**Akademia Górniczo-Hutnicza**
**im. Stanisława Staszica w Krakowie**

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Automatyki

PRACA MAGISTERSKA

Mikołaj Sklepek

# Oprogramowanie do sterowania komputerem przy użyciu wskaźnika laserowego

Promotor:
**dr inż. Paweł Rotter**

Kraków, październik 2010

# OŚWIADCZENIE AUTORA PRACY

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

Mikołaj Sklepek

**AGH – University of Science and Technology in Kraków**

Faculty of Electrical Engineering, Automatics,

Computer Science and Electronics

MASTER OF SCIENCE THESIS

Mikołaj Sklepek

# Software for computer control with a laser pointer

Supervisor:
**dr inż. Paweł Rotter**

Kraków, October 2010

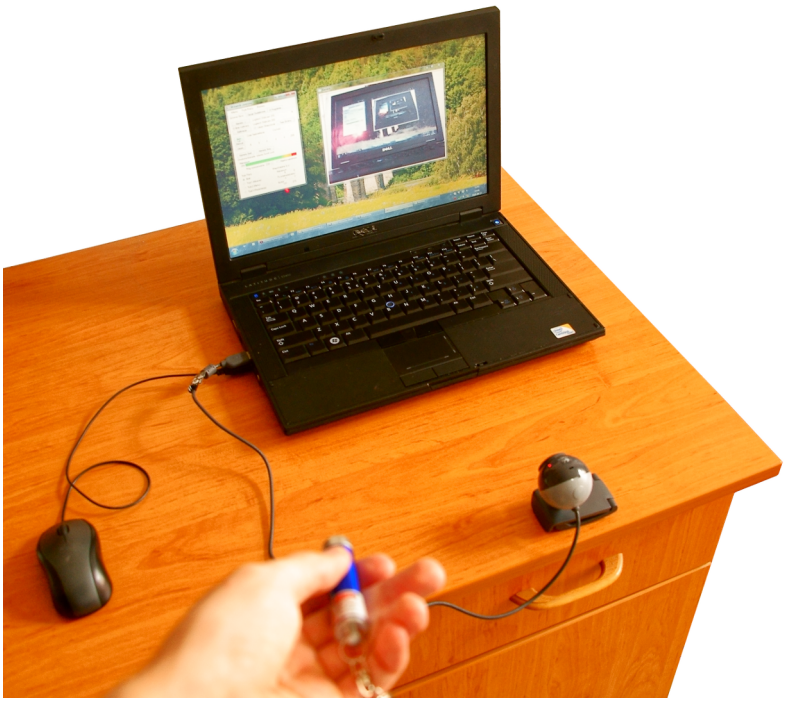
*Chciałbym podziękować promotorowi*
*Panu dr inż. Pawłowi Rotterowi*
*za poświęcony czas i pomoc*
*przy realizacji tej pracy.*

# Spis treści

# 1. Wstęp

Zadaniem postawionym w temacie pracy jest wykonanie oprogramowania pozwalającego na sterowanie komputerem za pomocą zwykłego wskaźnika laserowego. Napisany program – *Wskaźnik Laserowy* – ma na celu przede wszystkim ułatwić pokaz slajdów (prezentację). Zasada działania programu opiera się na zastąpieniu komputerowej myszki i klawiatury przez programowe wywoływanie odpowiednich zdarzeń tych urządzeń (przyciśnięcie klawisza, ustawianie kursora itp.), na podstawie ruchu plamki lasera na ekranie.

Do uruchomienia programu potrzebna jest kamera. Wystarczająca jest kamera internetowa, o standardowej rozdzielczości (640x480 pikseli), podłączana do komputera przez złącze USB. Kamera jest skierowana na ekran komputera (monitor – Rys.1. lub obraz z projektora) i obserwuje położenie plamki lasera. Oprogramowanie przelicza położenie plamki lasera na obrazie z kamery na położenie kursora na ekranie, tak że kursor podąża za wskaźnikiem laserowym. Przez odpowiedni ruch wskaźnika laserowego, możliwe jest zastąpienie myszki lub sterowanie pokazem slajdów, w zależności od wybranego trybu pracy programu.



**Rys.1.** Komputer z uruchomionym programem *Wskaźnik Laserowy*

Rozdział 2. opisuje elementy teorii przetwarzania obrazów wykorzystane w programie tj. przekształcenie perspektywiczne używane do kalibracji kamery. Pokazane zostały równania przekształcenia i sposób identyfikacji współczynników w oparciu o 4 punkty źródłowe i 4 punkty docelowe.

W rozdziale 3. została przedstawiona kalibracja kamery. Szczegółowo, ale bez wynikania w zagadnienia dotyczące implementacji, omówione zostały poszczególne etapy kalibracji i uzasadnione zastosowanie konkretnych rozwiązań. Ponadto rozdział zawiera rysunki i zrzuty ekranu z uruchomionej aplikacji obrazujące funkcjonowanie kalibracji w programie.

W rozdziale 4. został opisany ogólny schemat działania programu *Wskaźnik Laserowy*, czyli główna pętla programu. Podobnie jak we wcześniejszym rozdziale zostały wyróżnione etapy, z których składa się główna pętla programu, a ich opis został uzupełniony ilustracjami.

Rozdział 5. przedstawia zagadnienia dotyczące implementacji programu, czyli zastosowane środowisko programistyczne oraz biblioteki i funkcje używane w programie wraz z przykładowymi fragmentami kodu. W dalszej części rozdziału została przedstawiona struktura plików źródłowych projektu.

W rozdziale 6. zamieszczono specyfikację sprzętu używanego do testowania działania oprogramowania. Daje to informację o możliwościach uruchomienia programu na innym sprzęcie.

Rozdział 7. zawiera informacje nie przedstawione wcześniej. Opisuje działanie programu z punktu widzenia użytkownika, czyli wymagania systemowe, uruchomienie i użytkowanie programu. Zamieszczone zostały również zrzuty ekranu graficznego interfejsu i opisane zawarte w nim opcje.

# 2. Zniekształcenie perspektywiczne i metody jego korekcji

Przy pobieraniu obrazu za pomocą kamery częstym problemem są zniekształcenia geometryczne obiektu na obrazie [1]. Ich przyczyną mogą być krzywizna obiektu albo ustawienie kamery pod kątem (nie prostopadle) do obserwowanej powierzchni.

Powierzchnia ekranu z reguły jest płaska, dlatego w przypadku programu *Wskaźnik Laserowy*, występuje tylko druga przyczyna. Nieidealne ustawienie kamery powoduje zniekształcenia perspektywiczne obiektu na obrazie z kamery. Część powierzchni znajdująca się bliżej kamery jest bardziej rozciągnięta niż pozostałe [1].

Innym powodem (występującym tylko przy wyświetlaniu obrazu z komputera za pomocą projektora) może być ustawienie projektora względem ekranu.

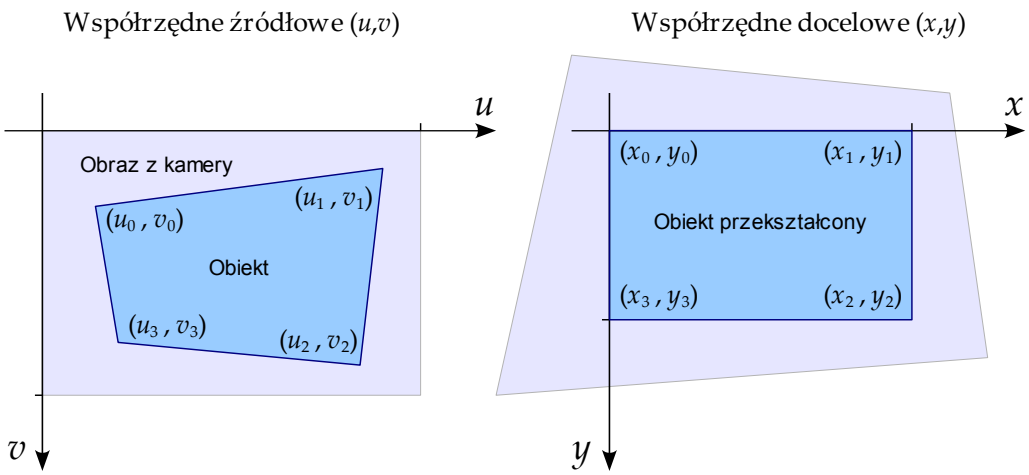
Do korekcji zniekształceń można skorzystać z jednego z przekształceń obrazu (i*mage warping*). Najprościej w oparciu o 4 punkty (narożniki obiektu na obrazie) – czteropunktowe przekształcenie obrazu (f*our corner image warping*).

## 2.1. Układ współrzędnych obrazu

Poniżej (Rys.2.) zostały przedstawione oznaczenia dla układów współrzędnych i charakterystycznych punktów w tych układach, użyte przy dalszym opisie przekształceń geometrycznych.

Przy zapisie i przetwarzaniu obrazów zwykle przyjmuje się, że lewy górny róg obrazu ma współrzędne (0,0) oraz, że współrzędne pikseli przyjmują wartości nieujemne, dlatego oś $y$ jest skierowana w dół [2]. Takie podejście nie ma praktycznego uzasadnienia, a pochodzi z telewizji (kineskopu) gdzie obraz jest wyświetlany liniami od góry.

W celu rozróżnienia współrzędnych przed i po przekształceniu, oznaczono współrzędne źródłowe jako $(u,v)$, natomiast współrzędne docelowe jako $(x,y)$.




**Rys.2.** Układ współrzędnych obrazu przed i po przekształceniu

## 2.2. Korekcja zniekształceń – metody

Do korekcji zniekształceń opisanych na początku tego rozdziału, można skorzystać z jednej z dwóch metod [2], [3], [5]:

- przekształcenie dwuliniowe (*bilinear transformation*)
- przekształcenie perspektywiczne (*perspective transformation* lub inaczej *projective mapping*)

### 2.2.1. Przekształcenie dwuliniowe

Przekształcenie dwuliniowe opisują następujące wzory [2]:

$$x = a_0 u + a_1 v + a_2 uv + a_3 \tag{1}$$

$$y = b_0 u + b_1 v + b_2 uv + b_3 \tag{2}$$

### 2.2.2. Przekształcenie perspektywiczne

Przekształcenie perspektywiczne najprościej zapisać we współrzędnych jednorodnych (*homogeneous coordinates*) [3], [4]:

$$p' = Hp \tag{3}$$

gdzie:

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4}$$

$$(5) \qquad p' = \begin{bmatrix} \hat{x} \\ \hat{y} \\ w \end{bmatrix} \qquad x = \frac{\hat{x}}{w} \qquad y = \frac{\hat{y}}{w}$$

$$(6) \qquad H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Bez straty ogólności można przyjąć, że $h_{33}=1$ z wyjątkiem przypadku gdy przekształcany jest punkt (0,0) na punkt w nieskończoności [3]. Przy korekcji zniekształceń obiektu na obrazie taki przypadek nie zachodzi.

Na podstawie równania (3) można podać wzory w postaci niemacierzowej na współrzędne $x$ (8) i $y$ (9):

$$(7) \qquad w = h_{31}u + h_{32}v + 1$$

$$(8) \qquad x = \frac{\hat{x}}{w} = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + 1}$$

$$(9) \qquad y = \frac{\hat{y}}{w} = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + 1}$$

### 2.2.3. Porównanie metod

Wymienione metody różnią się złożonością równań przekształcających współrzędne i ewentualnie dodatkowymi zniekształceniami wynikającymi z uproszczenia obliczeń. Im obliczenia bardziej złożone, tym przekształcenie jest dokładniejsze, czyli wprowadza mniej niepożądanych zniekształceń.

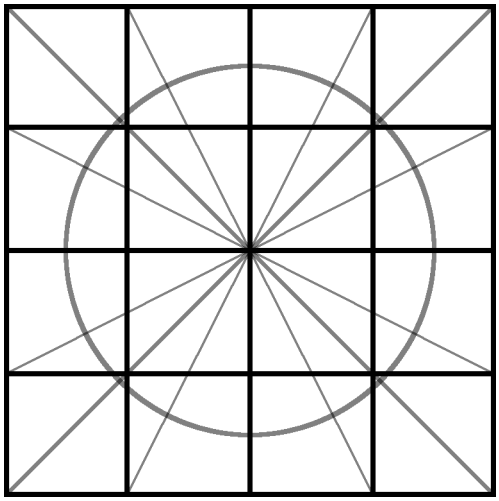
Główna różnica pomiędzy przekształceniami dwuliniowym i perspektywicznym to, że przekształcenie perspektywiczne zachowuje wszystkie linie proste z obrazu oryginalnego (Rys.5.), a przekształcenie dwuliniowe tylko linie pionowe i poziome (równoległe do osi współrzędnych).

Linie ukośnie na obrazie oryginalnym po przekształceniu dwuliniowym zwykle mają postać krzywych (paraboli) [3] (Rys.4.). Stopień zakrzywienia zależy od przekształcenia.

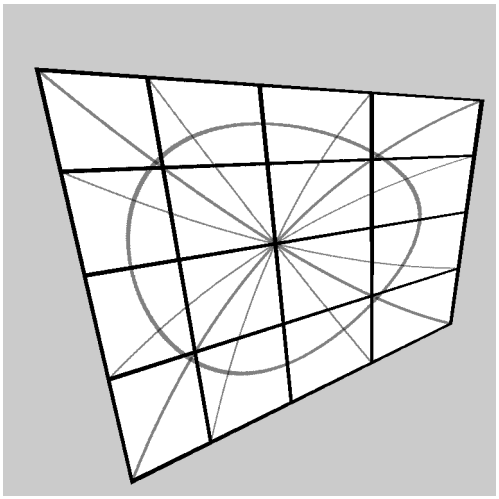
Na Rys.4. i Rys.5. zostały przedstawione przykładowe grafiki będące wynikiem danego przekształcenia. Dla porównania na Rys.3. został zamieszczony obraz przed przekształceniami.

Pliki graficzne po przekształceniu zostały utworzone w programie *Matlab* za pomocą skryptów zamieszczonych w załączniku (s.70). Przekształcenie w skryptach zostało zaimplementowane, tak jak w programie *Wskaźnik Laserowy*, tj. ze współrzędnych źródłowych na współrzędne docelowe. Przy takim postępowaniu zwykle nie są pokryte wszystkie piksele na obrazie docelowym, stąd lekkie prześwitywanie tła w lewym górnym rogu na Rys.5.

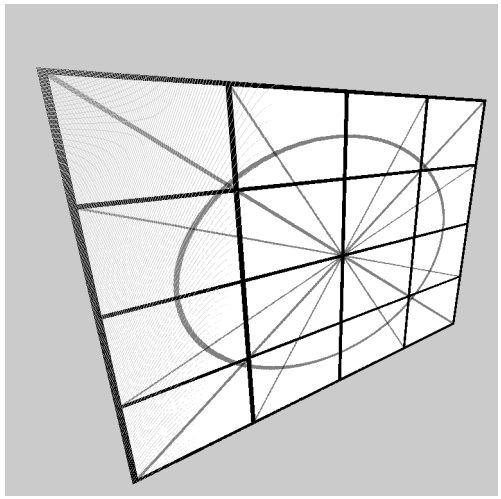
Przy przetwarzaniu całych obrazów zwykle stosuje się przekształcenie odwrotne, czyli do każdego piksela na obrazie docelowym przypisuje się kolor odpowiadającego mu piksela na obrazie źródłowym [2]. Zapobiega to powstawaniu „dziur" na obrazie wynikowym. Dodatkowo stosuje się jedną z metod interpolacji, dzięki czemu krawędzie na obrazie są „gładsze" [1].



**Rys.3.** Obraz oryginalny (bez przekształceń)



**Rys.4.** Przekształcenie dwuliniowe



**Rys.5.** Przekształcenie perspektywiczne

Do korekcji perspektywy w programie *Wskaźnik Laserowy* została wybrana metoda dokładniejsza tj. przekształcenie perspektywiczne (*perspective transformation*), ale nieco bardziej złożona obliczeniowo. Jednak obciążenie komputera

jest minimalne, ponieważ podczas działania programu w jednej iteracji (20 razy na sekundę), przekształcenie geometryczne jest wykonywane tylko dla jednej pary współrzędnych – dla jednego piksela, w którym zostało znalezione maksimum (4.4. s.26).

## 2.3. Wyznaczenie współczynników przekształcenia

Do obliczeń zostało wybrane przekształcenie perspektywiczne, dlatego dokładniej został przedstawiony sposób identyfikacji współczynników tego przekształcenia.

Do wyznaczenia współczynników $h_{ij}$ (macierz (6)) opisujących przekształcenie, potrzebne są współrzędne czterech punktów źródłowych $(u_k, v_k)$ i czterech odpowiadających im punktów docelowych $(x_k, y_k)$ [3]. Poszczególne pary punktów zostały oznaczone numerami $k=0,1,2,3$ (Rys.2., s.11).

W programie *Wskaźnik Laserowy* współrzędne punktów źródłowych są wyszukiwane podczas kalibracji (3.4. s.18), a współrzędne punktów docelowych wynikają z ustawionej rozdzielczości ekranu.

Przekształcenie perspektywiczne opisują równania (8) i (9). Przekształcając je do postaci odpowiednio (10) i (11) oraz uwzględniając współrzędne 4 punktów źródłowych $(u_k, v_k)$ i 4 punktów docelowych $(x_k, y_k)$, można na tej podstawie ułożyć układ ośmiu równań i zapisać w postaci macierzowej (12).

