Okulografia – śledzenie ruchu gałki ocznej

Patrycja Śliwińska

Poznań 2018

Spis treści

1. Uzasadnienie wyboru tematu

Tematem niniejszego projektu jest okulografia (ang. *eye tracking*) polegająca na śledzeniu ruchu gałki ocznej. W związku z tym, w ramach projektu została stworzona aplikacja rejestrująca ruchy źrenicy badanej osoby.

Na wybór takiego tematu wpłynęło wiele czynników. Niewątpliwie najważniejszym z nich było wykorzystanie okulografii do przeprowadzania wszelkiej maści badań naukowych, poczynając od różnego rodzaju badań marketingowych a kończąc na pracach badawczych. Sama okulografia swoje początki odnotowuje w XIX w., gdzie w miarę upływu lat, cieszy się coraz większą popularnością. Jako, że wzrok kształtuje wszystkie nasze zachowania, okulografia znalazła 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 technologia ta 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 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 elektryczna staje się powoli towarem deficytowym, staramy się ograniczać zużycie energii do minimum. Dzięki okulografii jesteśmy 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. Kosztem gorszej jakości

obrazu, w obszarach nierejestrowanych przez nasz wzrok, jesteśmy 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.

2. 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. Aparatura badawcza została zbudowana i przekazana na czas testów, przez osobę trzecią.

3. 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.

Na chwilę obecną aplikacja nie posiada innych funkcjonalności.

4. 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.

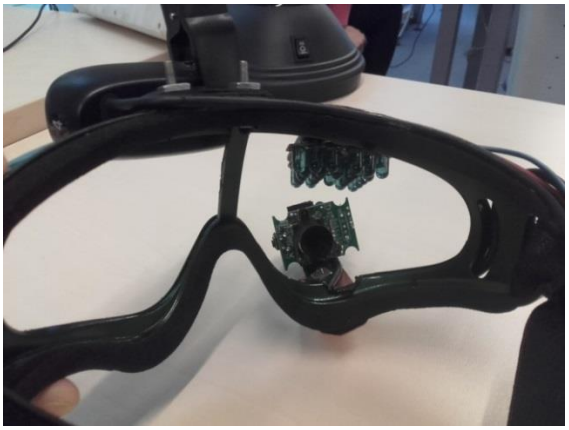
Biblioteka OpenCV jest wykorzystana do obsługi kamer. Dzięki niej w łatwy sposób możemy przechwycić strumień danych oraz jesteśmy wstanie dokonywać edycji obrazu w czasie rzeczywistym.

Natomiast biblioteka fstream została użyta w celu uzyskania możliwości operowania na plikach.

5. Architektura rozwiązania (jak jest zbudowane)

Prace nad projektem rozpoczęły się od budowy urządzenia służącego do rejestrowania obrazów. Urządzenie to zostało zaprojektowane oraz zbudowane przez osobę trzecią niebiorącą aktywnego udziału w tworzeniu kodu aplikacji. Prototyp aparatury pomiarowej przedstawiono na poniższych zdjęciach: Zdjęcie 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 23 diody podczerwieni IR 850 nm., które bezpośrednio oświetlają 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 w trakcie działania aplikacji jest podłączany do wejścia USB laptopa.

Aplikacja uruchamia się po podłączeniu aparatury badawczej do laptopa.
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 oraz 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. 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:

