtra tak aby, mogła rejestrować podczerwień. Dodatkowo została tak skierowana, aby rejestrowała ruchy źrenicy. Powyżej tej kamery są zamontowane diody podczerwieni IR 850 nm, które świecą bezpośrednio na gałkę oczną. Dzięki temu na obrazie rejestrowanym przez kamerę skierowaną w kierunku gałki ocznej jest widoczna tylko źrenica. Obie kamery oraz diody IR zostały podpięte do hub’a USB, który został podłączony do wejścia USB laptopa.

Po podłączeniu sprzętu do laptopa uruchamiamy aplikację.

Pierwszym krokiem w aplikacji jest rejestracji obrazów dla dwóch kamer internetowych ([1]).

W tym celu dokonujemy otwarcia dwóch strumieni przechwytywania danych dla każdej z kamer osobno([2]):

VideoCapture kanalczolo, kanaloko; //Obiekty, w których przetrzymujemy dane dla obu kamer

kanalczolo.open(0); //Otwieranie strumienia przechwytywania danych

kanaloko.open(1);

Dla każdej kamery stworzono obiekt typu Mat, w którym są przetrzymywane klatki filmu:

Mat img0, hsv\_img0, binary, czolo; //Miejsce na klatki

Następnie dla obu kamer zapisujemy wymiary kadrów, w celu późniejszego odtworzenia nagrań:

int frame\_width0 = kanalczolo.get(CV\_CAP\_PROP\_FRAME\_WIDTH); //wymiary do zapisu

int frame\_height0 = kanalczolo.get(CV\_CAP\_PROP\_FRAME\_HEIGHT);

int frame\_width1 = kanaloko.get(CV\_CAP\_PROP\_FRAME\_WIDTH);

int frame\_height1 = kanaloko.get(CV\_CAP\_PROP\_FRAME\_HEIGHT);

Dalej definiujemy kodeki I tworzymy obiekt dla każdego z nagranych plików .avi:

VideoWriter video0("Czolo.avi", CV\_FOURCC('M', 'J', 'P', 'G'), 5, Size(frame\_width0, frame\_height0));

VideoWriter video1("Oko.avi", CV\_FOURCC('M', 'J', 'P', 'G'), 5, Size(frame\_width1, frame\_height1));

VideoWriter video2("Czolo\_z\_okregami.avi", CV\_FOURCC('M', 'J', 'P', 'G'), 5, Size(frame\_width0, frame\_height0));

VideoWriter video3("Analiza.avi", CV\_FOURCC('M', 'J', 'P', 'G'), 5, Size(frame\_width0, frame\_height0));

W celu dopełnienia formalności definiujemy kanał do obsługi plików i otwieramy plik, do którego będą zapisywane poszczególne położenia źrenicy oka:

fstream wyniki;

wyniki.open("Wyniki.csv", ios::out); //Tworzenie pliku z wynikami

wyniki << "ramka; x; y; r" << endl;

Po stworzeniu niezbędnych narzędzi przechodzimy do samej aplikacji. Jednakże zanim przystąpimy do rejestracji obrazów musimy przeprowadzić skalowanie tak, aby odczytana współrzędna źrenicy pokrywała się ze współrzędną obiektu, na który w danym momencie patrzymy. Działanie to jest konieczne ze względu na „niestabilność” aparatury oraz indywidualne uwarunkowania badanych jednostek. Poniżej znajduje się przykładowe skalowanie, przy którym dokonano pomiarów załączonych wyników:

if ((srx > 320) && (srx<350)) srx = 1.1\*srx;

if ((srx > 215)&& (srx<250)) srx = 0.8\*srx;

if ((srx > 480) && (srx<550)) srx = 1.2\*srx;

if (sry < 200) sry = 0.82\*sry;

if ((sry > 200) && (sry < 238)) sry = 1.05\*sry;

if ((sry > 249)&& (sry<329)) sry = 1.15\*sry;

if (sry > 329) sry = 1.35\*sry;

Po odpowiednim dobraniu parametrów przesunięć współrzędnych gałki ocznej przechodzimy do rejestrowania obrazów.