(10) $$h_{11}u_k + h_{12}v_k + h_{13} - h_{31}u_k - h_{32}v_k = x_k$$

(11) $$h_{21}u_k + h_{22}v_k + h_{23} - h_{31}u_k - h_{32}v_k = y_k$$

(12) $$\begin{bmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0y_0 & -v_0y_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2y_2 & -v_2y_2 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3y_3 & -v_3y_3 \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Wartości $u_k$, $v_k$, $x_k$, $y_k$ są stałe dla danego przekształcenia, a szukane są wartości współczynników $h$, zatem układ równań (12) można potraktować jako niejednorodny układ równań liniowych postaci (13).

Wartość jednego z 9 współczynników przekształcenia perspektywicznego została ustalona ($h_{33}$=1), dlatego do wyznaczenia zostało 8 współczynników.

(13) $A \cdot x = b \qquad A \in 8x8, \qquad b, x \in 8x1, \qquad b \neq 0$

Poszukiwany jest wektor $x$. Do programowego rozwiązania takiego układu można skorzystać z funkcji zaimplementowanej w bibliotece umożliwiającej wykonywanie operacji na macierzach np: funkcja *cvSolve* (5.4.7. s.45) z biblioteki *OpenCV*.

Dla przekształcenia dwuliniowego sposób identyfikacji 8 współczynników ($a_0 \ldots a_3$ oraz $b_0 \ldots b_3$) z równań (1) i (2) (s. 11) jest bardzo podobny. Również opiera się na podstawieniu do odpowiednich równań (14) i (15) współrzędnych 4 punktów źródłowych i 4 punktów docelowych [2], [6].

Obliczenie współczynników $a_0 \ldots a_3$

(14) $$\begin{bmatrix} u_0 & v_0 & u_0 v_0 & 1 \\ u_1 & v_1 & u_1 v_1 & 1 \\ u_2 & v_2 & u_2 v_2 & 1 \\ u_3 & v_3 & u_3 v_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Obliczenie współczynników $b_0 \ldots b_3$

(15) $$\begin{bmatrix} u_0 & v_0 & u_0 v_0 & 1 \\ u_1 & v_1 & u_1 v_1 & 1 \\ u_2 & v_2 & u_2 v_2 & 1 \\ u_3 & v_3 & u_3 v_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Warto zauważyć, że równania (14) i (15) mają mniejszy wymiar w porównaniu do równania (12) – 4x4 zamiast 8x8.

# 3. Kalibracja kamery

Do poprawnej pracy programu, zaraz po uruchomieniu, konieczne jest wykonanie kalibracji kamery. Kalibracja określa położenie kamery względem ekranu, dlatego trzeba ją również wykonać po zmianie ustawienia kamery lub obrazu na ekranie.

Przed kalibracją trzeba się upewnić, że kamera jest poprawnie ustawiona, i że obraz z komputera jest wyświetlany w całości (7.3. s.61). W przypadku gdy jest uruchomiona główna pętla programu (4. s.23), przed kalibracją zostaje zatrzymana.

Podczas kalibracji wykonywane są następujące etapy (szczegółowo opisane w dalszej części tego rozdziału):

1. Wyświetlenie okna kalibracji
2. Ustawienie ekspozycji kamery (opcjonalne)
3. Pobranie dwóch obrazów do kalibracji
4. Wyszukanie narożników ekranu
5. Zaznaczenie wykrytych narożników na obrazie z kamery
6. Wyznaczanie maski maksimum
7. Obliczenie współczynników przekształcenia
8. Zakończenie kalibracji

## 3.1. Wyświetlenie okna kalibracji

Na ekranie wyświetlane jest okno kalibracji. Początkowo jest to tylko białe tło z jasnoszarym napisem „Kalibracja”. Okno jest wyświetlane do czasu zakończenia kalibracji, która może potrwać od kilku do kilkudziesięciu sekund. Okno kalibracji jest wyświetlane ponad innymi oknami, a kursor jest ukryty, tak żeby inne uruchomione aplikacje nie wpływały na przebieg kalibracji. Podczas kalibracji nie może być widoczne żadne inne okno ani element systemu operacyjnego, ponieważ mogłoby to wprowadzić błędy przy wykrywaniu narożników ekranu.

## 3.2. Ustawienie ekspozycji kamery

Ten etap jest opcjonalny, zależny od ustawień w głównym oknie programu (7.4. s.62). Może znacząco wydłużyć czas kalibracji. Bez niego kalibracja trwa kilka sekund. Ustawienie ekspozycji kamery z reguły jest konieczne (jeśli nie zostało wykonane wcześniej), ponieważ obraz z kamery zwykle jest za jasny do celów wykrywania wskaźnika laserowego.

W trybie automatycznym kamera dobiera ekspozycję tak, aby obraz był „przyjemny dla oka" tzn. dość jasny z wyraźnymi kolorami i wysokim kontrastem. Ze względu na ograniczone możliwości kamery w rejestrowaniu obszarów jasnych i ciemnych część obrazu może być prześwietlona (całkowicie biały kolor) lub niedoświetlona (całkowicie czarny). O ile niedoświetlenie nie będzie miało wpływu na widoczność plamki lasera na obrazie, to w obszarach prześwietlonych wykrycie lasera będzie niemożliwe. Dlatego konieczne jest niedoświetlenie obrazu, tak żeby maksimum na obrazie (gdy nie ma widocznego lasera) było znacznie mniejsze od pełnej jasności.

W tym celu, na potrzeby programu *Wskaźnik Laserowy*, została zaimplementowana funkcja ustawiania ekspozycji kamery. Tryb ustawiania ekspozycji w kamerze zostaje przełączony na ręczny i w przypadku gdy obraz jest zbyt jasny ekspozycja zostaje zmniejszona. Do ustawienia jasności obrazu funkcja korzysta z możliwości biblioteki *DirectShow* (5.5 s.50). Skracany jest czas naświetlania i zmniejszana czułość kamery, do chwili osiągnięcia zadanej jasności obrazu, lub dopóki nie osiągną swoich minimalnych wartości.

Ze względu na kolor lasera uwzględniany jest tylko kanał czerwony obrazu. Wynikiem działania funkcji jest ekspozycja ustawiona w taki sposób, że maksimum na kanale czerwonym (najjaśniejszy piksel) jest w okolicy wartości podanej jako argument (domyślnie 200 – około 80% pełnej jasności). Niewielkie wahania maksimum (rzędu ±10) są spowodowane szumami na obrazie z kamery.

Funkcja może nie zadziałać poprawnie w przypadku obecności silnego źródła światła widocznego na obrazie z kamery (np: okno, lampa), ze względu na ograniczone możliwości zmniejszania ekspozycji.

## 3.3. Pobranie dwóch obrazów do kalibracji

Następnie pobierane są z wybranej kamery 2 obrazy konieczne do wyszukania narożników ekranu. Pobieranie obrazu jest realizowane za pomocą funkcji z biblioteki *OpenCV* (5.4.1 s.38). Obraz jest zapisywany w strukturze *IplImage* (5.4.4 s.40) przechowującej dane obrazu (wartości współczynników koloru – RGB) jak również inne parametry takie jak rozdzielczość, liczba kanałów itp.

Pierwszy obraz to białe tło bez zaznaczonych narożników (Rys.6.).

Następnie w oknie kalibracji zostają zaznaczone narożniki ekranu za pomocą czarnych kwadratów (domyślnie wielkości 10x10 pikseli).

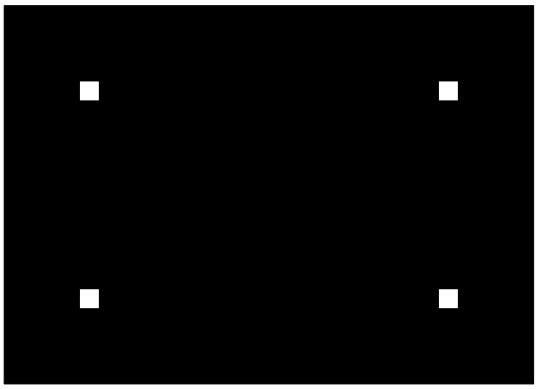
Zaraz po tym pobierany jest drugi obraz z kamery – białe tło z zaznaczonymi narożnikami (Rys.7.).

## 3.4. Wyszukanie narożników ekranu

Po pobraniu obu obrazów, zostaje odjęty obraz 2. od 1. i na tej podstawie są wyszukane narożniki ekranu. Największa różnica występuje tam gdzie zostały wyświetlone czarne kwadraty (Rys.8.).


**Rys.6.** Kalibracja – obraz 1.


**Rys.7.** Kalibracja – obraz 2.

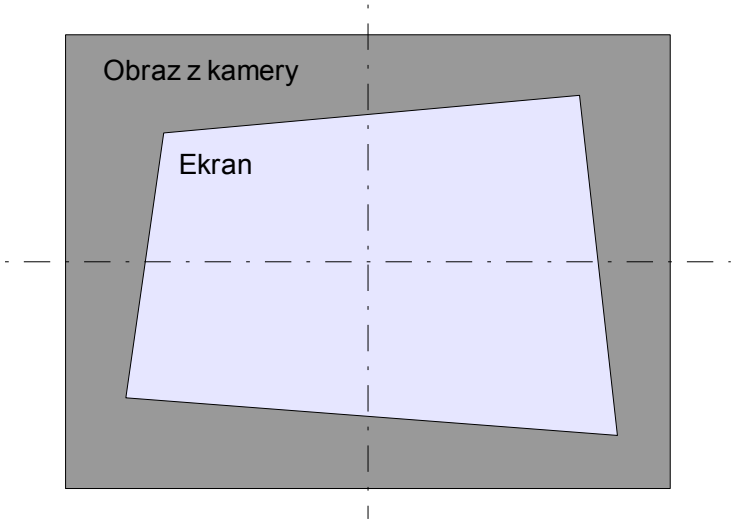


**Rys.8.** Kalibracja – różnica obrazów 1 i 2.

Nie jest to metoda bardzo dokładna, ale wystarczająca do celów sterowania komputerem za pomocą lasera. Dokładność można zwiększać pomniejszając rozmiar znaczników. Jednak przy zbyt małych znacznikach mogą wystąpić problemy z ich zarejestrowaniem na obrazie z powodu za małej rozdzielczości lub jakości obiektywu kamery. Inną możliwością, którą można by zastosować, jest wyświetlenie dość dużych znaczników i poszukiwanie środka lub krawędzi znacznika.

Dla szybkiego wykonywania kalibracji zaznaczane są od razu wszystkie 4 narożniki. Poszukiwanie narożników jest wykonywane osobno dla każdej ćwiartki obrazu. Znalezione 4 pary współrzędnych są zachowywane do dalszych obliczeń.

Każdy narożnik ekranu (obrazu z komputera) musi znajdować się w innej ćwiartce (Rys.9.). Warunek ten nie jest trudny do spełnienia i praktycznie zawsze można kamerę odpowiednio ustawić.

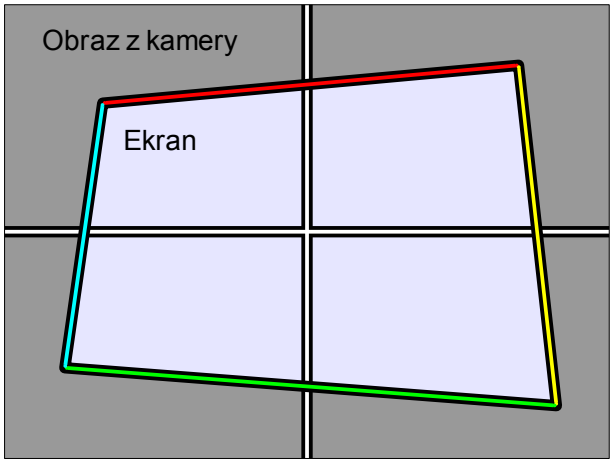



**Rys.9.** Prawidłowe ustawienie kamery względem ekranu – każdy narożnik ekranu umieszczony w innej ćwiartce obrazu z kamery.

## 3.5. Zaznaczenie wykrytych narożników na obrazie z kamery

Wykryte narożniki są zaznaczane na obrazie z kamery w postaci linii, które powinny pokrywać się z krawędziami ekranu – Rys.10. Dodatkowo rysowane są osie obrazu (linia pozioma w połowie wysokości i pionowa w połowie szerokości). Obraz jest widoczny dopiero po zakończeniu kalibracji i pozwala użytkownikowi ocenić poprawność wykonania kalibracji. Na obrazie z kamery powinny zostać prawidłowo zaznaczone kolorowymi liniami krawędzie ekranu w następującej kolejności:

- czerwony – górna krawędź
- żółty – prawa krawędź
- zielony – dolna krawędź
- błękitny – lewa krawędź




**Rys.10.** Poprawnie wykryte narożniki ekranu podczas kalibracji – kolorowe krawędzie pokrywają się z krawędziami ekranu

## 3.6. Wyznaczanie maski maksimum

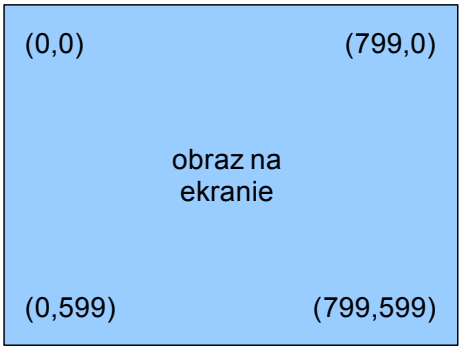
Maska maksimum jest obrazem binarnym (konkretnie zaimplementowana jako obraz w odcieniach szarości, ale użyte są tylko 2 kolory: czarny i biały). Służy do wyznaczenia obszaru poszukiwania maksimum.

Kolor biały (różny od czarnego) oznacza, że dany piksel będzie uwzględniany, a kolor czarny, że piksel ma zostać pominięty. Biały obszar pokrywa ekran na obrazie i obejmuje jeszcze dodatkowy margines o kilkanaście pikseli większy niż ekran. Maska maksimum umożliwia ograniczenie negatywnego wpływu (błędnego rozpoznania jako laser) jasnych obiektów znajdujących się poza ekranem – głównie odblaskowe lub świecące obiekty.

## 3.7. Obliczenie współczynników przekształcenia

Głównym celem kalibracji jest obliczenie współczynników przekształcenia, które umożliwia przeliczenie położenia plamki lasera na obrazie z kamery na położenie kursora na ekranie komputera. Obliczania opierają się na wyznaczeniu współczynników przekształcenia perspektywicznego (2.3. s.14).

Do obliczeń konieczne są współrzędne czterech punktów źródłowych i czterech odpowiadających im punktów docelowych. Współrzędne punktów źródłowych wyznaczone zostały we wcześniejszym etapie kalibracji (3.4. s.18). Natomiast współrzędne punktów docelowych wynikają z ustawionej rozdzielczości ekranu np.: dla rozdzielczości 800x600 px współrzędne zostały przedstawione na Rys.11. Część współrzędnych jest zawsze równa zero niezależnie od rozdzielczości.

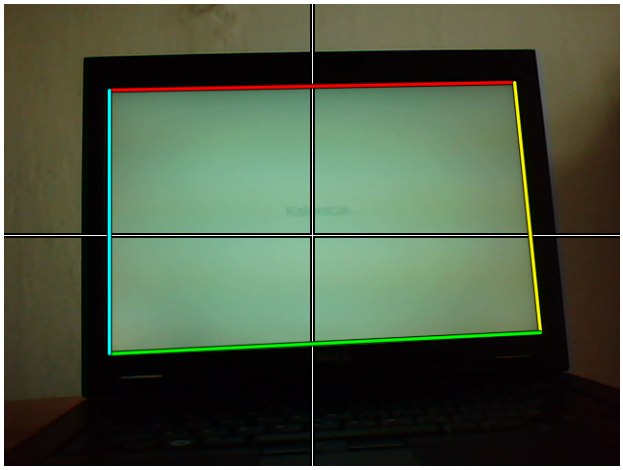



**Rys.11.** Współrzędne docelowe dla rozdzielczości ekranu 800x600

Współrzędne punktów są podstawiane do równania macierzowego (12) (s.14), które jest rozwiązywane za pomocą funkcji z biblioteki *OpenCV* (5.4.7 s.45). Wynikiem jest wektor zawierający 8 współczynników transformacji (dziewiąty zawsze równy 1). Otrzymane współczynniki są zapisywane i przechowywane w zmiennych z klasy *Class_Kamera* (opis pliku kamera.h, s. 56) do czasu wykonania kolejnej kalibracji.

## 3.8. Zakończenie kalibracji

Po zakończeniu kalibracji zamykane jest okno kalibracji i wyświetlane okno z wynikiem (obraz z zaznaczonymi krawędziami ekranu – Rys.12.) oraz okno komunikatu, w którym użytkownik ocenia poprawność kalibracji. Kolorowe linie powinny się pokrywać z krawędziami ekranu.