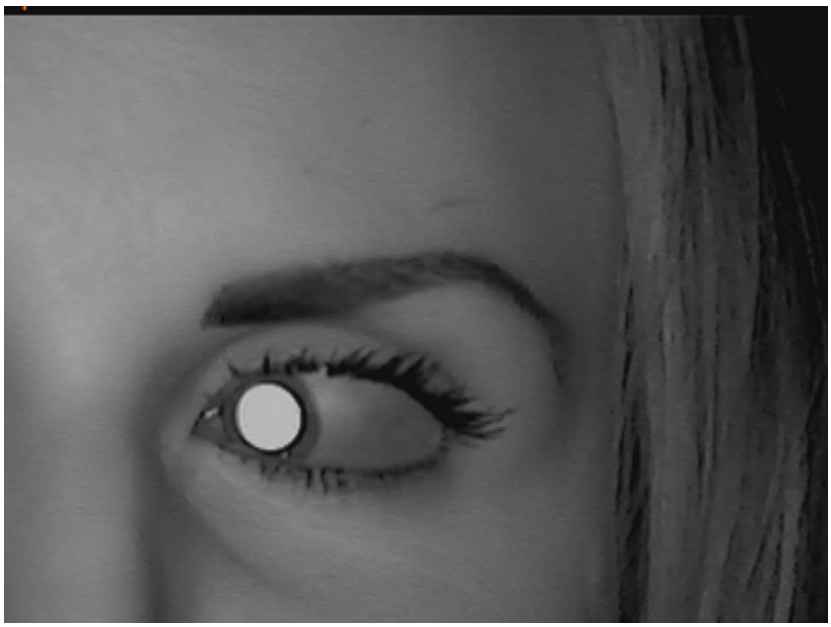
```
	CvCapture* vid = 
cvCreateFileCapture("E:/Okulografia/Okulografia/Eye_tracker/Eye_tracker/Czolo.avi");
// Odczytanie pliku avi
	double fps = cvGetCaptureProperty(vid, CV_CAP_PROP_FPS); // odczytujemy z 
wlasciwosci pliku liczbe klatek na sekunde
	int odstep_miedzy_klatkami = 1000 / fps; // wyliczamy 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);
```

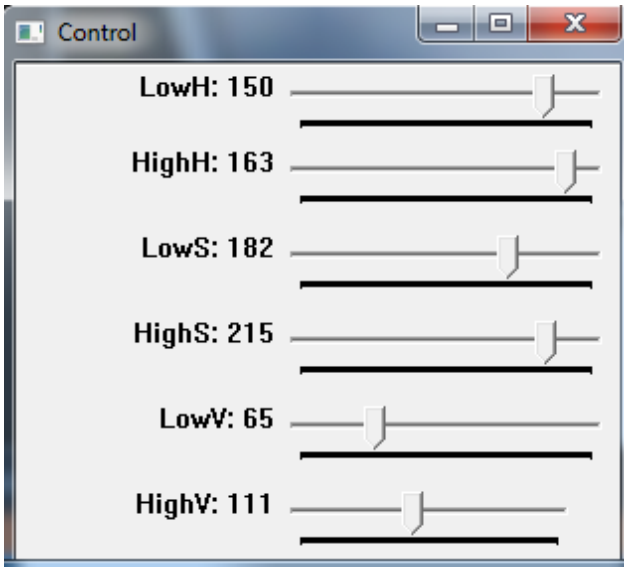
6. Interesujące problemy i rozwiązania ich na jakie się natknęliście

W trakcie tworzenia aplikacji musieliśmy zmierzyć się z kilkoma problemami. Najwiekszą trudnością okazało się namierzenie aktualnego położenia źrenicy. Początkowo prace nad aplikacją były wykonywane bez aparatury pomiarowej. Jako substytut użyto dwóch kamer: jedną – fabrycznie wbudowaną 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 oka, odbywało się 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.

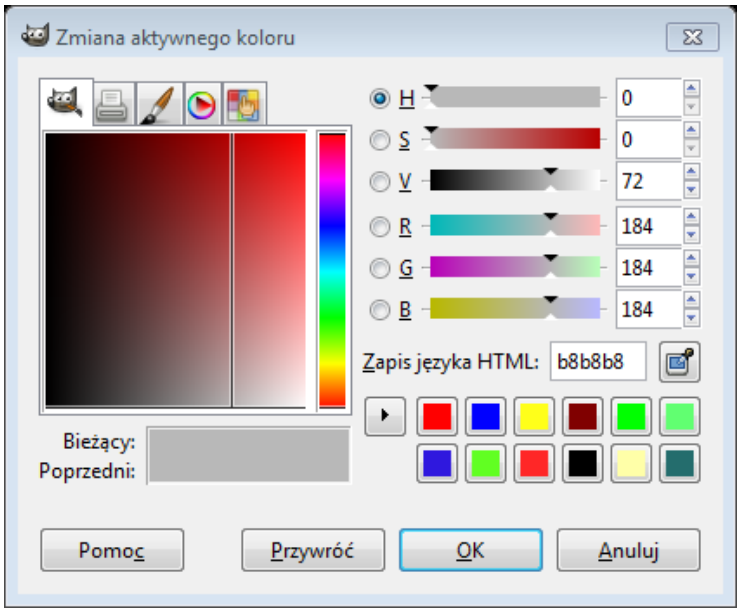


Zdjęcie 4. Panel kontrolny przy pomocy, którego były dobierane dane w celu eliminacji szumów