Nagrywany film odczytujemy jako sekwencja, odtwarzanych klatka po klatce, zdjęć np.

kanaloko >> RamkaOko;

Ze względu na zamontowanie kamery rejestrującej obraz gałki ocznej w pozycji „naturalnej”, należy dokonać rotacji obrazu tak, aby wyświetlała zarejestrowany obraz w pionie. W tym celu wykorzystano funkcję:

Mat MacierzRotacji = getRotationMatrix2D(Point(RamkaOko.cols / 2, RamkaOko.rows / 2), 260, 1);

Mat RamkaRotacji;

warpAffine(RamkaOko, RamkaRotacji, MacierzRotacji, RamkaOko.size());

RamkaOko = RamkaRotacji;

W celu znalezienia położenia gałki ocznej, a ściślej mówiąc, położenia źrenicy skorzystano z wbudowanej funkcji HoughCircles. Funkcja ta najpierw przekształca obraz na odcień szarości a następnie próbuje znaleźć okręgi. Zabieg ten wykonywany jest w celu wyeliminowania fałszywych okręgów.

Mat src\_gray;

cvtColor(img0, src\_gray, CV\_BGR2GRAY);

GaussianBlur(src\_gray, src\_gray, Size(9, 9), 2, 2); // Zmniejszenie hałasów w celu uniknięcie wykrywania fałszywych okręgów

vector<Vec3f> circles;

HoughCircles(src\_gray, circles, CV\_HOUGH\_GRADIENT, 1, src\_gray.rows / 8, 200, 20, 0, 0); // Wykonanie transformacji Hough w celu wykrycia okręgów

Po znalezieniu położenia okręgu, jego współrzędne są zapisywane do zewnętrznego pliku:

Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));

int radius = cvRound(circles[i][2]);

wyniki << nr << "; " << 640-cvRound(circles[i][0]) << "; " << cvRound(circles[i][1]) << "; " << cvRound(circles[i][2]) << endl;

Po zakończeniu rejestracji obrazów następuje zamknięcie wcześniej kanałów i zwolnienie zasobów:

wyniki.close();

video2.release();//Zwolnienie wszystkich zasobów VideoCapture

kanalczolo.release();

video0.release();

kanaloko.release();

video1.release();

W celu przeprowadzenia analizy otrzymanych nagrań, z otrzymanego pliku wynikowego zczytujemy poszczególne pozycje źrenicy. Następnie uwzględniamy je w materiale filmowym pozyskanym z kamery znajdującej się na czole badanego.

Na początku otwieramy plik .avi z nagraniem:

CvCapture\* vid = cvCreateFileCapture("E:/Okulografia/Okulografia/Eye\_tracker/Eye\_tracker/Czolo.avi"); // Odczytanie pliku avi

double fps = cvGetCaptureProperty(vid, CV\_CAP\_PROP\_FPS); // odczytujemy z wlasciwosci pliku liczbe klatek na sekunde

int odstep\_miedzy\_klatkami = 1000 / fps; // wyliczamy czas potrzebny do odtwarzania pliku z prawidlowa prędkoscia

Dla każdej klatki filmu nanosimy pierścień, o współrzędnych odczytanych z pliku wynikowego:

analiza = cvarrToMat(ramka);

Point srodek(srx, sry);

circle(analiza, srodek, 30, Scalar(0, 255, 0), 4, 0, 0);

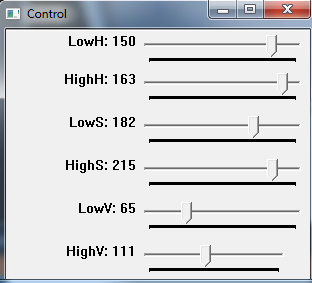
# Interesujące problemy i rozwiązania ich na jakie się natknęliście

W trakcie tworzenia aplikacji napotkaliśmy kilka problemów. Najbardziej znaczącym był problem znajdowania położenia źrenicy. Początkowo prace nad aplikacją były wykonywane bez aparatury pomiarowej. Jako substytut użyto dwóch kamer: jedną – wbudowaną fabrycznie w laptopie oraz drugą – zewnętrzną kamerę internetową podłączaną przez wejście USB. Z pierwotnych założeń otrzymywany obraz gałki ocznej miał być zbliżony do obrazu z noktowizora przedstawionego na Zdjęcie 3.



Zdjęcie 3. Zakładany obraz gałki ocznej

Wówczas namierzanie pozycji źrenicy miało się odbywać przez konwersję obrazu do HSV. Przy pomocy odpowiedniego panelu (Zdjęcie 4.) byliśmy wstanie, w czasie rzeczywistym, dobrać parametry, tak aby całkowicie wyeliminować wszelkie szumy i pozostawić jedynie obraz źrenicy.



Zdjęcie 4. Panel kontrolny przy pomocy, którego były dobierane dane w celu eliminacji szumów

W ten sposób łatwo eliminowaliśmy wszelkie zakłócenia i w efekcie otrzymywaliśmy czarne tło z „białą” okrągłą plamą identyfikującą położenie źrenicy.