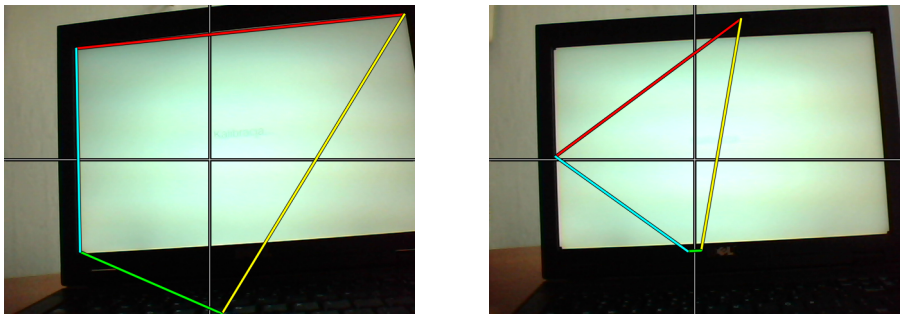
**Rys.12.** Poprawnie wykonana kalibracja – zrzut ekranu z działającej aplikacji

Gdy kalibracja zostanie zakończona powodzeniem, jest uruchamiana główna pętla programu. W przypadku niepowodzenia kalibrację trzeba wykonać ponownie.

Przyczynami nieudanej kalibracji może być:

- poruszenie kamery podczas kalibracji (Rys.13.)
- ruchomy obiekt w polu widzenia kamery (np. plamka lasera)
- obraz nie wyświetlany w całości na ekranie lub częściowo przysłonięty (niewidoczne znaczniki w narożnikach ekranu – Rys.13.)
- przykrycie okna kalibracji przez inne okno (np. okno komunikatu z innej aplikacji)
- błędne ustawienie kamery – poprawnie każdy narożnik ekranu musi być widoczny w innej ćwiartce obrazu z kamery.
- bardzo ciemny (lub bardzo jasny) obraz na ekranie lub obraz pobrany z kamery

Przykładowe zrzuty ekranu z nieudaną kalibracją zostały zamieszczone na Rys.13. W pierwszym przypadku kalibracja się nie udała ponieważ był niewidoczny prawy dolny narożnik ekranu (i tylko ten nie został poprawnie wykryty). Natomiast w drugim przypadku została poruszona kamera podczas kalibracji, w konsekwencji wszystkie narożniki są błędnie zaznaczone.



**Rys.13.** Przykłady błędnie wykonanej kalibracji. Z lewej niewidoczny jeden narożnik ekranu, z prawej poruszona kamera podczas kalibracji.

# 4. Schemat działania programu – główna pętla

Po poprawnie wykonanej kalibracji można uruchomić główną pętlę programu. Program może pracować w jednym z trzech trybów w zależności od ustawień w głównym oknie. Główna pętla programu jest wykonywana z częstotliwością 20Hz. W programie jest również opcja zmiany tej częstotliwości, jednak 20Hz jest optymalne. Okres wykonywania funkcji jest odmierzany za pomocą *Timera* (5.2 s.33). Obciążenie komputera podczas pracy programu nie jest duże (Tabela1.). Na średniej klasy sprzęcie użytym do testowania (6. s.59) przy domyślnej częstotliwości, program zajmuje kilka procent mocy procesora. Dlatego jego działanie nie powinno być odczuwalne dla użytkownika.

**Tabela1.** Przybliżone obciążenie komputera przez program *Wskaźnik Laserowy*.

| Częstotliwość głównej pętli [Hz] | Przybliżone obciążenie procesora [%] |
|---|---|
| 40 | 10 – 20 |
| **20** | **5 – 10** |
| 10 | do 5 |

Powyższe zestawienie zostało opracowane na podstawie wskazań *Menedżera zadań Windows*. Nie jest to metoda dokładna, ale daje ogólną informację na temat obciążenia systemu.

W każdej iteracji głównej pętli są wykonywane następujące operacje (szczegółowo opisane w dalszej części tego rozdziału):

1. Pobranie aktualnego obrazu
2. Wyszukanie maksimum (wartość i współrzędne)
3. Wyświetlenie obrazu z zaznaczonym maksimum
4. Przeliczenie położenia maksimum na współrzędne ekranu
5. Uśrednianie położenia kursora
6. Uruchomienie funkcji związanych z wybranym trybem pracy
7. Wyświetlanie wyników działania głównej pętli

## 4.1. Pobranie aktualnego obrazu

Pierwszym etapem jest pobranie aktualnego obrazu z wybranej kamery. Pobieranie obrazu jest wykonywane analogicznie jak przy kalibracji (3.3. s.17).

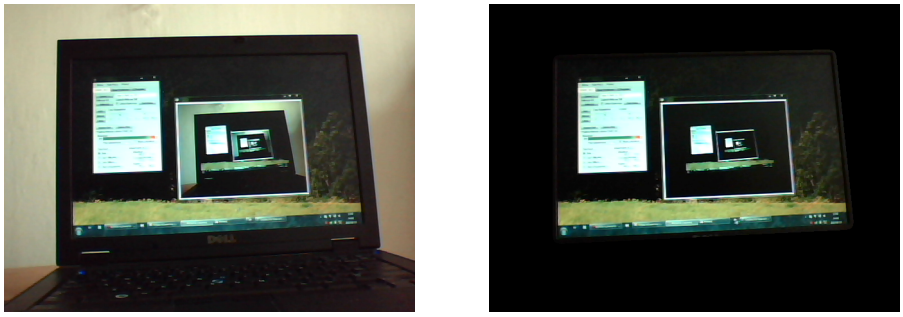
## 4.2. Wyszukanie maksimum

Następnie jest wyszukiwane maksimum na pobranym obrazie. Ponieważ we wskaźnikach laserowych zwykle jest stosowany laser w kolorze czerwonym, poszukiwanie maksimum jest realizowane tylko na kanale czerwonym obrazu. Wynikiem działania funkcji są współrzędne $(x,y)$ maksimum i jego wartość.

Poszukiwanie maksimum może być realizowane na 2 sposoby:

- na całym obrazie
- w wybranym obszarze

Wybór opcji jest możliwy w głównym oknie programu (7.2. s.60). W pierwszym przypadku poszukiwane jest po prostu maksimum globalne na kanale czerwonym. Natomiast w drugim obszar poszukiwania maksimum jest ograniczony przez maskę utworzoną na etapie kalibracji (3.6. s.20). Oba przypadki zostały przedstawione na Rys.14.

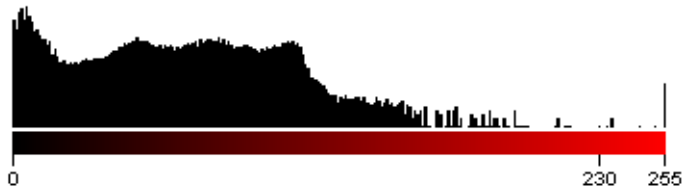


**Rys.14.** Obraz z kamery – maska wyłączona (z lewej) i włączona (z prawej). Przy włączonej masce uwzględniany jest tylko ekran z niewielkim marginesem.

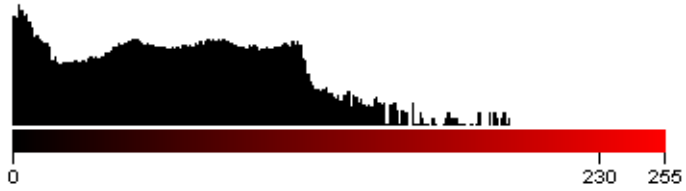
Po ustaleniu wartości maksimum na kanale czerwonym, jest określane czy wyszukane maksimum jest plamką lasera. W tym celu sprawdzany jest prosty warunek – jeśli wartość maksimum jest większa od pewnej wartości progowej (ustalonej w głównym oknie programu – 7.2. s.60), to można przyjąć, że maksimum odpowiada plamce lasera na obrazie z kamery. Można tak postąpić, ponieważ plamka lasera jest bardzo jasna i zwykle dużo jaśniejsza od obrazu na ekranie. Jedynie przy bardzo jasnym obrazie (ustawiony duży kontrast i jasność ekranu) mogą wystąpić problemy w zlokalizowaniu lasera.

Domyślnie próg został ustalony na 230, a jasność obrazu z kamery jest ustawiona podczas kalibracji (3.2. s.16) tak, żeby maksimum na kanale czerwonym bez lasera na obrazie wynosiło około 200 (Rys.16.). Przy takim ustawieniu w przypadku pojawienia się plamki lasera na ekranie wartość maksimum wzrasta najczęściej do granicznej wartości 255 (Rys.15.). Takie różnice pomiędzy wartościami maksimum w przypadkach gdy jest laser lub go nie ma oraz wartością progową, umożliwia wyeliminowanie wpływu szumów czy niewielkich zmian jasności obrazu z kamery.

Poniżej (s.25) zostały przedstawione histogramy (dla kanału czerwonego) obrazów w różnych przypadkach. Ze względu na fakt, że liczba pikseli odpowiadająca plamce lasera jest bardzo mała w porównaniu do liczby wszystkich pikseli na obrazie, dla zwiększenia czytelności histogramy zostały narysowane w skali logarytmicznej.




**Rys.15.** Histogram obrazu z kamery z widocznym laserem na ekranie

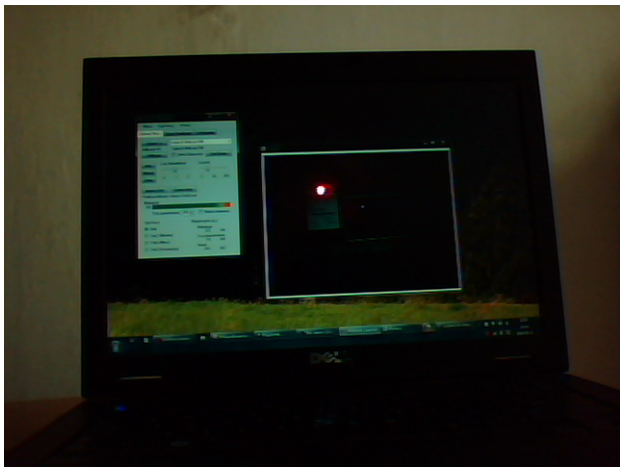



**Rys.16.** Histogram obrazu z kamery bez widocznego lasera

Gdyby nie było zaimplementowanej funkcji, która ustawia ekspozycję kamery, prawidłowe wyszukanie lasera byłoby z reguły niemożliwe. Obraz przy automatycznym ustawianiu jasności zwykle jest za jasny (Rys.17.).


**Rys.17.** Histogram obrazu z kamery przy automatycznym ustawianiu jasności (bez lasera)

Na Rys.18. został zamieszczony obraz z kamery przy prawidłowo ustawionej jasności obrazu za pomocą funkcji programu *Wskaźnik Laserowy*. Histogram tego obrazu został zamieszony na Rys.15.



**Rys.18.** Obraz z kamery z prawidłowo ustawioną jasnością (laser widoczny na środku)

## 4.3. Wyświetlenie obrazu z zaznaczonym maksimum

W celu oceny poprawności działania programu obraz z kamery wraz z zaznaczonym maksimum jest wyświetlany w osobnym oknie programu (okno podglądu obrazu - Rys.23. s.61). Położenie maksimum jest zaznaczane za pomocą dwóch linii – pionowej (współrzędna $x$) i poziomej (współrzędna $y$). Maksimum jest zaznaczane tylko wtedy gdy przekracza wartość progową (jest plamka lasera na ekranie). Pozwala to ocenić czy dobrana wartość progowa i jasność obrazu z kamery umożliwiają poprawne wyszukiwanie położenia plamki lasera.

Gdy jest wybrany jeden z trybów pracy programu w głównym oknie programu (7.2. s.60) wykonywane są dalsze instrukcje. Opcja „Brak" umożliwia przetestowanie wyszukiwania lasera bez wpływu na pracę komputera – kursor nie jest ustawiany i nie są wykonywane kliknięcia. Dodatkowo dalsze działanie głównej pętli programu może być zablokowane przez włączenie klawisza *Scroll Lock* na klawiaturze. Dzięki temu można łatwo odzyskać kontrolę nad komputerem w przypadku błędnego wykrywania maksimum jako plamki lasera. Taka sytuacja może wystąpić na skutek przypadkowego pojawienia się jasnego obiektu w polu widzenia kamery lub znaczącego zwiększenia jasności obrazu.

## 4.4. Przeliczenie położenia maksimum na współrzędne ekranu

Kiedy dalsze działanie funkcji nie jest zablokowane kolejnym etapem jest przeliczenie współrzędnych $(x,y)$ położenia plamki lasera na współrzędne ekranu komputera.

Do przeliczenia współrzędnych używane są równania (8) i (9) (s.12). Obliczone wartości są zaokrąglane do liczb całkowitych. Używane we wzorach współczynniki są wcześniej wyznaczone na etapie kalibracji. Wybrana metoda nie jest najszybsza, ale daje możliwie największą dokładność (2.2.3. s.12). W danej iteracji są przekształcane współrzędne tylko dla jednego punktu, więc wpływ na ogólną szybkość działania programu jest pomijalny. Mogłoby to mieć znaczenie np. w przypadku przekształcania całego obrazu.

## 4.5. Uśrednianie położenia kursora

W celu zwiększenia dokładności działania programu i zminimalizowania wpływu drgań wskaźnika laserowego trzymanego w ręce, została zaimplementowana funkcja uśredniająca położenie kursora.

Przed ustawieniem kursora obliczane jest średnie położenie plamki lasera na ekranie. Obliczana jest średnia arytmetyczna z 20 ostatnich położeń (iteracji

głównej pętli), osobno dla współrzędnej $x$ i osobno dla $y$. Żeby przyspieszyć przesuwanie kursora na większe odległości uśrednianie jest wykonywane tylko przy niewielkich zmianach położenia plamki lasera względem średniej wartości z wcześniejszych położeń (w promieniu mniejszym niż 100 pikseli, dobranym eksperymentalnie). W przypadku większego skoku, kursor jest natychmiast przesuwany w nowe położenie, a średnia jest liczona od nowa bez uwzględniania poprzednich wartości.

Algorytm działania funkcji:

1. Dopisz aktualne położenie lasera (po korekcji perspektywy) do tablicy (2 tablice – osobna dla współrzędnych $x$ i $y$)
2. Oblicz średnią $(x_s, y_s)$ z elementów z obu tablic – średnie położenie $x$ i $y$
3. Sprawdź warunek: $\sqrt{(x_s-x)^2+(y_s-y)^2}>R$
   - jeśli prawdziwy (przeskok plamki lasera na obrazie większy niż $R$), to krok 4.
   - w przeciwnym przypadku – krok 5 (pomiń krok 4).
4. Jako średnią $(x_s, y_s)$ przyjmij aktualne położenie $(x, y)$
5. Zwróć wartość średnią $(x_s, y_s)$

## 4.6. Uruchomienie funkcji związanych z wybranym trybem pracy

Na końcu pętli (w zależności od wybranego trybu pracy programu) jest ustawiany kursor i ewentualnie wywoływane są programowo zdarzenia klawiszy myszy lub klawiatury w zależności od położenia plamki lasera na ekranie. Kolejne zdarzenia są wywoływane w odstępach co najmniej 2 sekund (40 iteracji głównej pętli), tak żeby było możliwe zgaszenie lasera bez ponownego uruchamiania tej samej funkcji.

Program może działać w kilku trybach, które można wybrać z listy w oknie głównym programu (7.2. s.60). Pomiędzy niektórymi trybami jest możliwe przechodzenie za pomocą wskaźnika laserowego. Wszystkie opierają się opierają się na programowym wywołaniu zdarzeń klawiatury lub myszy (5.3 s.34).

### 1. Tryb Klikania

Najprostszy tryb do pracy w systemie. Nadaje się raczej do przetestowania działania programu na danym sprzęcie niż do konkretnych zastosowań.

Możliwości:

- przesuwanie kursora na ekranie
- kliknięcie lewym klawiszem myszy po krótkim przytrzymaniu lasera (ok. 2s) mniej więcej w jednym miejscu (np. nad ikoną)

Ze względu na drgania lasera nie ma sensu sprawdzanie czy przez kilkanaście iteracji położenie lasera na ekranie było takie samo (takie same współrzędne). Konieczne jest dopuszczenie zmian współrzędnych w pewnym promieniu.

Programowo sprawdzanie czy laser był przytrzymany zostało zaimplementowane następująco:

- obliczenie średniego położenia lasera z ostatnich 40 iteracji
- porównanie średniej z aktualnym położeniem
- gdy laser był na ekranie w poprzednich iteracjach i różnica pomiędzy jego średnim i aktualnym położeniem jest niewielka, wykonywane jest kliknięcie.

### 2. Tryb Menu

Zadaniem tego trybu jest zastąpienie myszki podczas pracy w systemie operacyjnym. Ze względu na ograniczoną precyzję prowadzenia lasera po ekranie (drgania ręki) oraz rozdzielczość kamery, dość trudne jest zapewnienie dokładności pracy porównywalnej z typową myszką.

Możliwości:

- przesuwanie kursora zgodnie z położeniem lasera
- po przytrzymaniu wskaźnika laserowego w jednym miejscu przez około 2 sekundy (np. nad ikoną lub przyciskiem) obok kursora wyświetlane jest menu (Rys.19. s.29) z następującymi opcjami:
  - kliknięcie lewym klawiszem myszy
  - podwójne kliknięcie lewym klawiszem myszy
  - kliknięcie prawym klawiszem myszy
  - uruchomienie trybu prezentacji + przełączenie na pełny ekran prezentacji otwartej w programie (*Power Point* lub *Impress*)
  - wyłączenie menu


**Rys.19.** Menu

Wykrywanie przytrzymania lasera przed wyświetleniem menu jest realizowane analogicznie jak w przypadku trybu klikania. Położenie okna menu jest ustalane w taki sposób, aby było wyświetlane w pobliżu wskaźnika laserowego (poniżej lub powyżej) i zawsze widoczne w całości na ekranie.