W ten sposób łatwo eliminowaliśmy wszelkie zakłócenia. W efekcie otrzymywaliśmy czarne tło z „białą" okrągłą plamą identyfikującą położenie źrenicy. Powyższy efekt otrzymywaliśmy za pomocą poniższej implementacji:

```
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 wyszukanie położenia źrenicy realizowaliśmy poprzez, rozłożenie zarejestrowanego obrazu oka, na pojedyncze składowe barwy danego piksela [3]. Do wyznaczenia poprawnych wartości składowych HSV szukanego piksela, skorzystaliśmy z darmowego programu graficznego GIMP. Po skorzystaniu z odpwiedniej funkcji, jesteśmy wstanie, w łatwy sposób zczytać poszczególne składowe HSV danego piksela. Przykładowy rozkład piksela na poszczególne składowe został przedstawiony na Zdjęcie 5.



Zdjęcie 5. Rozkład barwy piksela na poszczególne składowe HSV

Sposób ten nie okazał się jednak optymalny. Po każdorazowej zmianie natężenia oświetlenia, konieczne było wykonanie, korekcji poszczególnych składowych modelu HSV szukanego piksela.

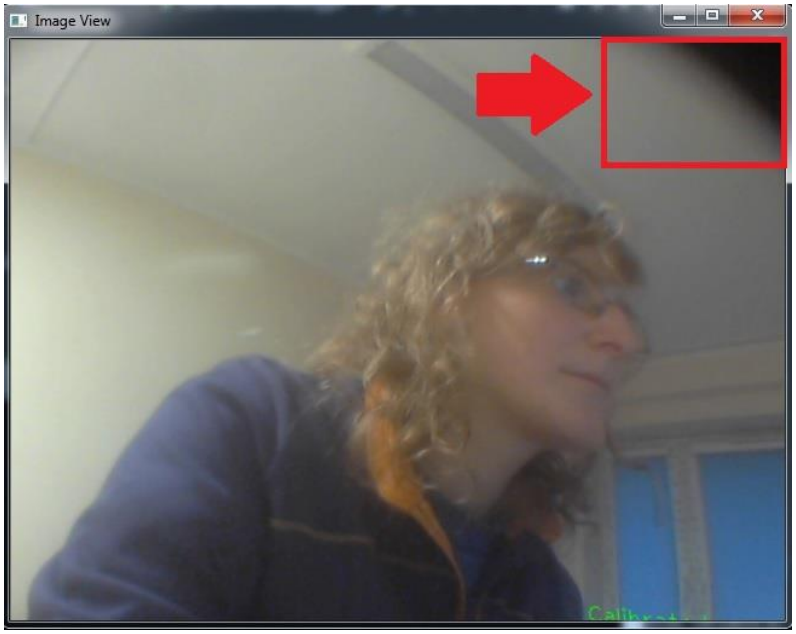
W momencie, w którym okazało się, że stworzona aparatura badawcza nie zwraca wyników zbliżonych do obrazu zwracanego przy użyciu noktowizora, konieczna była zmiana podejścia.

Wówczas skorzystano z rozwiązania, wykorzystanego w finalnej wersji aplikacji czyli poprzez użycie wbudowanej funkcji do znajdowania okręgów [4]. Rozwiązanie to nie jest jednak idealne. W celu zapewnienia odpowiedniej dokładności, jesteśmy zmuszeni do wyboru pomiędzy czułością wykrywania okręgów, a pewnością wykrycia źrenicy. Zdarza się bowiem, że zamiast położenia źrenicy, odczytujemy położenie obrazu odbitej diody IR w źrenicy oka. Ten mały mankament można odpowiednio skorygować. Wystarczy w trakcie procesu analizy wyników, dokonać sprawdzenia poprawności otrzymanych danych. Średni promień źrenicy oscyluje w wartościach z przedziału 40-50. Z tego względu, promienie rzędu jedności możemy wyeliminować nie powodując przekłamania wyników.

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 była wykonywana kalibracja