RamkaOko.copyTo(img0); // Skopiowanie klatki do img

cvtColor(img0, hsv\_img0, CV\_BGR2HSV); //Konwersja do HSV

split(hsv\_img0, hsv\_split); //Podzial HSV na poszczegolne kanaly

inRange(hsv\_split[0], Scalar(150, 182, 65), Scalar(163, 215, 111), binary);

//usuwanie małych obiektów z planu

erode(binary, binary, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

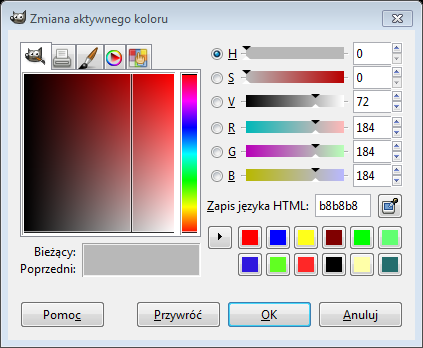
dilate(binary, binary, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

//wypełnianie małych dziur na pierwszym planie

dilate(binary, binary, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

erode(binary, binary, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

Na początku położenie źrenicy odnajdowaliśmy szukając składowych położenia danego piksela [3]. W tym celu, dokładną wartość jaką chcieliśmy szukać, byliśmy wstanie odnaleźć poprzez odczyt koloru w prostym programie graficznym (w tym przypadku został wykorzystany GIMP).



Zdjęcie 5. Odnajdowanie szukanej barwy

Sposób ten nie okazał się optymalny. Po każdorazowym zmianie natężenia oświetlenia konieczna była zmiana składowych szukanej barwy w modelu HSV.

W momencie, w którym okazało się, że stworzona aparatura pomiarowa nie zwraca wyników zbliżonych do obrazu zwracanego przez noktowizor, konieczna była zmiana podejścia.

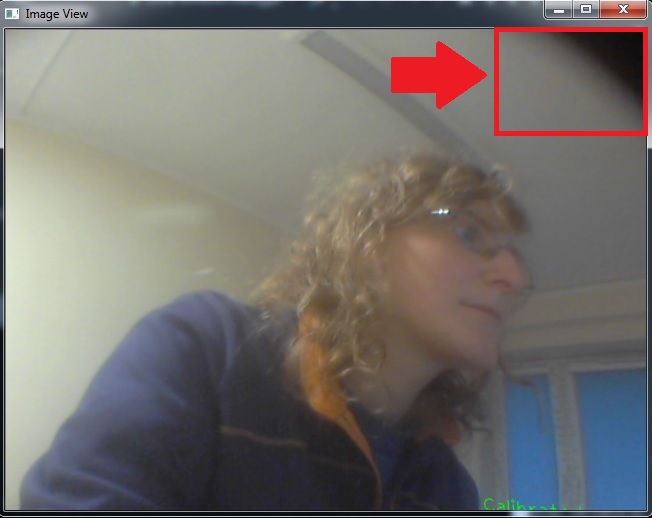
Wówczas skorzystano z podejścia wykorzystywanego w aplikacji czyli poprzez użycie wbudowanej funkcji do znajdowania okręgów [4]. Rozwiązanie to nie jest jednak idealne. Musimy bowiem wybierać pomiędzy czułością wykrywania okręgów a pewnością wykrycia źrenicy. Zdarza się bowiem, że zamiast źrenicy odczytujemy położenie odbicia diody IR w źrenicy. Jednakże wyboru odpowiednich okręgów można dokonać poprzez promień znajdowanych okręgów. Dla źrenicy są to wartości w przedziału 40-50. Natomiast dla obrazu diod wartości te są zdecydowanie mniejsze.

Kolejnym problemem jest konieczność każdorazowego skalowania aparatury badawczej. Pierwotnie dla aplikacji wykonywaliśmy kalibrację, która była przeprowadzana przy pomocy wzorcowego programu oraz tablicy szachowej 4x9 (Zdjęcie 6.).



Zdjęcie 6. Wzór, dla którego wykonywaliśmy skalowanie

W ten sposób chcieliśmy usunąć efekt rybiego oka z kamery umieszczonej na czole. Niestety, pomimo wykonania kilku prób kalibracji zakończonych sukcesem, efekt rybiego oka nie był w pełni eliminowany (Zdjęcie 7.). W związku z tym, w naszej aplikacji, proces kalibracji został pominięty.