Wybór opcji następuje po wskazaniu wskaźnikiem laserowym odpowiedniego przycisku z menu. Kliknięcia są wykonywane przez programowe wywołanie odpowiednich zdarzeń myszy (5.3.2. s.34). Przy wykonywaniu kliknięć kursor jest ustawiany w miejscu, z którego zostało wywołane menu.

### 3. Tryb Prezentacji

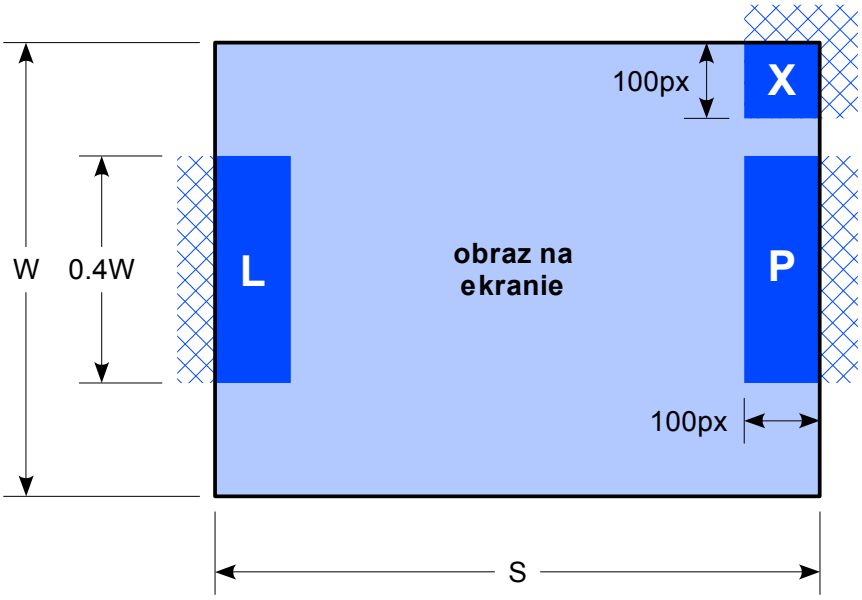
Celem tego trybu jest przewijanie slajdów podczas prezentacji w programach takich jak *Power Point* lub *OpenOffice Impress*.

Możliwości:

- przewijanie slajdów przez przytrzymanie wskaźnika laserowego przy krawędzi ekranu (mniej więcej w połowie wysokości ekranu – Rys.20.)
  - z prawej strony – następny slajd,
  - z lewej – poprzedni slajd,
- przytrzymanie wskaźnika laserowego w prawym górnym rogu ekranu – wyjście z prezentacji i powrót do poprzedniego trybu (trybu menu),
- wyłączona możliwość klikania – umożliwia to pokazywanie treści na slajdach za pomocą wskaźnika laserowego bez zakłócania pokazu slajdów.

Funkcja realizująca tryb prezentacji przechowuje informacje o obecności lasera na ekranie i jego współrzędnych z 40 poprzednich iteracji. Jeżeli w co najmniej 30 iteracjach laser znajdował się w jednym obszarze (jednym z 3 wyróżnionych na ekranie – Rys.20.), oznacza to że laser był przytrzymany w tym obszarze przez około 2 sekundy i można uruchomić funkcję przypisaną do tego obszaru.

Przechodzenie do następnego (poprzedniego) slajdu jest realizowane przez programowe wywołanie wciśnięcia klawisza strzałki w prawo (w lewo), natomiast wyjście z prezentacji przez wywołanie wciśnięcia klawisza *Escape* (*Esc*) (5.3.3 s.35).




**Rys.20.** Obszary działania wskaźnika laserowego na ekranie w trybie prezentacji
L – lewy (poprzedni slajd), P – prawy (następny slajd), X – koniec pokazu
W – wysokość ekranu [px], S – szerokość ekranu [px]

## 4.7. Wyświetlanie wyników działania głównej pętli

Podczas wykonywania głównej pętli w oknie programu wyświetlane są następujące wartości:

- położenie maksimum $(x,y)$
- wartość maksimum
- współrzędne po przeliczeniu na współrzędne ekranu $(x,y)$
- położenie kursora na ekranie $(x,y)$
- aktualny tryb pracy programu

# 5. Implementacja

## 5.1. Środowisko programistyczne

Program *Wskaźnik Laserowy* został zrealizowany w środowisku programistycznym *Microsoft Visual C++ 2005 Express Edition*.

Główne przyczyny wyboru tego środowiska:

- język programowania C++ charakteryzujący się dużą uniwersalnością, możliwościami i dobrą wydajnością tworzonego oprogramowania
- szczegółowa dokumentacja i przykłady użycia dostępnych funkcji na stronie producenta
- znacząca popularność środowiska, a co się z tym wiąże dostępna w sieci internet duża ilość przykładów kodu, rozwiązań problemów przy kompilacji itp.
- łatwość tworzenia graficznego interfejsu użytkownika (*Windows Forms*)
- środowisko udostępnione bezpłatnie zarówno do celów edukacyjnych jak i komercyjnych (dotyczy wersji *Express Edition*)

W przypadku tworzenia tego typu projektu ograniczenia wersji *Express Edition* praktycznie nie mają żadnego znaczenia. Są to przede wszystkim [18]:

- brak możliwości tworzenia jednego programu w kilku językach
- brak wsparcia dla urządzeń przenośnych
- brak rozszerzeń współpracy zespołowej

Została wybrana nieco starsza wersja – 2005, głównie ze względu na mniejsze wymagania sprzętowe (mniejsza ilość zajmowanego miejsca na dysku twardym i mniejsze rozmiary plików instalacyjnych). Na początku pisania pracy najnowszą wersją była wersja 2008, natomiast wersja 2010 była jeszcze w fazie *beta*.

Użyte standardowe możliwości środowiska *Visual C++*:

- tworzenie, debugowanie i kompilacja całego projektu (5.6 s.56)
- graficzny interfejs użytkownika (*Windows Forms*)
- podstawowe struktury i typy danych
- odmierzanie czasu – *timer* (5.2 s.33)
- funkcje do programowego wywoływania zdarzeń klawiatury i myszy (5.3 s.34)

Standardowe możliwości środowiska *Visual C++*, zostały dodatkowo rozszerzone przez oddzielne biblioteki funkcji: *OpenCV* oraz *DirectShow*.

**Biblioteka *OpenCV***

Biblioteka *OpenCV* zawiera implementację wielu algorytmów używanych przy przetwarzaniu obrazów, oraz struktur i funkcji ułatwiających operacje na danych obrazu (5.4 s.37).

Przy tworzeniu programu *Wskaźnik Laserowy* została wykorzystana biblioteka w wersji 2.0.0a.

Jej głównymi zadaniami w programie są:

- pobieranie i wyświetlanie obrazu z kamery (znacznie prostsze niż za pomocą biblioteki *DirectShow*) (5.4.1 s.38 oraz 5.4.2 s.39)
- przechowywanie obrazu w strukturze *IplImage*, umożliwiającej szybki dostęp do danych opisujących obraz (5.4.4 s.40)
- rozwiązywanie równań macierzowych (5.4.7 s.45) przy kalibracji kamery
- zaznaczanie na obrazie wyników działania programu (5.4.8 s.47)

**Biblioteka *DirectShow***

Głównym zastosowaniem biblioteki *DirectShow* są aplikacje multimedialne korzystające ze strumieni audio i wideo [42] (5.5 s.50).

Cel zastosowania biblioteki:

- sterowanie pracą kamery – ustawianie ekspozycji
- wyświetlanie nazw zainstalowanych urządzeń (kamer)

W kolejnych rozdziałach, przy opisie bibliotek zostały zamieszczone przykładowe fragmenty kodu. Część z nich została przedstawiona w „czystym" języku C++, tj. możliwa do skompilowania w kompilatorze takim jak na przykład *Dev-Cpp*. Ma to na celu zwiększenie przejrzystości zamieszczonego kodu. Podstawową różnicą jest sposób wyprowadzania danych. W *Visual C++* bardziej rozbudowany, ale oferujący większe możliwości – wypisywanie danych do pola tekstowego, etykietki itp. Do uruchomienia tych przykładów w środowisku *Visual C++* konieczna jest modyfikacja kodu np. dodanie funkcji konwertujących wartości liczbowe na ciąg znaków.

## 5.2. Cykliczne wykonywanie głównej pętli programu (*Timer*)

Funkcje dotyczące ustalania położenia kursora i wykrywania „kliknięć" wykonywane są cyklicznie. Do odmierzania czasu został wykorzystany *Timer* dostępny w *MS Visual C++* [19].

Ma on jednak kilka ograniczeń [20]:

- nie zapewnia dużej dokładności – nie większą niż 1/18 sekundy,
- nie gwarantuje idealnie równych ostępów czasu,
- zakres możliwych do ustawienia odstępów czasu wynosi od 1 do 64767 milisekund,
- w przypadku gdy czas wykonania funkcji będzie dłuższy niż zadeklarowany okres *Timera*, wywoływana funkcja nie zostanie przerwana, a odstępy czasu będą wydłużone.

Pomimo powyższych ograniczeń zastosowany *timer* nie powoduje znaczącego ograniczenia funkcjonalności programu. Nie są konieczne idealnie równe odstępy czasowe, a częstotliwość rzędu 20Hz jest wystarczająca.

Główny cel *timera* w programie to zapewnienie mniej więcej stałej częstotliwości wykonywania pętli programu i zmniejszenie zużycia zasobów systemowych – pętla nie jest wykonywana z maksymalną możliwą na danym sprzęcie częstotliwością. Dodatkową zaletą jest prosta implementacja. Do działania wystarczy zdefiniować funkcję *timer_Tick*, wykonywaną cyklicznie w zadanych odstępach czasowych i uruchomić *timer* funkcją *Start*. Do zatrzymania timera służy funkcja *Stop*.

Przykład – implementacja prostego licznika:

- funkcja *timer1_Tick* powiązana z timerem (*timer1*):

```
static int counter = 0;
counter++;                          // inkrementacja
if (counter >= 100) { counter=0; }  // zerowanie
label1->Text = counter.ToString();  // wyswietlenie wartosci
```

- uruchomienie timera:

```
timer1->Interval = 1000;    // ustawienie czasu (1000ms)
timer1->Start();
```

- zatrzymanie timera:

```
timer1->Stop();
```

## 5.3. Funkcje do programowego wywoływania zdarzeń myszy i klawiatury

### 5.3.1. Ustawianie kursora myszy na ekranie

Do ustawienia kursora w określonym punkcie na ekranie została zdefiniowana funkcja *SetCursorPos* [21]. Funkcja jest zadeklarowana w pliku nagłówkowym *windows.h* i przyjmuje dwa argumenty – współrzędne ekranu $x$ i $y$. W przypadku gdy któryś z argumentów przekracza zakres wynikający z rozdzielczości ekranu, kursor jest ustawiany przy brzegu ekranu.

Przykładowe wywołanie funkcji *SetCursorPos*:

```
SetCursorPos(40, 20);
```

Kursor jest ustawiany w punkcie $x$:=40, $y$:=20 tj. 40 pikseli od lewej krawędzi ekranu i 20 pikseli od góry.

### 5.3.2. Programowe wywołanie kliknięcia klawiszem myszy

Do programowego wywołania kliknięcia myszy można skorzystać z funkcji *mouse_event* [22]. Funkcja jest zadeklarowana w pliku nagłówkowym *windows.h*.

Przy wywoływaniu kliknięć używany jest tylko pierwszy argument funkcji *mouse_event*. Pozostałe argumenty nie są używane w programie *Wskaźnik Laserowy* (mają wartość 0), dlatego ich opis został pominięty.

Wartości argumentu używane w programie:

- *MOUSEEVENTF_LEFTDOWN*
  – przyciśnięcie (i przytrzymanie) lewego klawisza
- *MOUSEEVENTF_LEFTUP*
  – puszczenie lewego klawisza
- *MOUSEEVENTF_RIGHTDOWN*
  – przyciśnięcie (i przytrzymanie) prawego klawisza
- *MOUSEEVENTF_RIGHTUP*
  – puszczenie prawego klawisza

Pozostałe wartości, które można znaleźć w [22], odpowiadają wciśnięciu jednego z dodatkowych klawiszy lub obróceniu kółka myszy.

Rozróżnienie przyciśnięcia i puszczenia klawisza daje dodatkowe możliwości np: dłuższe przytrzymanie przyciśniętego klawisza, albo „przeciąganie”.

Przykładowy kod – kliknięcie lewym klawiszem myszy:

```
mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
```

Przykładowy kod – kliknięcie prawym klawiszem myszy:

```
mouse_event(MOUSEEVENTF_RIGHTDOWN, 0, 0, 0, 0);
mouse_event(MOUSEEVENTF_RIGHTUP, 0, 0, 0, 0);
```

### 5.3.3. Programowe wywołanie przyciśnięcia klawisza klawiatury

Do programowego wywołania zdarzenia klawiatury można skorzystać z funkcji *keybd_event* [23]. Funkcja jest zdefiniowana w pliku nagłówkowym *windows.h*. Przy uruchomieniu funkcji są rozróżniane zdarzenia przyciśnięcia i puszczenia klawisza. Umożliwia to np. wywołanie przyciśnięcia kilku klawiszy na raz lub przytrzymanie klawisza przez dłuższy czas.

Do prawidłowego wywołania przyciśnięcia pojedynczego klawisza trzeba uruchomić funkcję *keybd_event* dwa razy z różnymi parametrami (wciśnięcie i puszczenie), co przedstawia poniższy przykład. Uruchomienie tego kodu spowoduje włączenie (lub wyłączenie) numerycznej części klawiatury, co będzie widoczne przez zaświecenie (lub zgaszenie) diody na klawiaturze odpowiadającej klawiszowi *NumLock*.

Przykładowy kod (przyciśnięcie klawisza *NumLock*).

```
// Programowe "przyciśnięcie" klawisza
    keybd_event( VK_NUMLOCK,
                 MapVirtualKey(VK_NUMLOCK,0),
                 0,
                 0 );
// Programowe "puszczenie" klawisza
    keybd_event( VK_NUMLOCK,
                 MapVirtualKey(VK_NUMLOCK,0),
                 KEYEVENTF_KEYUP,
                 0 );
```

Funkcja *keybd_event* przyjmuje 4 argumenty:

- kod wirtualnego klawisza (*virtual-key code*) [24]
- sprzętowy kod (*hardware scan code*) – można go uzyskać np. za pomocą funkcji *MapVirtualKey* [25] na podstawie kodu wirtualnego klawisza, lub z tablicy kodów [26]
- flagi – ustawiona flaga *KEYEVENTF_KEYUP* oznacza puszczenie klawisza, nieustawiona oznacza przyciśnięcie klawisza.
- dodatkowa wartość, zwykle równa 0.

### 5.3.4. Kody klawiszy klawiatury (*Virtual-Key Codes*)

Każdy klawisz na klawiaturze ma przypisany swój własny, unikalny numer (kod). Do programowego wywołania zdarzeń konkretnych klawiszy konieczna jest znajomość tych kodów. Można je znaleźć w [24].

Fragment listy kodów (kody, które zostały użyte w programie *Wskaźnik Laserowy*) zostały zamieszczone poniżej (Tabela2.):

**Tabela2.** Kody wirtualnych klawiszy (*Virtual-Key Codes*)

| **Kod i wartość (hex)** | **Klawisz** | **Funkcja** |
|---|---|---|
| *VK_ESCAPE* (0x1B) | Escape (ESC) | zakończenie prezentacji |
| *VK_LEFT* (0x25) | Strzałka w lewo | następny slajd |
| *VK_RIGHT* (0x27) | Strzałka w prawo | poprzedni slajd |
| *VK_F5* (0x74) | F5 | włączenie prezentacji (pełny ekran) |
| *VK_SCROLL* (0x91) | Scroll Lock | szybka blokada działania programu; stan klawisza sprawdzany za pomocą funkcji *GetKeyState* [27] |

## 5.4. Przetwarzanie obrazów – biblioteka OpenCV

*OpenCV* (*Open Source Computer Vision*) jest biblioteką funkcji i struktur przygotowanych do komputerowego przetwarzania obrazu [7], [8], [15], [16]. Głównym celem jest przetwarzanie obrazu w czasie rzeczywistym.

Początkowo biblioteka została napisana w języku C, a od wersji 2.0 częściowo również w C++. Jest publikowana wraz z kodem źródłowym na wolnej licencji typu *BSD*. W związku z tym, może być bez przeszkód użyta zarówno w programach niekomercyjnych, jak i komercyjnych (w tym programy z zamkniętym kodem źródłowym) [17].

Oprócz funkcji ściśle związanych z przetwarzaniem obrazów zawiera również inne pomocnicze funkcje i struktury (m.in. do wykonywania obliczeń macierzowych, własny interfejs graficzny). W sumie zaimplementowano ponad 500 funkcji.

Biblioteka *OpenCV* nie jest związana z konkretnym systemem operacyjnym. Może być używana zarówno w systemie *Windows* jak i *Linux*, *MacOS*, *FreeBSD*. Ponieważ została napisana w większości w języku C, możliwe jest także przeportowanie na inne platformy np. procesory sygnałowe [16].