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

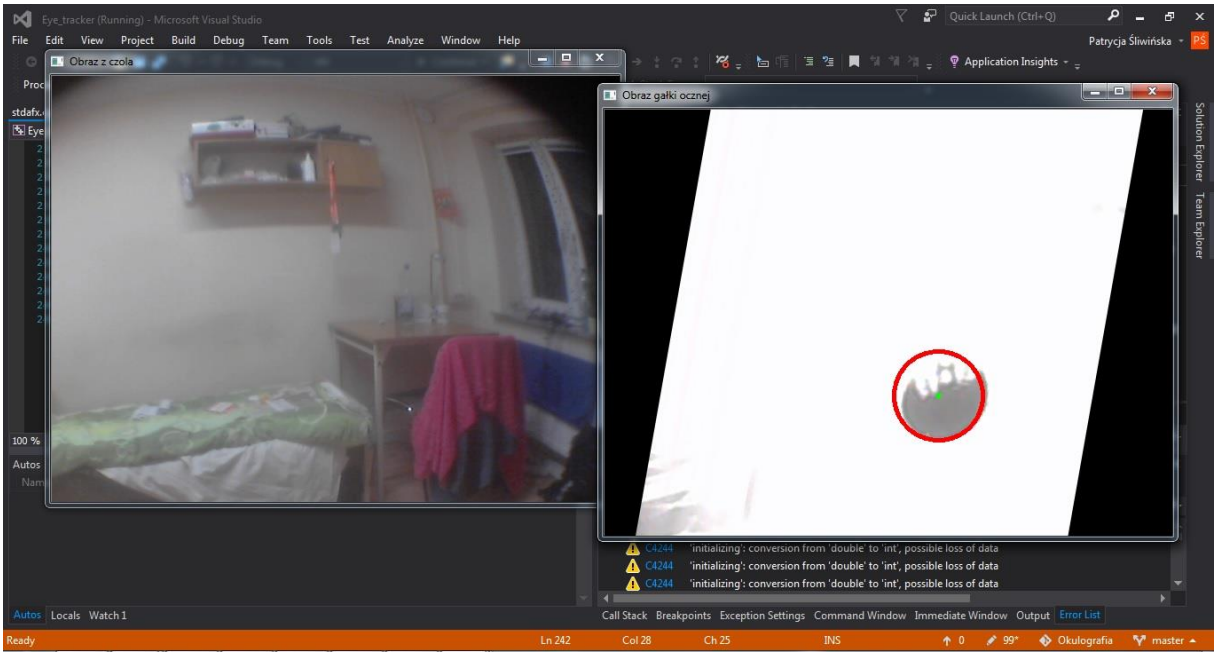
u ludzi. Biorąc pod uwagę standardowego badanego, różnice te są co prawda nieznaczne jednakże niewielki odchył w ruchu źrenicy może przełożyć się na odległość rzędu kilku metrów, w zależności od odległości obiektów obserwowanych. Dla przedmiotów obserwowanych z niewielkiej odległości, różnice te będą 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 korygować położenie kamery względem oka. Jej położenie jest na tyle ważne, że nie może ona przesłaniać 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 zmiana jej położenia, spowoduje relatywnie duże odchylenie od wartości obserwowanych obiektów. W związku z tym „plusy" widoczne na etapie budowy aparatury, w trakcie testów, przerodziły się w „minusy".

7. Instrukcja użytkowania aplikacji

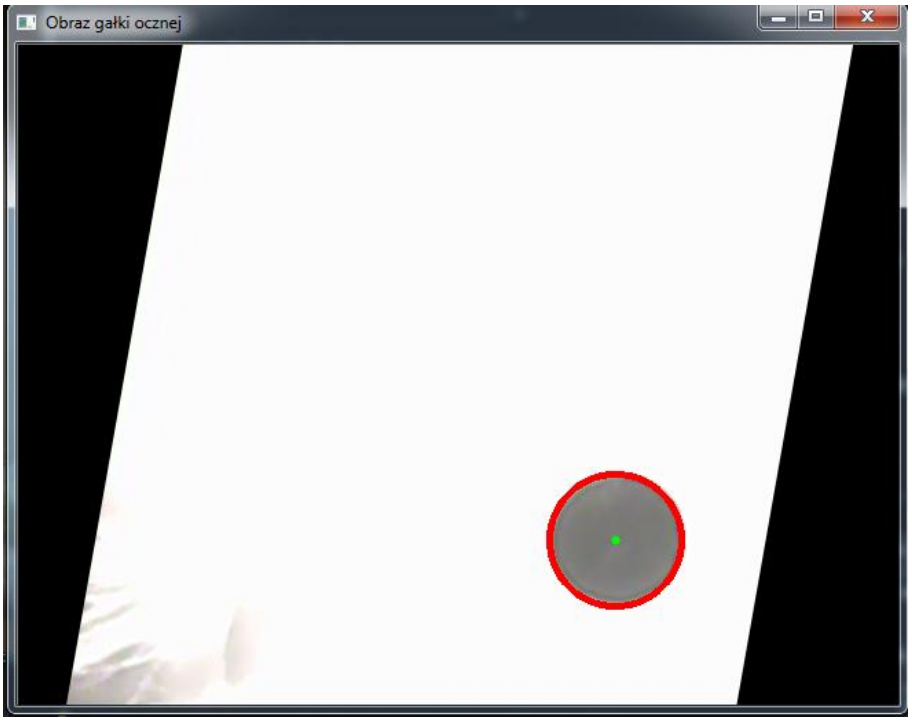
Niniejsza aplikacja została stworzona do celów naukowych, jako narzędzie do przeprowadzania wszelkiej maści badań. Ze względu na swój charakter, aplikacja nie została stworzona w wersji obiektowej. Użytkownik nie ma możliwości dokonywania jakichkolwiek ustawień. W zamyśle, aplikacja po uruchomieniu ma zarejestrować odpowiedni materiał filmowy, a następnie poddać go analizie. Ze względu na „kruchość" aparatury badawczej konieczne jest każdorazowe ręczne przeskalowanie obrazów. W tym celu należy kilkukrotnie uruchomić aplikację i metodą „prób i błędów" wyznaczyć wartości parametrów.

Po uruchomieniu aplikacji, od razu rozpoczyna się rejestrowanie obrazów z kamer. Po starcie aplikacji wyświetlają się dwa okna. Na jednym z nich ukazany jest obraz z kamery zamontowanej na czole badanej osoby. Natomiast w drugim oknie wyświetlany jest obraz z kamery skierowanej na gałkę oczną. Widok po uruchomieniu aplikacji został przedstwiony na Zdjęcie 8.



Zdjęcie 8. Widok po uruchomieniu aplikacji