Zdjęcie 7. Przykład nie w pełni pozytywnej kalibracji kamery

Ze względu na budowę aparatury pomiarowej, jesteśmy zmuszeni do każdorazowego przeprowadzania skalowania kamer. Jest to związane z różnym rozstawieniem gałek ocznych u ludzi. Biorąc pod uwagę standardowego badanego, różnice te są co prawda nieznaczne jednakże niewielki odchył w ruchu źrenicy może przełożyć się na odległość rzędu kilku metrów w zależności od odległości obiektów obserwowanych. Dla przedmiotów obserwowanych z niewielkiej odległości różnice te będą nieznaczne. Jednakże w przypadku obiektów oddalonych o kilkadziesiąt metrów, możemy otrzymać różnicę rzędu kilku metrów.

Kolejnym powodem, dla którego konieczne jest przeprowadzanie każdorazowo skalowania jest „kruchość” aparatury. Kamera zamontowana na czole porusza się w górę oraz w dół. W związku z tym, przy każdorazowym założeniu bądź ściągnięciu aparatury następuje przemieszczenie kamery, co wiąże się ze zmianą zakresu rejestrowanego obrazu. Ponadto kamera rejestrująca położenie źrenicy jest zamontowana na elastycznej płytce, którą można przemieszczać w czterech podstawowych kierunkach: góra, dół, lewo oraz prawo. Kamera ta, została tak zamocowana, aby w łatwy sposób można było ustawić położenie kamery względem oka, gdyż nie może ona zasłaniać obszaru widzenia badanego oraz musi rejestrować położenie oka w najbardziej skrajnych pozycjach. W związku z tym każdorazowe poruszenie tej kamery powoduje konieczność ponownego skalowania kamer. W przypadku kamery skierowanej na źrenicę, nawet niewielka różnica zmiany położenia kamery, powoduje duże odchyły od obserwowanych obiektów.

# Instrukcja użytkowania aplikacji

# Bibliografia

Podłączenie kamery:

[1] <https://thefreecoder.wordpress.com/2012/09/11/opencv-c-video-capture/>

Zapisywanie do pliku:

<https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/>

Kamery:

<http://www.codepool.biz/multiple-camera-opencv-python-windows.html>

<https://msdn.microsoft.com/en-us/library/windows/desktop/dd377566(v=vs.85).aspx>

[2] <https://putuyuwono.wordpress.com/2015/05/12/single-thread-multi-camera-capture-using-opencv/>

<https://putuyuwono.wordpress.com/2015/05/29/multi-thread-multi-camera-capture-using-opencv/>

Kalibracja:

<http://aishack.in/tutorials/calibrating-undistorting-opencv-oh-yeah/>

<http://vgl-ait.org/cvwiki/doku.php?id=opencv:tutorial:camera_calibration>

Otwieranie pliku video z kamery1:

<http://ratixu.blogspot.com/2009/03/opencv-cz-3-otwieranie-i-zapis-obrazu.html>

Wykrywanie obiektów:

<https://forbot.pl/blog/opencv-2-wykrywanie-obiektow-id4888>

<http://opencv-srf.blogspot.com/2010/09/object-detection-using-color-seperation.html>

<https://www.biomed.org.pl/eye-tracking-w-neurorehabilitacji-pl.html#breadcrumb>

Współrzędne piksela

[3] <http://rpetryniak.blogspot.com/2011/03/dostep-do-skadowych-piksela-w-opencv-22.html>

Znajdowanie okręgów

[4] <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html>

Eksport wyników

<https://www.youtube.com/watch?v=h2Taf16gQDI>

Zastosowania eyetrackingu:

<http://www.eyetracker.pl/oferta-view/obszary-badan/>

<https://www.biomed.org.pl/zastosowanie-eyetrackingu-w-terapii.html#breadcrumb>

# Spis zdjęć

[Zdjęcie 1. Widok z przodu 8](#_Toc503049365)

[Zdjęcie 2. Widok z tyłu 8](#_Toc503049366)

[Zdjęcie 3. Zakładany obraz gałki ocznej 12](#_Toc503049367)

[Zdjęcie 4. Panel kontrolny przy pomocy, którego były dobierane dane w celu eliminacji szumów 13](#_Toc503049368)

[Zdjęcie 5. Odnajdowanie szukanej barwy 14](#_Toc503049369)

[Zdjęcie 6. Wzór, dla którego wykonywaliśmy skalowanie 15](#_Toc503049370)

[Zdjęcie 7. Przykład nie w pełni pozytywnej kalibracji kamery 15](#_Toc503049371)