Możliwości biblioteki *OpenCV* [8]:

- Pobieranie obrazów z plików lub kamer
- Zapis obrazów i sekwencji video do plików
- Operacje na obrazach m.in.
  - Podstawowe: filtracja, wykrywanie krawędzi, konwersja kolorów itd.
  - Analiza ruchu
  - Rozpoznawanie obiektów
- Operacje algebraiczne na macierzach (wektorach), oraz inne operacje: wyznacznik macierzy, wektory i wartości własne, rozwiązywanie równań macierzowych
- Elementy graficznego interfejsu użytkownika (wyświetlanie obrazu w oknie, funkcje myszy i klawiatury, suwaki do ustawiania wartości parametrów)
- Rysowanie na obrazie (podstawowe kształty, tekst)

### 5.4.1. Pobieranie obrazu z kamery

Z pobieraniem obrazu związane są następujące funkcje i struktury [10]:

- *cvCaptureFromCAM*
- zmienna (struktura) typu *CvCapture*
- *cvQueryFrame*
- *cvReleaseCapture*

**Funkcja *cvCaptureFromCAM***

```
CvCapture* cvCaptureFromCAM(int index)
```

Tworzy i zwraca wskaźnik do struktury *CvCapture*, która umożliwia pobranie obrazu z wybranej kamery.

Jako argument (*index*) funkcja *cvCaptureFromCAM* przyjmuje numer kamery (numerowane od zera). W przypadku gdy do komputera podłączona jest tylko jedna kamera (lub nie jest istotne, która kamera zostanie wybrana) można podać wartość „-1" lub zdefiniowaną stałą *CV_CAP_ANY*.

**Funkcja *cvQueryFrame***

```
IplImage* cvQueryFrame(CvCapture* capture)
```

Pobiera obraz ze źródła (kamera lub plik video) opisanego w strukturze *CvCapture* i zwraca wskaźnik do obrazu zapisanego jako struktura *IplImage* (5.4.4 s.40). Funkcja *cvQueryFrame* jest połączeniem dwóch funkcji *cvGrabFrame* (pobiera obraz) oraz *cvRetrieveFrame* (dekompresuje zapisaną klatkę i zwracająca wskaźnik do obrazu), które można wywoływać oddzielnie. Oddzielne wywołanie funkcji można wykorzystać przy równoczesnym pobieraniu obrazu z kilku kamer.

Argumentem funkcji jest wskaźnik do struktury *CvCapture*.

**Funkcja *cvReleaseCapture***

```
void cvReleaseCapture(CvCapture** capture)
```

Usuwa strukturę *CvCapture*. Jako argument funkcji podawany jest wskaźnik do usuwanej struktury.

### 5.4.2. Wyświetlanie obrazu z kamery

Do wyświetlania obrazu wystarczy skorzystać z trzech funkcji z biblioteki OpenCV [11]:

- *cvNamedWindow*
- *cvShowImage*
- *cvDestroyWindow*

**Funkcja *cvNamedWindow***

```
int cvNamedWindow(const char* name, int flags)
```

Tworzy nowe okno, lub nie robi nic w przypadku gdy okno o danej nazwie już zostało utworzone.

Przyjmuje 2 argumenty:

- *name*
  Ciąg znaków określający nazwę-identyfikator okna. Za pomocą tego ciągu znaków można się odwoływać do okna przy wyświetlaniu obrazu.
- *flags*
  Dodatkowe ustawienia dotyczące tworzonego okna. Obecnie (*OpenCV* wersja 2.0.0) jedyna możliwa flaga to: *CV_WINDOW_AUTOSIZE*. Gdy jest ustawiona wymiary okna są dopasowywane do zawartości, a użytkownik nie może zmieniać rozmiarów okna.

***Funkcja cvShowImage***

```
void cvShowImage(const char* name, const CvArr* image)
```

Umożliwia wyświetlane obrazu. Gdy dla okna została ustawiona flaga *CV_WINDOW_AUTOSIZE*, obraz jest wyświetlany w oryginalnym rozmiarze, w przeciwnym przypadku obraz jest skalowany do wymiarów okna. Funkcja skaluje również wartości kolorów pikseli na zakres [0..255] gdy dane są zapisane w innym formacie niż 8-bitowy.

Przyjmuje 2 argumenty:

- *name*
  Ciąg znaków będący nazwą okna, w którym ma być wyświetlany obraz.
- *image*
  Wskaźnik do struktury zawierającej obraz (struktura *IplImage*).

**Funkcja *cvDestroyWindow***

```
void cvDestroyWindow(const char* name)
```

Zamyka („niszczy") okno o podanej nazwie w argumencie *name*.

Istnieje również bezargumentowa funkcja (*cvDestroyAllWindows*) „niszcząca" wszystkie okna utworzone przez funkcje biblioteki *OpenCV*.

### 5.4.3. Inne funkcje *OpenCV* używane przy pobieraniu obrazów

**Funkcja *cvWaitKey***

```
int cvWaitKey(int delay=0)
```

Funkcja oczekuje na wciśnięcie klawisza, zwraca kod wciśniętego klawisza lub „-1" gdy żaden klawisz nie został wciśnięty. Czas oczekiwania (w milisekundach) jest określany argumentem *delay*. Przy wartości argumentu ujemnej lub równej 0 funkcja oczekuje do wciśnięcia klawisza (bez ograniczenia czasowego).

Funkcja może być używana do wprowadzania opóźnień w programie np. pomiędzy pobraniem kolejnych klatek obrazu, albo do odczekania na automatyczne ustawienie ekspozycji kamery.

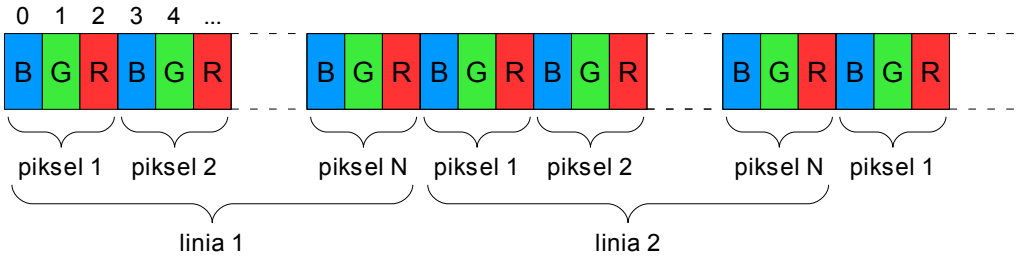
### 5.4.4. Strukutra *IplImage*

Struktura *IplImage* [12] w bibliotece OpenCV służy do przechowywania danych obrazu pobranego np. z kamery. Została wzięta z biblioteki *Intel Image Processing Library*. Część elementów struktury nie jest wykorzystywana przez funkcje *OpenCV*.

Najważniejsze elementy struktury *IplImage*:

- *height*
  Wysokość obrazu (w pikselach).
- *width*
  Szerokość obrazu (w pikselach).
- *nChannels*
  Liczba kanałów – w *OpenCV* używane 1-4 kanałów, dla obrazów pobranych za pomocą kamery USB równe 3 – (RGB).

- *widthStep*
  Liczba bajtów odpowiadająca jednej linii (wierszowi) obrazu. W przypadku obrazu z kamery USB równe: *nChannels*width*. Wartość przydatna przy odwoływaniu się do pikseli za pomocą współrzędnych obrazu (*x*,*y*) – umożliwia odliczanie linii obrazu.
- *imageData*
  Wskaźnik do jednowymiarowej tablicy (do wyrównanych danych – *aligned data*) opisującej kolory pikseli. Dla obrazów z kamery USB dla danego piksela przypadają 3 bajty (3x8 bitów – 3 jedno-bajtowe kanały). Dane są zapisane wierszami zaczynając od lewego górnego rogu obrazu, kolory są zapisane w kolejności *BGR* (niebieski, zielony, czerwony). Kolejność zapisu danych obrazu została przedstawiona na Rys.21.




**Rys.21.** Kolejność zapisu danych obrazu – *imageData* (pole w strukturze *IplImage*)

Pozostałe elementy struktury nie są tak istotne, lub w ogóle niewykorzystane przy programowaniu z użyciem biblioteki *OpenCV*. Opis całej struktury można znaleźć w [12].

Poniższy kod [8] umożliwia dostęp do piksela o współrzędnych (*x*,*y*):

```
// img – wskaźnik do struktury IplImage zawierającej obraz
int step      = img->widthStep;
int channels  = img->nChannels;
int data      = (uchar *)img->imageData;
int k         = 2;          /* kanał czerwony */
// przypisanie do piksela (x,y) wartości 255 na kanale
       czerwonym
data[y*step+x*channels+k] = 255;
```

$0 <= x < width$

$0 <= y < height$

$k = 0$ – dla kanału niebieskiego

$k = 1$ – dla kanału zielonego

$k = 2$ – dla kanału czerwonego

Przykładowy fragment kodu realizujący inwersję (negatyw) obrazu. Kod napisany w oparciu o przykład zamieszczony w [8].

```
IplImage* img = 0;
  int height,width,step,channels;
  uchar *data;
  int i,j,k;

// pobranie danych obrazu
  height    = img->height;
  width     = img->width;
  step      = img->widthStep;
  channels  = img->nChannels;
  data      = (uchar *)img->imageData;
  printf("Processing a %dx%d image with %d
        channels\n",height,width,channels);

// odwrócenie (negatyw)
  for(i=0;i<height;i++) for(j=0;j<width;j++)
        for(k=0;k<channels;k++)
    data[i*step+j*channels+k]=255-data[i*step+j*channels+k];
```

Możliwy jest też wygodniejszy dostęp do danych obrazu za pomocą zdefiniowanych funkcji *OpenCV* [8], jednak taki sposób nie jest tak szybki jak opisany powyżej.

Bardzo szybki dostęp do danych jest szczególnie ważny przy przetwarzaniu obrazu w czasie rzeczywistym, kiedy oprócz odczytania informacji o obrazie wykonywane są inne operacje kilkanaście razy w ciągu sekundy.

Odczyt lub zapis wartości kolorów zwykle jest wykonywany dla wszystkich pikseli na obrazie, co daje kilkaset tysięcy takich operacji dla danej klatki. Np. przy szukaniu maksimum na obrazie z kamery internetowej o typowej rozdzielczości 640x480 konieczne jest wykonanie 307 200 operacji odczytu dla jednego kanału z pojedynczej klatki.

W przypadku oprogramowania do sterowania komputerem, będącego tematem tej pracy, wydajność jest również ważnym problemem. Operacje przetwarzania obrazu muszą być wykonane kilkanaście razy na sekundę bez nadmiernego obciążania komputera, tak żeby możliwe były również inne działania w tym samym czasie.

### 5.4.5. Przykładowy program z wykorzystaniem *OpenCV*

Przykładowy program realizujący pobranie obrazu z kamery i wyświetlenie na ekranie. Kod programu pochodzi z [9].

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

// A Simple Camera Capture Framework
int main() {

	// Zainicjowanie przechwycenia strumienia video
	CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
	if( !capture ) {
		fprintf( stderr, "ERROR: capture is NULL \n" );
		getchar();
		return -1;
	}

	// Stworzenie okna w którym przechwycone obrazy będą wyświetlane
	cvNamedWindow( "Kamera_internetowa", CV_WINDOW_AUTOSIZE );

	// Wyświetlenie w powyższym oknie przechwyconej klatki strumienia
	// wszystko odbywa się w nieskończonej pętli while
	while( 1 ) {
		// Pobierz jedna klatke ze strumienia
		IplImage* frame = cvQueryFrame( capture );
		if( !frame ) {
			fprintf( stderr, "ERROR: frame is null...\n" );
			getchar();
			break;
		}

		// Wyświetl pobraną klatkę
		cvShowImage( "Kamera_internetowa", frame );

		// Oczekiwanie przez 10 ms na wcisniecie klawisza ESC (kod
		// ASCII 27)
		// jeśli klawisz zostanie nacisniety, wtedy program wyskakuje
		// z nieskonczonej petli i konczy dzialanie
		if( cvWaitKey(10) == 27 ) break;
	}

	// Zwalnia strumien video pochodzacy z kamery
	cvReleaseCapture( &capture );

	// Niszczy okno
	cvDestroyWindow( "Kamera_internetowa" );
	return 0;
}
```

### 5.4.6. Operacje na macierzach i wektorach

W bibliotece *OpenCV* zostały zdefiniowane również funkcje i struktury umożliwiające operacje na macierzach i wektorach. [8], [13]. Wektory są definiowane tak samo jak macierz, tylko z jednym wymiarem równym 1.

**Funkcja *cvCreateMat***

```
CvMat* cvCreateMat(int rows, int cols, int type);
```

Tworzy strukturę macierzy i zwraca wskaźnik do zmiennej typu *CvMat*.

Argumenty:

- *rows*
  liczba wierszy
- *cols*
  liczba kolumn
- *type*
  typ danych przechowywanych w macierzy zapisany według wzoru:
  *CV_<liczba_bitów>(S|U|F)C<liczba_kanałów>*
  gdzie:
  *S* – typ całkowity ze znakiem
  *U* – typ całkowity bez znaku
  *F* – typ zmiennoprzecinkowy
  przykładowo:
  *CV_8UC1* – 1 kanał, typ „unsigned char" (8 bitów, całkowity bez znaku)
  *CV_32FC2* – 2 kanały, typ „float" (32 bity, zmiennoprzecinkowy)
  *CV_64FC1* – 1 kanał, typ „double" (64 bity, zmiennoprzecinkowy)

**Funkcja *cvReleaseMat***

```
void cvReleaseMat(CvMat** mat)
```

Usuwa macierz i zwalnia pamięć przez nią zajmowaną.

**Niektóre operacje matematyczne na macierzach**

W bibliotece *OpenCV* zaimplementowano podstawowe operacje algebraiczne na macierzach, takie jak: dodawanie, odejmowanie, mnożenie [13].

Istnieją również funkcje wykonujące inne operacje takie jak obliczenie wyznacznika macierzy, macierzy odwrotnej, wartości i wektorów własnych.

We wszystkich przypadkach funkcje wyglądają bardzo podobnie. Jako argumenty przyjmują 1 lub 2 wskaźniki do macierzy zawierających dane wejściowe i ewentualnie (gdy wynikiem jest macierz lub wektor) wskaźnik do macierzy, w której ma zostać zapisany wynik.

**Dodawanie macierzy**

```
void cvAdd(const CvArr* src1, const CvArr* src2, CvArr* dst,
        const CvArr* mask=NULL)
```

Funkcja *cvAdd* dodaje 2 macierze (*src1* i *src2*), a wynik zapisywany jest w macierzy *dst*. Opcjonalnie można podać jako czwarty argument – maskę która określa które elementy mają być dodane. Wszystkie macierze muszą mieć takie same wymiary.

**Wyznacznik macierzy**

```
double cvDet(const CvArr* mat)
```

Funkcja zwraca wyznacznik macierzy podanej jako argument.

### 5.4.7. Rozwiązywanie równań macierzowych

Oprócz podstawowych operacji na macierzach zaimplementowano również funkcję rozwiązującą równanie macierzowe [13].

