Laboratorium Inteligentnych Maszyn i Systemów

Politechnika Warszawska
Wydział Elektryczny

PRACA PRZEJŚCIOWA

Temat: **Przetwarzanie obrazu i sterowanie manipulatorem Scorbot-er 4u**

Opiekun:
Dr inż. Witold Czajewski

Wykonali:
Paweł Golba **Nr albumu: 202617 sem.VIII Automatyka i Robotyka**
email: pawelgolba@gmail.com
Tomasz Rzepiński **Nr albumu: 202638 sem.VIII Automatyka i Robotyka**
email: rzepek86@vp.pl

Warszawa, 15.06.2009

Spis treści:

1 Wprowadzenie

Cele naszego projektu było wykorzystanie manipulatora z wizyjnym sprzężeniem zwrotnym do lokalizacji i manipulacji obiektami. W naszej pracy wykorzystaliśmy kamerę, którą sprzęgliśmy z manipulatorem przy wykorzystaniu języka wysokiego poziomu - C++. Zasadniczym celem było wykrycie i zlokalizowanie obiektów o różnych kształtach i kolorach. Dla uatrakcyjnienia działania układu zostało to zastosowane do gry: „kółko i krzyżyk."
Celami pobocznymi było zapoznanie się z podstawami przetwarzania obrazów oraz sterowaniem manipulatorem w praktyce.

2 Wybór rozwiązania

Do osiągnięcia celu postanowiliśmy zaimplementować, do działającego kodu gry kółko i krzyżyk, funkcje ruchu manipulatora, które będą odwzorowaniem położenia obiektów wykrytych poprzez system wizyjny.

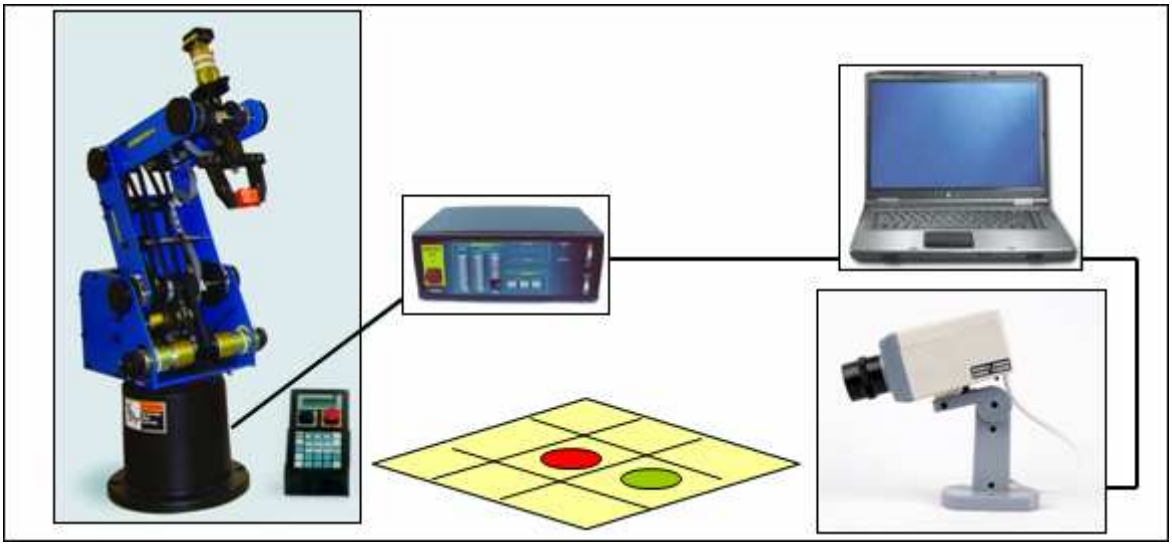
3 Założenia projektu

Nasz projekt realizowaliśmy korzystając z manipulatora Scorbot-er 4u, kamery FireWire, oraz stanowiska komputerowego. Ten sprzęt został dla nas wydzielony w laboratorium na czas realizacji projektu. Cały kod był pisany w języku wysokiego poziomu C++, ponieważ jest to język, za pośrednictwem, którego można sterować manipulatorem jaki mieliśmy do dyspozycji. Operacje związane z przetwarzaniem obrazów wspomagała biblioteka OpenCV. Środowisko, które użyliśmy w naszym projekcie to Microsoft Visual Studio.

Podzespoły układu sterowania

Nasz system obejmuje 4 komponenty sprzętowe:

- komputer
- kontroler USB
- manipulator Scorbot-er 4u
- kamera



Rys.1 schemat połączonego układu

Komputer połączony jest z Kontrolerem USB poprzez kabel USB. Manipulator natomiast łączy się z kontrolerem-USB poprzez specjalny opatentowany przewód dostarczony przez producenta. Kamera jest podłączona bezpośrednio do komputera przewodem. Całość ilustruje rys. 1.

Dla przejrzystości, powstały kod programu został podzielony i pogrupowany na 5 plików (wizja.cpp, scorbot.cpp, ttt.cpp, blobs.cpp, main.cpp).

W pierwszym zawarte są funkcje odpowiadające za obsłużenie wizji, w drugim funkcje sterujące robotem, w trzecim algorytm kółka i krzyżyk, w czwartym funkcje dodatkowej biblioteki do analizy blobów. W piątym znajduje się funkcja główna porządkująca cały program. Zawartość tych plików znajduje się w rozdziale DODATEK.

4 Realizacja projektu

4.1 Programowanie manipulatora z poziomu języka C++

Do tworzenia kodów źródłowych sterujących manipulatorem z poziomu języka C++ niezbędne są pliki konfiguracyjne oraz biblioteki udostępnione przez producenta.