Dodatkowo, na obrazie gałki ocznej, w czasie rzeczywistym zaznaczane są okręgi (Zdjęcie 9.). Dzięki temu jesteśmy wstanie, na bieżąco sprawdzać czy pozycja źrenicy, jest prawidłowa wykrywana.

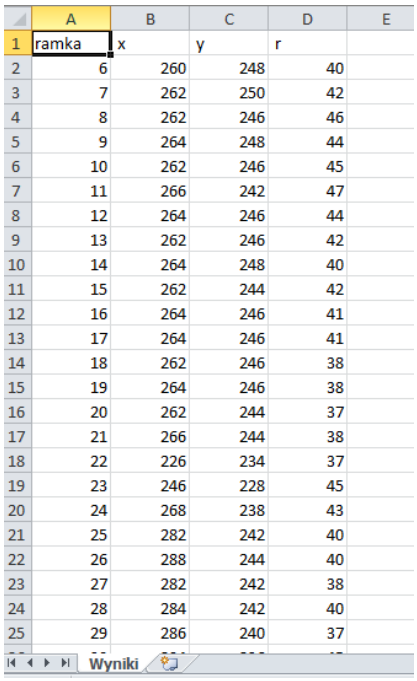


Zdjęcie 9. Wykrywanie źrenicy w czasie rzeczywistym

W przypadku błędów możemy dokonać natychmiastowej korekcji poprzez przesłonięcie części diod IR. Rejestrowane obrazy z kamer są automatycznie zapisywane na dysku, aby można było w łatwy sposób powrócić do nagranych wcześniej materiałów, w celu przeprowadzenia bardziej dogłębnych analiz. Nagrywanie materiałów filmowych przerywamy za pomocą przycisku „ESC". Wówczas aplikacja samoczynnie przechodzi do trybu analizy. Na ekranie wyświetla się okno z materiałem filmowym, zarejestrowanym z kamery umieszczonej na czole badanego. Na obrazie nanoszone są pierścienie w kilku rozmiarach. Największy pierścień wskazuje położenie oka w danej chwili. Wygląd pierścieni ma znaczenie. Pierścień, który jest największy a zarazem najjaśniejszy, wskazuje aktualną pozycję gałki ocznej. Kolejne pierścienie, będące odpowiednio pomniejszone oraz przyciemnione, wskazują poprzednie pozycje gałki ocznej. Im pierścień jest mniejszy i ciemniejszy tym wcześniej obserwowaliśmy dane miejsce. Po przeprowadzonej analizie plik jest zapisywany na dysku. W tym samym miejscu znajduje się plik .csv z danymi położenia gałki ocznej. Można z niego odczytać:

- numer klatki, w której został wykryty okrąg,
- współrzędne położenia okręgu,
- promień okręgu.

Przykładowy plik danych wynikowych został przedstawiony na Zdjęcie 10.



	A	B	C	D	E
1	ramka	x	y	r	
2	6	260	248	40	
3	7	262	250	42	
4	8	262	246	46	
5	9	264	248	44	
6	10	262	246	45	
7	11	266	242	47	
8	12	264	246	44	
9	13	262	246	42	
10	14	264	248	40	
11	15	262	244	42	
12	16	264	246	41	
13	17	264	246	41	
14	18	262	246	38	
15	19	264	246	38	
16	20	262	244	37	
17	21	266	244	38	
18	22	226	234	37	
19	23	246	228	45	
20	24	268	238	43	
21	25	282	242	40	
22	26	288	244	40	
23	27	282	242	38	
24	28	284	242	40	
25	29	286	240	37	

Zdjęcie 10. Przykładowy plik danych wynikowych