```
int cvSolve(const CvArr* src1, const CvArr* src2, CvArr* dst,
        int method=CV_LU)
```

Funkcja *cvSolve* rozwiązuje układ równań postaci:

$$A \cdot x = b \tag{16}$$

gdzie:

$A$ – macierz współczynników $N \times N$

$b$ – macierz (wektor) wyrazów wolnych $N \times 1$

$x$ – macierz (wektor) niewiadomych $N \times 1$

Funkcja jako argumenty przyjmuje wskaźniki do 3 macierzy (wektorów) – 2 wejściowe $A$ i $b$, oraz wynikowa $x$. Dodatkowo opcjonalny czwarty argument odpowiada za wybór metody rozwiązywania równania (obliczenie macierzy odwrotnej). Do wyboru są 3 metody. Domyślnie stosowana jest *metoda LU*.

Przy rozwiązywaniu równania metodą LU, funkcja zwraca wartość „1" gdy macierz *A* jest nieosobliwa (*Nonsingular Matrix* – wyznacznik różny od zera), w przeciwnym przypadku funkcja zwraca wartość „0".

Do odczytania poszczególnych elementów wektora wynikowego można użyć funkcji *cvmGet*, zwracającej konkretny element macierzy lub wektora (przykład poniżej).

**Przykład:**

Dany jest układ równań (17)

$$A \cdot x = b \tag{17}$$

w którym:

$$A = \begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix} \tag{18}$$

$$b = \begin{bmatrix} 5 \\ -3 \end{bmatrix} \tag{19}$$

Kod rozwiązujący powyższy układ równań:

```
// określenie współczynników macierzy A i b
        double A_temp[4]={1, 3, -2, 1}; // a11 a12 a21 a22
        double b_temp[2]={5, -3};       // b1 b2

// utworzenie macierzy (struktur cvMat) - CV_64FC1 <=> double
        // macierz 2x2
        CvMat A = cvMat( 2, 2, CV_64FC1, A_temp );
        // wektor 2-elementowy
        CvMat b = cvMat( 2, 1, CV_64FC1, b_temp );
        // wektor 2-elementowy (wynik)
        CvMat* x = cvCreateMat( 2, 1, CV_64FC1);

// rozwiązanie okładu równań Ax=b ze względu na x
        int rozwiazanie = cvSolve(&A, &b, x, CV_LU);

// wypisanie wyników
        if (rozwiazanie)
        { // gdy rozwiązanie możliwe, wypisanie x1 i x2
            cout << "x1 = " << cvmGet(x,0,0) << endl;
            cout << "x2 = " << cvmGet(x,1,0) << endl;
        }
        else
        { // gdy rozwiązanie niemożliwe
            cout << "Macierz A osobliwa - brak jednoznacznego
              rozwiązania" << endl;
        }
```

Wynik działania programu:

```
x1 = 2
x2 = 1
```

W programie *Wskaźnik Laserowy* funkcja *cvSolve* jest używana do rozwiązania układu równań przy kalibracji (3 s.16), którego wynikiem są współczynniki potrzebne do korekcji zniekształcenia obrazu.

### 5.4.8. Rysowanie

*OpenCV* umożliwia również dodawanie prostych elementów graficznych (linie, prostokąty, elipsy i napisy) do wczytanych obrazów [8].

Zaimplementowane funkcje:

- *cvRectangle* – prostokąt
- *cvCircle* – elipsa
- *cvLine* – linia
- *cvPolyLine* – linia łamana / wielokąt
- *cvFillPoly* – wielokąt (wypełniony)
- *cvPutText* – tekst

**Rysownie linii:**

```
cvLine(img, cvPoint(x1,y1), cvPoint(x2,y2), cvScalar(B,G,R),
       w, CV_AA);
```

Argumenty funkcji:

- *img* – wskaźnik do struktury *IplImage* – obrazu na którym będzie rysowana linia
- *x1,y1* – współrzędne początku
- *x2,y2* – współrzędne końca
- *B,G,R* – współczynniki określające kolor linii (Niebieski, Zielony, Czerwony)
- *w* – grubość linii w pikselach
- *CV_AA* – stała zdefiniowana w *OpenCV* określająca, że linia ma być rysowana z wygładzaniem (*antyaliasing*). Gdy ostatni argument zostanie pominięty linia będzie narysowana bez wygładzania. Można również podać wartość 8 lub 4, określając algorytm rysowania linii [14].

dodatkowo przy wywołaniu funkcji zostały użyte funkcje (konstruktory):

- *cvPoint(x,y)* tworzący strukturę opisującą punkty początkowy i końcowy linii
- *cvScalar(B,G,R)* tworzący strukturę zawierającą współczynniki *RGB* koloru linii.

**Rysowanie prostokąta**

```
cvRectangle(img, cvPoint(x1,y1), cvPoint(x2,y2),
        cvScalar(B,G,R), w);
```

Argumenty funkcji:

- *img* – wskaźnik do struktury *IplImage*
- *x1,y1* – współrzędne jednego z wierzchołków prostokąta
- *x2,y2* – współrzędne przeciwległego wierzchołka (po przekątnej)
- *w* – Gdy dodatnie oznacza grubość linii (w pikselach) rysowanego prostokąta (samo obramowanie). Gdy ujemne rysowany jest prostokąt wypełniony bez krawędzi. Zamiast ujemnej wartości można podać zdefiniowaną stałą *CV_FILLED* [14].

Dodatkowo można podać jeszcze jeden argument określający sposób rysowania krawędzi (tak jak przy linii).

**Rysowanie linii łamanej lub wypełnionego wielokąta.**

Za pomocą funkcji *cvPolyLine* (linia łamana) i *cvFillPoly* (wypełniony wielokąt) można przy jednym wywołaniu narysować od razu kilka figur.

Przykładowy kod pochodzący z [8] – rysowanie linii łamanych:

```
CvPoint  curve1[]={10,10, 10,100, 100,100, 100,10};
CvPoint  curve2[]={30,30, 30,130, 130,130, 130,30, 150,10};
CvPoint* curveArr[2]={curve1, curve2};
int      nCurvePts[2]={4,5};
int      nCurves=2;
int      isCurveClosed=1;
int      lineWidth=1;

cvPolyLine(img,curveArr,nCurvePts,nCurves,isCurveClosed,
        cvScalar(0,255,255),lineWidth);
```

Przy uruchomieniu funkcji trzeba określić kilka zmiennych:

- zbiór współrzędnych wierzchołków osobno dla każdej krzywej – w tym przypadku dwie: *curve1* i *curve2*; współrzędne są zapisane w postaci tablicy jednowymiarowej kolejno: *x1*,*y1*,*x2*,y2, ... *xn*,*yn*.
- wskaźnik do tablicy zawierającej współrzędne wszystkich krzywych (*curveArr*)
- liczbę wierzchołków dla każdej krzywej (*nCurvePts*)
- liczbę krzywych do narysowania
- czy rysowane krzywe mają być zamknięte (połączony koniec z początkiem)
  - tak – gdy *isCurveClosed:*=1,
  - nie – gdy *isCurveClosed*:=0.
- grubość linii (*lineWidth*) w pikselach
- kolor linii np: *cvScalar(0,255,255)* – struktura zawierająca współczynniki *RGB*

Rysowanie wielokąta wypełnionego wygląda bardzo podobnie. Różnica jest taka, że nie określa się grubości linii i czy krzywa ma być zamknięta.

```
cvFillPoly(img,curveArr,nCurvePts,nCurves,cvScalar(0,255,255)
        );
```

## 5.5. Ustawianie parametrów naświetlania obrazu z kamery – biblioteka *DirectShow*.

### 5.5.1. Uwagi dotyczące biblioteki *DirectShow*

W przypadku większości kamer USB jest możliwe programowe („ręczne" obok trybu automatycznego) ustawianie parametrów naświetlania obrazu m.in. czasu naświetlania, czułości oraz innych parametrów takich jak ogniskowa, ostrość.

W systemie *Windows* dostęp do parametrów kamery można uzyskać za pomocą biblioteki *DirectShow* firmy *Microsoft* [41]. Biblioteka *DirectShow* jest przystosowana do tworzenia oprogramowania w języku C++. Żeby z niej skorzystać należy w kodzie programu dołączyć plik nagłówkowy *dshow.h*. Potrzebne pliki nagłówkowe są zawarte w *DirectX SDK (DirectX Software Development Kit)* lub w przypadku nowszych wersji instalowane razem z *Windows SDK*.

Sterowanie pracą kamery jest tylko jednym z zadań biblioteki *DirectShow*, która została zaprojektowana do tworzenia programów multimedialnych korzystających ze strumieni audio i wideo [42]. Ułatwia między innymi:

- odtwarzanie plików multimedialnych,
- pobieranie obrazu i dźwięku z urządzeń zewnętrznych (kamery cyfrowe lub karty telewizyjne),
- kodowanie i dekodowanie strumienia (obsługiwana jest większość popularnych typów plików i formatów kompresji wideo i dźwięku),
- dostęp do przesyłanych danych (poszczególnych klatek wideo) i ich przetwarzanie.

Technologia *DirectShow* jest dość uniwersalna i przewiduje dostęp do praktycznie wszystkich paramentów kamery. Jednak w przypadku większości kamer nie można wykorzystać wszystkich możliwości biblioteki, ze względu na konstrukcję kamery (np. brak opcji zmiany ogniskowej lub ustawienia ostrości).

Żeby móc sterować parametrami kamery konieczne jest uzyskanie w programie wskaźnika do konkretnego urządzenia (typ *IbaseFilter*). W tym celu trzeba wykonać procedurę wyliczenia urządzeń (*Enumerating Devices*), której opis można znaleźć w dokumentacji biblioteki *DirectShow* [30], a przykładową aplikację, korzystającą z *DirectShow* wraz z kodem źródłowym np: w [31].

Przy wyszukiwaniu urządzeń zainstalowanych w systemie ważne jest określenie ich typu np. urządzenia pobierające obraz video (*Video capture devices*) [32].

Aby wybrać konkretne urządzenie, w przypadku gdy w systemie zainstalowane jest więcej niż jedno, można sprawdzać jego nazwę (*FriendlyName*) lub ścieżkę dostępu (*DevicePath*). Nazwa jest charakterystyczna dla danego modelu (może się powtarzać), natomiast ścieżka dostępu jednoznacznie określa konkretne urządzenie – nawet jeśli do komputera jest podłączonych kilka identycznych modeli [32].

### 5.5.2. Ustawianie czasu naświetlania

Ekspozycja kamery może być ustawiana przez zmianę czasu naświetlania obrazu. Taka opcja jest dostępna w większości kamer., a tylko niektóre mają dodatkowo możliwość zmiany przysłony w obiektywie.

Ze względu na to, że przy tworzeniu programu *Wskaźnik Laserowy* została użyta kamera bez możliwości zmiany przysłony, dalej zostanie opisane tylko ustawienie czasu naświetlania.

W *DirectShow* czas naświetlania jest określany w sekundach jako potęga liczby 2 (Tabela3.). Zwykle wartość wykładnika jest ujemna co oznacza czasy naświetlania krótsze od 1 sekundy.

**Tabela3.** Czas naświetlania w *DirectShow*– fragment tabeli z [29]

| **N – Wartość *DirectShow*** | **$2^N$ – Poprawny czas [s]** | **LWS* [s]** | **QC 11.5** [s]** |
|---|---|---|---|
| -14 | 1/16384 | nd. | 1/15 |
| -13 | 1/8192 | nd. | 1/5 |
| -12 | 1/4096 | nd. | 1/5 |
| -11 | 1/2048 | nd. | 1/5 |
| -10 | 1/1024 | nd. | 1/5 |
| -9 | 1/512 | 1/512 | 1/5 |
| -8 | 1/256 | 1/256 | 1/5 |
| -7 | 1/128 | 1/128 | 1/8 |
| -6 | 1/64 | 1/64 | 1/16 |
| -5 | 1/32 | 1/32 | 1/31 |
| -4 | 1/16 | 1/16 | 1/63 |
| -3 | 1/8 | 1/8 | 1/125 |
| -2 | 1/4 | 1/5*** | 1/250 |
| -1 | 1/2 | 1/5*** | 1/500 |
| 0 | 1 | 1/5*** | 1/5 |

Objaśnienia oznaczeń w tabeli:
nd. – niedostępne
**LWS* – *Logitech Webcam Software* – oprogramowanie kamer firmy *Logitech* – dotyczy kamery używanej do testowania oprogramowania (6.1 s.59)
** QC 11.5 – jedna ze starszych wersji oprogramowania *Logitech QuickCam* – poprzednik *LWS*
*** wartości niezgodne z założeniem *DirectShow*.

Wartości mniejsze od -9 czyli czasy naświetlania krótsze od 1/512 nie są używane w większości kamer. Podobnie wartości większe niż -2 (czasy dłuższe niż 1/4). Wartości spoza tego zakresu są niedostępne lub jest przypisana do nich inna wartość niż to wynika z założeń *DirectShow*. W niektórych kamerach czasy naświetlania mogą jednak być określone zupełnie inaczej niż zakłada *DirectShow* - przykładowo w oprogramowaniu *Logitech QuickCam* w wersji 11.5 (Tabela3., s.51 – kolumna „QC 11.5”).

### 5.5.3. *DirectShow* – interfejs *IAMCameraControl*

Ustawianie ekspozycji kamery (i nie tylko) umożliwia interfejs *IAMCameraControl* [33]. Lista wszystkich elementów, które można kontrolować za pomocą tego interfejsu została przedstawiona w dokumentacji [34]

Poniżej został opisany sposób użycia interfejsu *IAMCameraControl* do ustawienia czasu naświetlania w kamerze. Przykładowe fragmenty kodu zostały opracowane na podstawie [33] i [35].

W pierwszej kolejności trzeba zadeklarować i przypisać wartość wskaźnika do interfejsu *IAMCameraControl*. Można to zrobić za pomocą funkcji *QueryInterface*.

```
IAMCameraControl *pCameraControl
hr = pFilter_kamera->QueryInterface(IID_IAMCameraControl,
        (void **)&pCameraControl);
```

W powyższym kodzie został użyty wskaźnik do urządzenia (kamery) – *pFilter_kamera* typu *IBaseFilter**, który został wcześniej zapisany podczas wyszukiwania urządzeń do pobierania obrazu.

Funkcja *QueryInterface* (tak jak większość funkcji z biblioteki *DirectShow*) zwraca zmienną typu *HResult*, której wartość informuje o prawidłowym wykonaniu funkcji lub ewentualnym błędzie. Poprawność wykonania funkcji najłatwiej sprawdzić za pomocą jednej z poniższych instrukcji:

Sposób 1.

```
if (hr==S_OK)
```

Sposób 2.

```
if (SUCCEEDED(hr))
```

W przypadku niepowodzenia można od razu przerwać działanie programu unikając dalszych błędów.

Do korzystania z interfejsu *IAMCameraControl* zostały zdefiniowane 3 funkcje:

- *GetRange* [36] – pobranie zakresu możliwych wartości
- *Get* [37] – pobranie aktualnie ustawionej wartości
- *Set* [38] – zapisanie nowej wartości

Przed pierwszym wywołaniem funkcji *Set* lub *Get* dobrze jest pobrać zakres możliwych wartości za pomocą funkcji *GetRange*.

Przykłady dla czasu naświetlania:

```
hr = pCameraControl->GetRange(
        CameraControl_Exposure,   /* czas naświetlania */
        &CamExpMin,  /* minimalna dopuszczalna wartość */
        &CamExpMax,  /* maksymalna dopuszczalna wartość */
        &CamExpDelta,       /* krok zmian wartości */
        &CamExpDefault,     /* wartość domyślna */
        &CamExpFlags);      /* flaga: „auto” lub „manual” */
```

```
hr = pCameraControl->Get(
        CameraControl_Exposure, /* czas naświetlania */
        &CamExpValue,           /* aktualna wartość */
        &CamExpFlags);          /*aktualny tryb(auto/manual)*/
```

Zmiana czasu naświetlania

```
pCameraControl->Set(
        CameraControl_Exposure,     /* czas naświetlania */
        CamExpValue,                /* ustawiana wartość */
        CameraControl_Flags_Manual); /* ustawianie ręczne */
```

Przełączenie do trybu automatycznego ustawiania czasu naświetlania

```
pCameraControl->Set(
        CameraControl_Exposure,    /* czas naświetlania /
        0,                         /* dowolna wartość */
        CameraControl_Flags_Auto); /* ustawianie automat. */
```

Drugi argument funkcji *Set* w przypadku przełączania na tryb automatyczny nie ma znaczenia.. Można podać np. 0 albo wartość domyślną pobraną przez funkcję *GetRange*. Czas naświetlania jest ustawiany automatycznie od razu po uruchomieniu funkcji.

Wszystkie argumenty wyżej opisanych funkcji są typu *long* – wejściowe lub wyjściowe (podane przez referencję). Pierwszy argument wszystkich trzech funkcjach określa czego dotyczy wywołanie funkcji. W powyższych przykładach jest to stała *CameraControl_Exposure,* czyli czas naświetlania. Inne dopuszczalne wartości można znaleźć w dokumentacji [34].

Ostatni argument to flaga, określająca czy dana właściwość ma być ustawiana automatycznie (przez sterownik kamery) czy ręcznie (przez oprogramowanie korzystające z kamery). Możliwe wartości:

- *CameraControl_Flags_Auto*
- *CameraControl_Flags_Manual*

Pozostałe zmienne: *CamExpMin*, *CamExpMax*, *CamExpDefault*, *CamExpValue*, *CamExpDelta*, przechowują odpowiednio wartość maksymalną, minimalną, domyślną, aktualną i krok z jakim można zmieniać daną wartość.

Automatyczne ustawienie ekspozycji (3.2. s.16) w programie *Wskaźnik Laserowy* zakłada, że czasy ekspozycji w kamerze są zgodne z *DirectShow*, ewentualnie wystarczy, że są tym krótsze im mniejsza jest wartość ekspozycji *DirectShow*. W przypadku takim jak np. QC 11.5 (Tabela3., s. 51 – kol. QC 11.5) automatyczne ustawianie ekspozycji może nie działać poprawnie i pozostaje ręczne ustawianie czasu naświetlania (7.5. s.62).

### 5.5.4. Ustawianie czułości kamery

Oprócz czasu naświetlania istnieje możliwość ustawienia czułości kamery. Daje to większą dokładność przy ustawianiu jasności obrazu i powiększa zakres jasności wynikający tylko z czasu naświetlania. Jednak przy większej czułości pogarsza się jakość obrazu (pojawiają się wyraźne szumy), dlatego w pierwszej kolejności najlepiej jest ustawiać jasność obrazu za pomocą czasu naświetlania.

W *DirectShow* czułość jest określana przez zakres liczb całkowitych, który nie jest precyzyjnie określony w dokumentacji *DirectShow* – wynika ze sterownika danego urządzenia. Większe wartości odpowiadają wyższej czułości, a mniejsze – niższej. Dla kamery użytej do testowania programu (6.1 s.59) możliwe do ustawienia wartości czułości wynoszą od 0 do 255 (z krokiem co 1).

### 5.5.5. DirectShow – interfejs *IAMVideoProcAmp*

Czułość kamery można ustawiać za pomocą interfejsu *IAMVideoProcAmp* [39]. Lista wszystkich elementów, które można kontrolować za pomocą interfejsu *IAMVideoProcAmp* została przedstawiona w dokumentacji [40].

Sposób użycia tego interfejsu jest niemal identyczny jak opisanego wcześniej interfejsu *IAMCameraControl* (5.5.3 s.52). Różnice występują jedynie w nazwach funkcji i stałych. Ogólnie zmiana czułości wygląda bardzo podobnie jak zmiana czasu naświetlania, dlatego poniżej zostały zamieszczone jedynie przykładowe fragmenty kodu. Użyte są w tym celu tylko inne funkcje i stałe, ale idea jest taka sama.

Przykład:

- pobranie wskaźnika do interfejsu (funkcja *QueryInterface*),
- pobranie zakresu możliwych wartości czułości (funkcja *GetRange*)
- pobranie aktualnych wartości (funkcja *Get*):

```
IAMVideoProcAmp *pVideoProcAmp;

hr = pFilter_kamera->QueryInterface(IID_IAMVideoProcAmp,
        (void **)&pVideoProcAmp);

if(hr==S_OK)
{       hr = pVideoProcAmp->GetRange(
               VideoProcAmp_Gain,
               &CamGainMin,
               &CamGainMax,
               &CamGainDelta,
               &CamGainDefault,
               &CamGainFlags);

        hr = pVideoProcAmp->Get(
               VideoProcAmp_Gain,
               &CamGainValue,
               &CamGainFlags);
}
```

- Ustawienie nowej wartości (funkcja *Set*):

```
pVideoProcAmp->Set(
        VideoProcAmp_Gain,
        CamGainValue,
        VideoProcAmp_Flags_Manual);
```

## 5.6. Struktura plików źródłowych projektu

WskaznikLaserowy.cpp

- stdafx.h
  - kamera.h
    - FormKalibracja.h + FormKalibracja.resx
  - mysz.h
    - FormMenu.h + FormMenu.resx
- Form1.h + Form1.resx
  - FormTestEkranu.h + FormTestEkranu.resx

### 1. WskaznikLaserowy.cpp

Główny plik programu wygenerowany przez *Visual C++*, zawierający funkcję *main* i dyrektywy *#include* podłączające pozostałe pliki.

### 2. stdafx.h

Zawiera dyrektywy *#include* do podłączenia używanych plików nagłówkowych (bibliotek).

### 3. kamera.h

Plik z definicją klasy *Class_Kamera*. Jest to klasa zaimplementowana na potrzeby programu *Wskaźnik Laserowy*. Zawiera zmienne i funkcje związane z pobieraniem i przetwarzaniem obrazu z kamery.

***Class_Kamera* – zmienne:**

- współczynniki przekształcenia (obliczone przy kalibracji)
- zmienne związane z biblioteką *DirectShow* (5.5. s.50)
- zmienna typu *CvCapture* (5.4.1. s.38)
- zmienna typu *IplImage* (5.4.4. s.40) przechowująca maskę maksimum (3.6. s.20)
- pomocnicze zmienne np: nazwa aktualnie wybranej kamery, zmienna logiczna określająca czy została wykonana kalibracja kamery.

***Class_Kamera* – funkcje:**

- konstruktor klasy ustawiający początkowe wartości zmiennych
- funkcje związane z odczytem i ustawianiem parametrów naświetlania obrazu, w tym funkcja ustawiająca ekspozycję i czułość kamery (3.2. s.16)
- funkcja odpowiedzialną za kalibrację kamery

- funkcje do uruchomienia i zatrzymania pobierania obrazu z kamery
- funkcja wyszukującą maksimum na obrazie z kamery
- funkcja korygująca zniekształcenie perspektywiczne
- funkcja uśredniająca współrzędne położenia wskaźnika laserowego
- funkcje umożliwiające wybór urządzenia (kamery)

**4. mysz.h**

Plik nagłówkowy zawierający definicję klasy *Class_Mysz*. Klasa zawiera funkcje wykonujące określone zadania na podstawie położenia (współrzędnych) wskaźnika laserowego na ekranie. Wybór odpowiedniej funkcji następuje wewnątrz głównej pętli programu, na podstawie aktualnie wybranego trybu pracy programu. Dla każdego trybu pracy została zaimplementowana oddzielna funkcja.

**5. Form1.h**

- Funkcje głównego okna programu (funkcje przycisków i innych kontrolek).
- Główna pętla programu (4 s.23) zdefiniowana jako funkcja *timera* (5.2 s. 33).
- Funkcje opisujące wygląd okna programu, wygenerowane przez *Visual C++*.

**6. Form1.resx**

wygląd głównego okna programu i jego właściwości

**7. FormKalibracja.h**

Funkcje związane z oknem kalibracji kamery – pokazywanie, ukrywanie, ustawianie rozmiarów, wyświetlanie znaczników w narożnikach itp.

**8. FormKalibracja.resx**

Wygląd okna kalibracji (białe tło z zaznaczonymi narożnikami).

**9. FormMenu.h**

Funkcje związane z wyświetlaniem menu kursora (Tryb Menu, s.28) i funkcjami przycisków tego menu. Działanie przycisków opiera się na programowym wywołaniu odpowiednich zdarzeń myszy i klawiatury (5.3. s.34).

**10. FormMenu.resx**

Wygląd i właściwości menu wyświetlanego przy aktywnym *trybie menu* (Tryb Menu, s.28).

**11. FormTestEkranu.h**

Funkcje odpowiadające za wyświetlanie okna „Test Ekranu" – Rys.24. (s.61).

**12. FormTestEkranu.resx**

Wygląd i właściwości okna „Test Ekranu".

Pozostałe (nie wymienione wyżej) pliki zostały w całości wygenerowane przez *Visual C++* lub nie są bezpośrednio związane z kodem źródłowym np. plik projektu, ikona aplikacji, plik „ReadMe.txt".

# 6. Specyfikacja sprzętu użytego do testów oprogramowania

## 6.1. Kamera

Program był testowany przy użyciu kamery internetowej *Logitech Webcam 200*.

Najważniejsze cechy kamery to [28]:

- Matryca VGA (640 x 480 pikseli)
- Nagrania wideo do 30 klatek na sekundę (przy zalecanych systemach)
- Zgodność ze standardem USB 2.0
- Ręczne ustawianie ostrości

Jest to typowa kamera internetowa, o niezbyt wysokiej jakości obrazu. Jednak do uruchomienia programu w większości przypadków jest wystarczająca. Mała rozdzielczość kamery negatywnie wpływa na dokładność działania programu, ale mniej obciąża komputer (mniej operacji do wykonania).

Kamera ma możliwość programowego (oprócz trybu automatycznego) kontrolowania ekspozycji przez zmianę czasu naświetlania (5.5.2 s.51).

- Najkrótszy czas naświetlania – 1/512 [s]
- Najdłuższy czas naświetlania – 1/5 [s]

Powyższe dane pochodzą z tabeli czasów naświetlania [29].

Ponadto jasność obrazu z kamery można regulować przez zmianę czułości (5.5.4 s.54).

## 6.2. Komputer

Program był testowany na komputerze typu laptop (Rys.1. – s.8).

Najważniejsze parametry komputera ze względu na działanie programu *Wskaźnik Laserowy*:

- System operacyjny: Windows 7 (32-bit)
- Procesor: Intel Core 2 Duo T7250 (2.0 GHz)
- Pamięć RAM: 2GB
- Matryca LCD 14'' o rozdzielczości 1440x900 px

# 7. Instrukcja obsługi programu

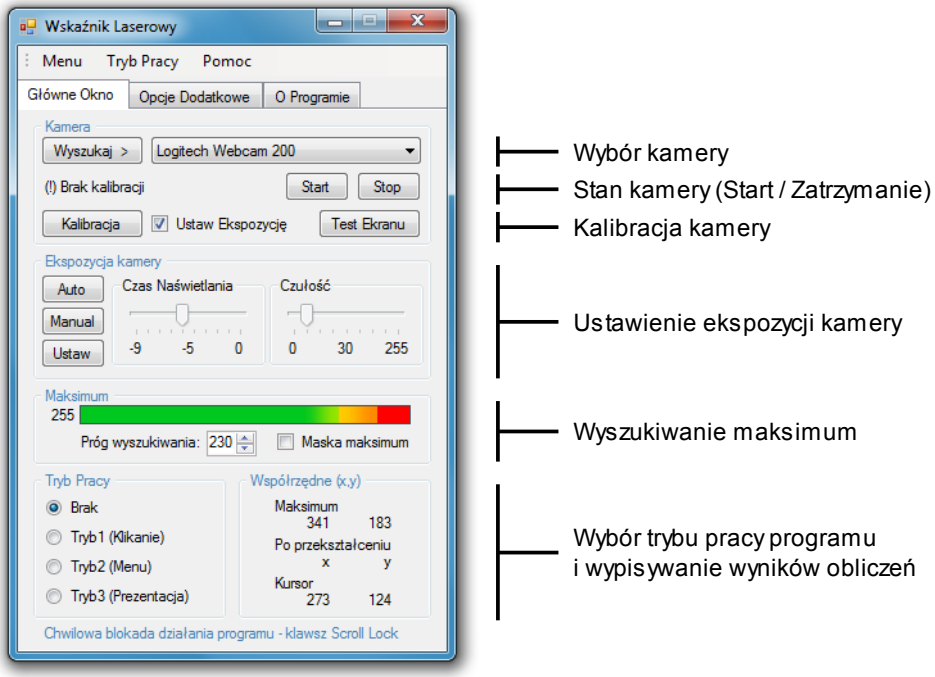
## 7.1. Wymagania systemowe

Wymagania konieczne do uruchomienia programu:

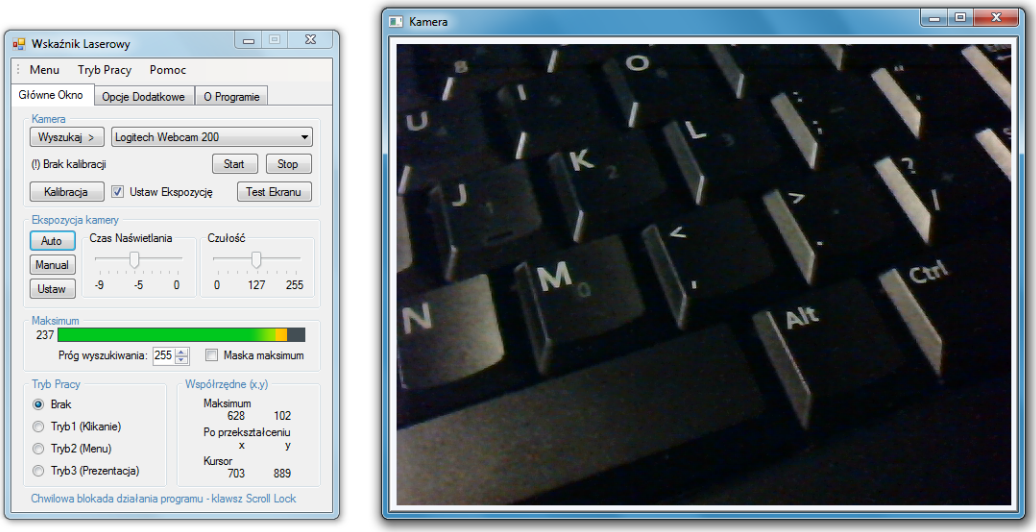
- System *Windows 2000/Xp/Vista/7*
- Biblioteki *DirectX* oraz *.NET Framework 2.0* (obie zwykle są zainstalowane w systemie)
- Kamera – zalecane co najmniej 20 klatek na sekundę i rozdzielczość 640x480px.

## 7.2. Uruchomienie programu

Program można uruchomić za pomocą pliku *WskaznikLaserowy.exe*. Zaraz po uruchomieniu pojawia się główne okno programu (Rys.22.). Jeśli kamera nie została podłączona wcześniej, można to zrobić teraz i kliknąć przycisk „Wyszukaj". Po tym zostanie wczytana lista aktualnie podłączonych do komputera kamer. W przypadku gdy podłączone jest więcej niż jedna kamera, odpowiednią można wybrać z listy obok. Jeśli kamera działa poprawnie pojawi się okno z podglądem obrazu (Rys.23.).




**Rys.22.** Główne okno programu

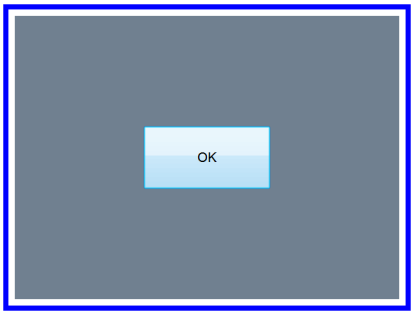



**Rys.23.** Okno główne programu (z lewej) i okno podglądu obrazu z kamery (z prawej)

## 7.3. Ustawienie kamery

Kamerę należy skierować na ekran. Najlepiej ustawić kamerę możliwie najbardziej na wprost ekranu. Jeżeli kamera nie ma automatycznego ustawiania ostrości (*autofocus*), trzeba ustawić ostrość „ręcznie" za pomocą pokrętła na obudowie kamery. Przy źle ustawionej ostrości mogą wystąpić problemy w działaniu programu.

Ze względu na sposób kalibracji, każdy narożnik ekranu musi być widoczny i znajdować się w osobnej ćwiartce obrazu z kamery. Poprawne ustawienie kamery zostało przedstawione wcześniej na Rys.9. (s.19). Do stwierdzenia czy obraz z komputera faktycznie jest wyświetlany w całości (nie jest ucięty np. przez ustawienie ekranu) można skorzystać z przycisku „Test Ekranu" – zostanie wyświetlona ramka jak na Rys.24.




**Rys.24.** Ramka wyświetlana po kliknięciu przycisku „Test Ekranu"

## 7.4. Kalibracja

Po każdym uruchomieniu programu i po każdej zmianie położenia kamery względem ekranu konieczne jest wykonanie kalibracji. Do jej uruchomienia służy przycisk „Kalibracja" z głównego okna programu (Rys.22.).

Podczas kalibracji wyszukiwane są narożniki ekranu i obliczane współczynniki przekształcenia do przeliczania położenia lasera na ekranie na położenie kursora. Dodatkowo jeśli w głównym oknie zaznaczona jest opcja „Ustaw ekspozycję", dopasowywana jest jasność obrazu z kamery na potrzeby programu. Ustawianie ekspozycji można również zrobić ręcznie (7.5. s.62).

## 7.5. Ręczne ustawienie ekspozycji kamery

Domyślnie ekspozycja kamery jest ustawiana podczas kalibracji. W szczególnych przypadkach można ją ustawić „ręcznie" za pomocą dwóch suwaków: „Czas naświetlania" i „Czułość".Przesunięcie suwaka w lewo przyciemnia obraz. Zakresy na suwakach zależą od wybranej kamery. Przy ręcznym ustawianiu ekspozycji pomocny jest pasek graficznie reprezentujący maksimum (Rys.25.). Powinien być cały zielony lub lekko wchodzić w żółty obszar., wtedy jasność obrazu jest odpowiednia i program powinien poprawnie wykrywać laser. Nie jest zalecane ustawianie czasów naświetlania dłuższych od okresu wykonywania głównej pętli programu. Przy domyślnych ustawieniach najdłuższy sensowny czas naświetlania to 1/32s (wartość -5). Co do czułości nie ma szczególnych ograniczeń, jednak przy wysokiej czułości pojawia się więcej szumów na obrazie.

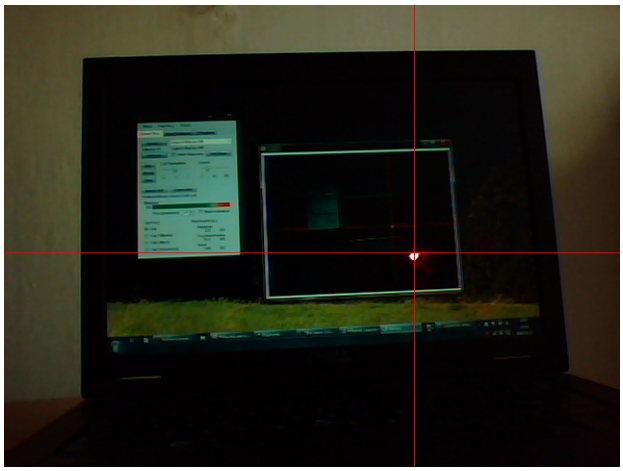

**Rys.25.** Pasek graficznie pokazujący maksimum – jasność obrazu odpowiednia (z lewej), obraz za jasny (z prawej)

Domyślnie suwaki „Czas naświetlania" i „Czułość" są nieaktywne oznacza to, że kamera dobiera jasność obrazu automatycznie. Przełączenie w tryb „ręczny" umożliwia przycisk „Manual". Do trybu automatycznego można powrócić za pomocą przycisku „Auto". Przycisk „Ustaw" odpowiada za ustawienie ekspozycji przez program *Wskaźnik Laserowy* – sposób działania jest taki sam jak przy kalibracji (3.2. s.16).

## 7.6. Działanie programu

Po wykonaniu kalibracji można przetestować poprawność działania programu. W oknie obrazu laser powinien być zaznaczany za pomocą dwóch czerwonych linii (pionowej i poziomej – Rys.26.). Linie powinny być rysowane tylko

w przypadku gdy laser jest widoczny na obrazie. Jeśli tak nie jest rozwiązaniem może być przyciemnienie obrazu albo zmiana progu wyszukiwania lasera.



**Rys.26.** Zrzut ekranu z działającego programu – widoczny laser zaznaczony za pomocą dwóch czerwonych linii.

Jeśli laser jest poprawnie wykrywany można już uruchomić sterowanie komputerem za pomocą lasera. Uruchomienie polega na wybraniu jednego z trybów pracy w głównym oknie programu. Po włączeniu trybu „Menu" albo „Klikanie" kursor na ekranie zacznie podążać za wskaźnikiem laserowym. Położenie kursora jest uśredniane, co zapewnia płynniejszy ruch kursora (minimalizuje wpływ drgań lasera). Uśrednianie można wyłączyć w zakładce „Opcje Dodatkowe". W trybie „Prezentacja" domyślnie kursor nie jest ustawiany. Tryby pracy programu zostały szczegółowo opisane w rozdziale 4.6. (s.27).

Działanie programu można w każdej chwili szybko zablokować przez włączenie klawisza *Scroll Lock*. Dzięki temu można zawsze odzyskać kontrolę nad komputerem, w przypadku gdy laser zacznie być błędnie wykrywany. Podczas normalnej pracy programu, *Scroll Lock* musi być wyłączony (zgaszona odpowiednia dioda na klawiaturze).

Niektóre programy również używają klawisza *Scroll Lock* do sygnalizowania (za pomocą diody) lub przełączania jakiegoś tryby pracy. Takie programy powinny być wyłączone podczas działania programu *Wskaźnik Laserowy*.

## 7.7. Zakończenie działania programu

Żeby zakończyć działanie programu (sterowanie komputerem za pomocą lasera) wystarczy wybrać z listy trybów „Brak" lub zamknąć główne okno.

# 8. Podsumowanie

## 8.1. Uwagi dotyczące działania programu

Program *Wskaźnik Laserowy* spełnia swoje zadanie, jednak nie w każdych warunkach działa poprawnie. Problem pojawia się na przykład przy bardzo jasnym obrazie z projektora. Kontrast pomiędzy plamką lasera a obrazem na ekranie może być niewystarczający do komfortowego używania programu. Na pewno w jakimś zakresie ograniczeniem jest też kamera o niezbyt wysokiej jakości obrazu. O ile dla oka plamka lasera zwykle jest bardzo wyraźna, to na obrazie z kamery już nie zawsze. Możliwe, że pomocne byłoby zastosowanie jakiejś innej metody wyszukiwania lasera nie tylko w oparciu o maksimum.

Wykonane testy użytkowania programu pokazują, że jest możliwe zastąpienie myszy czy nawet klawiatury (np. stosując klawiaturę ekranową). Zwykle jednak precyzja działania jest znacznie mniejsza niż typowych urządzeń, ale wystarczająca m.in. do uruchomienia programu za pomocą ikony na pulpicie, czy wybrania jakiejś opcji z menu. Głównym ograniczeniem są tu drgania lasera trzymanego w ręce.

Przy testowaniu programu ujawniła się pewna cecha monitorów komputerowych. Laser od takiego ekranu odbija się mniej więcej jak od lustra, czyli większa część wiązki lasera pod jednym kątem, a tylko niewielka część ulega rozproszeniu. Efekt występuje praktycznie taki sam niezależnie od typu monitora: kineskop czy LCD, matowy czy błyszczący. Dlatego przy korzystaniu z monitora komputerowego konieczne jest świecenie laserem na wprost ekranu w dość ograniczonym zakresie kątowym. Taki problem nie występuje przy wyświetlaniu obrazu z projektora. Powierzchnia ekranu zwykle dobrze rozprasza światło i plamka lasera jest widoczna praktycznie pod dowolnym kątem.

## 8.2. Dodatkowe usprawnienia programu

Program przede wszystkim wymagałby przetestowania z większą ilością urządzeń. O ile różnice pomiędzy komputerami nie powinny powodować większych utrudnień, to dużym ograniczeniem mogą się okazać kamery w których np. sposób ustawiania ekspozycji może się znacząco różnić (Tabela3. s.51). Konieczne byłoby dopasowanie odpowiednich funkcji do różnych urządzeń lub nawet zaimplementowanie różnych ich wariantów.

Z punktu widzenia użytkownika przydaną opcją może być ograniczenie działania programu do wybranego obszaru na ekranie. Taka funkcja miałaby znaczenie w przypadku gdy nie jest możliwe wyświetlenie obrazu w całości np. z powodu za małej odległość projektora od ekranu.

Inną przydatną funkcją, która może mieć zastosowanie, jest zmiana rozmiarów, położenia, czy nawet zdefiniowanie dodatkowych stref działania wskaźnika laserowego (Rys.20. s.30). Dałoby to użytkownikowi możliwość dopasowania programu do własnych potrzeb.

Ważną rzeczą byłoby na pewno przetestowanie programu w różnych nietypowych sytuacjach, takich jak zmiana przez użytkownika rozdzielczości ekranu w dowolnym momencie pracy programu, i ewentualne zaimplementowanie odpowiedniej reakcji pozwalającej uniknąć zawieszenia programu czy nawet zablokowania komputera.

## 8.3. Możliwość praktycznego zastosowania

Jeśli napisany program miałby znaleźć praktyczne zastosowanie np. przy wspomaganiu prezentacji, to pewnym utrudnieniem jest kalibracja i ustawienie kamery. Kalibracja musi być wykonywana przy każdym uruchomieniu programu i po każdej zmianie położenia kamery względem ekranu. Dodatkowo konieczne jest zapewnienie niezmiennego położenia kamery względem ekranu podczas pracy programu.

Dość trudne byłoby zaimplementowanie ciągłego śledzenia położenia ekranu na obrazie z kamery, głównie dlatego że obraz na ekranie może się dość mocno zmieniać, lub być chwilowo nawet całkowicie wygaszony. Dodatkowo takie rozwiązanie byłoby skuteczne w dość małym zakresie, czyli pod warunkiem, że ekran cały czas jest obserwowany przez kamerę w całości.

Znacznie większą wygodę użytkowania zapewniłaby kamera zamontowana na stałe (np. połączenie z projektorem), wtedy wystarczyłoby wykonać kalibrację raz (po zainstalowaniu urządzeń) a wyniki przechowywać w pliku z ustawieniami programu. Po każdym uruchomieniu byłyby wczytywane i nie byłaby już konieczna kalibracja, a program byłby od razu gotowy do działania.

Kolejną rzeczą, która na pewno byłaby przydatna, to ciągłe sprawdzanie jasności obrazu z kamery i ewentualnie na bieżąco wprowadzanie korekty ekspozycji kamery. Zminimalizowałoby to wpływ zmian warunków oświetleniowych (np. w różnych porach dnia).

Nawet bez wprowadzania wyżej wymienionych usprawnień program *Wskaźnik Laserowy* może wspomagać prezentację. Główną zaletą programu jest to, że do działania nie jest wymagany żaden specjalistyczny sprzęt. Wystarczające są: dowolna kamera podłączana przez USB i wskaźnik laserowy, który w większości przypadków i tak jest używany przy prezentacjach. Po uruchomieniu programu, wskaźnik laserowy ma podwójną rolę: wskazywanie treści na slajdach i zastąpienie myszki lub klawiatury do sterowania pokazem slajdów.

# 9. Bibliografia

Przetwarzanie obrazów

[1] John C. Russ
***The Image Processing Handbook*** – *Fifth Edition*
CRC Press, Taylor & Francis Group,
Boca Raton, FL USA, 2006
ISBN: 978-0-8493-7254-4

[2] Wilhelm Burger, Mark James Burge
***Digital Image Processing*** – *An Algorithmic Introduction Using Java*
Springer,
New York, NY USA, 2008
ISBN: 978-1-84628-379-6
+ errata: http://www.imagingbook.com/index.php?id=103

[3] Paul S. Heckbert
***Fundamentals of Texture Mapping and Image Warping*** – *Master's thesis*
Dept. of Electrical Engineering and Computer Science, University of California,
Berkeley, CA USA, June 1989
http://www.cs.cmu.edu/~ph/texfund/texfund.pdf

[4] Bernardo F. Reis, João Marcelo X. N. Teixeira, Veronica Teichrieb, Judith Kelner
***Perspective Correction Implementation for Embedded (Marker-Based) Augmented Reality***
Universidade Federal de Pernambuco, Centro de Informática, Grupo de Pesquisa em Realidade Virtual e Multimídia
http://www2.fc.unesp.br/wrva/artigos/50227.pdf

[5] ***Four Corner Image Warping***
http://www.fmwconcepts.com/imagemagick/bilinearwarp/FourCornerImageWarp2.pdf

[6] ***Zniekształcenia Bilinearne*** – *Bilinear Distort(Efekt perspektywy)*
http://4programmers.net/PHP/Zniekształcenia_Bilinearne-Bilinear_Distort(Efekt_perspektywy)

OpenCV

[7] Gary Bradski, Adrian Kaehler
***Learning OpenCV***
O'Reilly Media,
Sebastopol, CA USA, 2008
ISBN: 978-0-596-51613-0

[8] Gady Agam
***Introduction to programming with OpenCV***
Department of Computer Science, Illinois Institute of Technology,
January 27, 2006
http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/index.html

[9] Filip Romanowski
***Krótki wstęp do biblioteki OpenCV –***
*Jak wykorzystać kamerkę internetową do przetwarzania obrazów*
Koło Naukowe Robotyków,
Wrocław, 13.10.2008r.
http://konar.pwr.wroc.pl/uploads/download/pdf/OpenCV.pdf

[10] ***Reading and Writing Images and Video***
http://opencv.willowgarage.com/documentation/reading_and_writing_images_and_video.html

[11] ***User Interface***
http://opencv.willowgarage.com/documentation/user_interface.html

[12] ***Basic Structures***
http://opencv.willowgarage.com/documentation/basic_structures.html

[13] ***Operations on Arrays***
http://opencv.willowgarage.com/documentation/operations_on_arrays.html

[14] ***Drawing Functions***
http://opencv.willowgarage.com/documentation/drawing_functions.html

[15] ***OpenCV – Wiki***
http://opencv.willowgarage.com/wiki/

[16] ***OpenCV***
http://en.wikipedia.org/wiki/OpenCV

[17] ***Licencja BSD***
http://pl.wikipedia.org/wiki/Licencja_BSD

Visual C++

[18] ***Visual C++ 2005 Express Edition***
http://www.dobreprogramy.pl/Visual-C,Program,Windows,11894.html

[19] ***How to: Run Procedures at Set Intervals with the Windows Forms Timer Component***
http://msdn.microsoft.com/en-us/library/3tszykws(VS.80).aspx

[20] ***Limitations of the Windows Forms Timer Component's Interval Property***
http://msdn.microsoft.com/en-us/library/xy0zeach(v=VS.80).aspx

Mysz i Klawiatura

[21] ***SetCursorPos Function***
http://msdn.microsoft.com/en-us/library/ms648394(VS.85).aspx

[22] ***mouse_event Function***
http://msdn.microsoft.com/en-us/library/ms646260(VS.85).aspx

[23] ***keybd_event Function***
http://msdn.microsoft.com/en-us/library/ms646304(VS.85).aspx

[24] ***Virtual-Key Codes***
http://msdn.microsoft.com/en-us/library/ms645540(VS.85).aspx

[25] ***MapVirtualKey Function***
http://msdn.microsoft.com/en-us/library/ms646306(VS.85).aspx

[26] ***Key Scan Codes***
http://msdn.microsoft.com/en-us/library/aa299374(VS.60).aspx

[27] ***GetKeyState Function***
http://msdn.microsoft.com/en-us/library/ms646301(VS.85).aspx

Kamera

[28] ***Logitech Webcam C200*** – *Specyfikacja*
http://www.logitech.com/pl-pl/webcam_communications/webcams/devices/5865

Direct Show

[29] ***Questions about accessing the exposure time through DirectShow***
http://www.quickcamteam.net/documentation/faq/questions-about-accessing-the-exposure-time-through-directshow

[30] ***Using the System Device Enumerator***
http://msdn.microsoft.com/en-us/library/dd407292(VS.85).aspx

[31] ***Video Preview and Frames Capture to Memory with SampleGrabber in Buffered Mode*** – *przykład programu do pobierania obrazu z kamery z użyciem DirectShow*
http://www.codeproject.com/KB/audio-video/video_capture.aspx

[32] ***Selecting a Capture Device***
http://msdn.microsoft.com/en-us/library/dd377566(v=VS.85).aspx

[33] ***IAMCameraControl Interface***
http://msdn.microsoft.com/en-us/library/dd389145(v=VS.85).aspx

[34] ***CameraControlProperty Enumeration***
http://msdn.microsoft.com/en-us/library/dd318253(v=VS.85).aspx

[35] ***IAMCameraControl Interface***
http://www.ab-soft.com/AGEHELP/agehelp.html
> DirectShow > Interfaces > IAMCameraControl

[36] ***IAMCameraControl – GetRange Method***
http://msdn.microsoft.com/en-us/library/dd389147(v=VS.85).aspx

[37] ***IAMCameraControl – Get Method***
http://msdn.microsoft.com/en-us/library/dd389146(v=VS.85).aspx

[38] ***IAMCameraControl – Set Method***
http://msdn.microsoft.com/en-us/library/dd389148(v=VS.85).aspx

[39] ***IAMVideoProcAmp Interface***
http://msdn.microsoft.com/en-us/library/dd376033(v=VS.85).aspx

[40] ***VideoProcAmpProperty Enumeration***
http://msdn.microsoft.com/en-us/library/dd407328(v=VS.85).aspx

[41] ***DirectShow***
http://msdn.microsoft.com/en-us/library/dd375454(VS.85).aspx

[42] ***DirectShow***
http://pl.wikipedia.org/wiki/DirectShow

# 10. Załączniki

## 10.1. Skrypty programu *Matlab*

Kod skryptu użyty do przekształcenia dwuliniowego obrazu (Rys.4. s.13)