Za pomocą tych danych możemy przeanalizować, jak zmienia się szerokość źrenicy w danej chwili. Dodatkowo możemy w łatwy sposób zinterpretować poprawność danych i dokonać eliminacji błędnych pomiarów. Funkcja ta jest przydatna w przypadku wykrywania okręgów, które nie odzwierciedlają położenia gałki ocznej.

8. Bibliografia

[1] https://thefreecoder.wordpress.com/2012/09/11/opencv-c-video-capture/ (dostęp 01.01.2018)

[2] https://putuyuwono.wordpress.com/2015/05/12/single-thread-multi-camera-capture-using-opencv/ (dostęp 01.01.2018)

[3] http://rpetryniak.blogspot.com/2011/03/dostep-do-skadowych-piksela-w-opencv-22.html (dostęp 01.01.2018)

[4] https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html (dostęp 01.01.2018)

[5] https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/ (dostęp 01.01.2018)

[6] http://www.codepool.biz/multiple-camera-opencv-python-windows.html (dostęp 01.01.2018)

[7] https://msdn.microsoft.com/en-us/library/windows/desktop/dd377566(v=vs.85).aspx (dostęp 01.01.2018)

[8] https://putuyuwono.wordpress.com/2015/05/29/multi-thread-multi-camera-capture-using-opencv/ (dostęp 01.01.2018)

[9] http://aishack.in/tutorials/calibrating-undistorting-opencv-oh-yeah/ (dostęp 01.01.2018)

[10] http://vgl-ait.org/cvwiki/doku.php?id=opencv:tutorial:camera_calibration (dostęp 01.01.2018)

[11] http://ratixu.blogspot.com/2009/03/opencv-cz-3-otwieranie-i-zapis-obrazu.html (dostęp 01.01.2018)

[12] https://forbot.pl/blog/opencv-2-wykrywanie-obiektow-id4888 (dostęp 01.01.2018)

[13] http://opencv-srf.blogspot.com/2010/09/object-detection-using-color-seperation.html (dostęp 01.01.2018)

[14] https://www.biomed.org.pl/eye-tracking-w-neurorehabilitacji-pl.html#breadcrumb (dostęp 01.01.2018)

[15] https://www.youtube.com/watch?v=h2Taf16gQDI (dostęp 01.01.2018)

[16] http://www.eyetracker.pl/oferta-view/obszary-badan/ (dostęp 01.01.2018)

[17] https://www.biomed.org.pl/zastosowanie-eyetrackingu-w-terapii.html#breadcrumb (dostęp 01.01.2018)

9. Spis zdjęć