```
clear all; clc;

%% wczytanie obrazu
% obraz 768px x 768px, odcienie szarosci (1 kanal)
obraz = imread('obraz.png');
rozmiar = 768;
figure(1); imshow(obraz);

       %u  %v  - wspolrzedne zrodlowe
Z = [   1   1;
      768   1;
      768 768;
        1 768];

       %x  %y  - wspolrzedne docelowe
D = [  50 100;
      750 150;
      700 500;
      200 750];

%% przeksztalcenie dwuliniowe
% obliczenie wspolczynnikow przeksztalcenia
% M - macierz przeksztalcenia
M = [Z Z(:,1).*Z(:,2) ones(4,1)];
a = M\D(:,1); % rozwiazanie Ma=D (Ax=b) - metoda Gaussa
b = M\D(:,2); % rozwiazanie Mb=D (Ax=b) - metoda Gaussa

% przeksztalcenie dwuliniowe
obraz_przeksztalcony = imread('tlo.png')'; % ustawienie szarego tla
for u=1:rozmiar
    for v=1:rozmiar
       x = round( a(1)*u + a(2)*v + a(3)*u*v + a(4) );
       y = round( b(1)*u + b(2)*v + b(3)*u*v + b(4) );
       if x>=1 && x<=rozmiar && y>=1 && y<=rozmiar
           obraz_przeksztalcony(y,x) = obraz(v,u); % odwrocone wspolrzedne
       end
    end
end

imwrite(obraz_przeksztalcony,'obraz_bilinear.png','PNG');
figure(2); imshow(obraz_przeksztalcony);
```

Kod skryptu użyty do przekształcenia perspektywicznego obrazu (Rys.5. s.13)

```
clear all; clc;

%% wczytanie obrazu
% obraz 768px x 768px, odcienie szarosci (1 kanal)
obraz = imread('obraz.png');
rozmiar = 768;
figure(1); imshow(obraz);

       %u  %v  - wspolrzedne zrodlowe
Z = [   1   1;
      768   1;
      768 768;
        1 768];

       %x  %y  - wspolrzedne docelowe
D = [  50 100;
      750 150;
      700 500;
      200 750];

%% przeksztalcenie perspektywiczne
% obliczenie wspolczynnikow przeksztalcenia
% M - macierz przeksztalcenia
M = [Z ones(4,1) zeros(4,3)  -Z(:,1).*D(:,1) -Z(:,2).*D(:,1) ;
     zeros(4,3)  Z ones(4,1) -Z(:,1).*D(:,2) -Z(:,2).*D(:,2) ];
% rozwiazanie rownania: Mh=D (Ax=b) - metoda Gaussa
h = M\[D(:,1);D(:,2)];
h11=h(1); h12=h(2); h13=h(3); h21=h(4); h22=h(5); h23=h(6);
h31=h(7); h32=h(8); h33=1;

% przeksztalcenie perspektywiczne
obraz_przeksztalcony = imread('tlo.png')'; % ustawienie szarego tla
for u=1:rozmiar
    for v=1:rozmiar
       x = round( ( h11*u + h12*v + h13 ) / ( h31*u + h32*v + 1 ) );
       y = round( ( h21*u + h22*v + h23 ) / ( h31*u + h32*v + 1 ) );
       if x>=1 && x<=rozmiar && y>=1 && y<=rozmiar
           obraz_przeksztalcony(y,x) = obraz(v,u); % odwrocone wspolrzedne
       end
    end
end

imwrite(obraz_przeksztalcony,'obraz_perspektywa.png','PNG');
figure(2); imshow(obraz_przeksztalcony);
```

## 10.2. Płyta CD

Zawartość płyty CD dołączonej do pracy:

- Praca magisterska w formie elektronicznej (plik PDF)
- Skompilowana wersja programu *Wskaźnik Laserowy*
- Dodatkowe biblioteki konieczne do uruchomienia programu
- Kod źródłowy programu (projekt *MS Visual C++ 2005*)