Niezbędne do programowania manipulatora pliki to:

- biblioteka usbc.dll – powinna zostać umieszczona w folderze, w którym znajduje się kod programu bądź w folderze Windows.
- usbc.ini – powinien znajdować się razem z plikiem usbc.dll . Plik usbc.ini jest plikiem konfiguracyjnym przekazującym m.in. informacje o miejscu gdzie znajduje się plik konfiguracyjny er4conf.ini zawierający parametry manipulatora Scorbot-er 4u.
- folder PAR, w którym umieszczone są pliki konfiguracyjne poszczególnych osi manipulatora.

Do prawidłowej pracy kodu źródłowego z manipulatorem niezbędne są również pliki nagłówkowe, które należy dołączyć na etapie tworzenia programu w kodzie źródłowym, są to pliki:

- usbc.h, w którym znajdują się definicje dostępnych podczas programowania manipulatora funkcji.
- usbcdef.h, który przechowuje definicje stałych oraz struktur wykorzystywanych przez bibliotekę usbc.dll.
- extern.h, w którym znajduje się definicja klasy ErrorInfo.
- error.h, który zawiera definicję błędów.

Kolejnym niezbędnym plikiem podczas tworzenia programów w C++ jest plik usbc.lib, który powinien znajdować się w katalogu projektu.

4.2 Etapy nawiązywania połączenia między komputerem, a manipulatorem

Przed rozpoczęciem programowania właściwych ruchów konieczne jest nawiązanie połączenia z manipulatorem. Połączenie realizowane jest poprzez funkcję Initialization z odpowiednimi parametrami aktualnymi.

Opis parametrów formalnych funkcji Initialization:
bool Initialization(short sMode, short sSystemType, CallBackFun InitEnd, CallBackFun ErrMessage)

Parametr sMode odpowiada za ustawienie trybu pracy manipulatora. W miejsce parametru można wstawić następujące wartości:
INIT_MODE_DEFAULT=0 – parametr odwołuje się do pliku konfiguracyjnego i powoduje włączenie ostatnio używanego trybu pracy z manipulatorem
INIT_MODE_ONLINE=1 – parametr ten nawiązuje połączenie online z manipulatorem.

Parametr sSystemType odpowiada za nawiązanie połączenia z określonym typem manipulatora. W miejsce parametru można wstawić następujące wartości:
DEFAULT_SYSTEM_TYPE=0 – parametr ten odpowiada za autodetekcję, pozwala automatycznie wykryć podłączony rodzaj manipulatora.
ER4USB_SYSTEM_TYPE=41 – parametr ten odpowiada za nawiązanie połączenia z manipulatorem Scorbot-er 4u

Ostatnie dwa parametry są funkcjami o parametrach określonych przez wskaźnik do funkcji CallBackFun. Typ CallBackFun jest wskaźnikiem pokazującym na funkcję, której deklaracja wygląda następująco: void nazwa_funkcji (void*).

Funkcja InitEnd jest wywoływana jeżeli inicjalizacja z urządzeniem przebiegła pomyślnie. Przeciwieństwem funkcji InitEnd jest funkcja będąca jednocześnie ostatnim parametrem inicjalizacji ErrMessage. Funkcja ta jest realizowana jeżeli przy połączeniu z manipulatorem wystąpi błąd.

Przykład

```
//W przykładzie wykorzystano rzutowanie wskaźnika typu ConfigData oraz ErrorInfo na
typ //void*. Niezbędne jest wcześniejsze dołączenie do kodu bibliotek wcześniej opisanych.

void InitEnd(ConfigData *pTheConfigData)
{
printf("Inicjalizacja...");
}
void ErrorMessage(ErrorInfo *pTheErrorInfo)
{
printf("Blad");
printf("\nError: %d ",pTheErrorInfo->lNumber);
printf("%s",pTheErrorInfo->cOptional);
error=1;
}
int main(void)
{
Initialization(INIT_MODE_ONLINE,ER4USB_SYSTEM_TYPE,(CallBackFun)&InitEnd,
(CallBackFun)&ErrorMessage);
}
```

Kolejnym zalecanym krokiem jest włączenie kontroli ruchu dla poszczególnych osi manipulatora. Bez wykonania tej czynności funkcje ruchu nie będą wykonywane. Kontrola włączana bądź wyłączana jest za pomocą funkcji Control z odpowiednimi parametrami aktualnymi. Parametr BisOn ustawiony jako TRUE włącza kontrolę poszczególnych osi, natomiast ustawiony jako FALSE wyłącza sterowanie określonymi wcześniej osiami.

Opis parametrów formalnych funkcji Control:

bool Control(unsigned char ucGroupAxis, bool BisOn)

Parametr ucGroupAxis odpowiada za wybranie określonych osi, które następnie zostaną włączone bądź wyłączone. Parametr ucGroupAxis może przyjmować następujące wartości:
‘A’ – włącza sterowanie wszystkimi osiami manipulatora (bez osi urządzeń peryferyjnych – 7 i 8).
‘&’ – włącza sterowanie wszystkimi osiami manipulatora, włącznie z urządzeniami peryferyjnymi.
‘B’ – włącza sterowanie osiami odpowiadającymi za urządzenia peryferyjne (oś 7 oraz 8).

Przykład

```
//Włączenie sterowania wszystkich osi.
Control('&',TRUE)
```

Pomimo nawiązania połączenia z manipulatorem oraz zezwolenia na określanie ruchów tylko niektóre funkcji ruchowe będą wykonywane. Do prawidłowego poruszania manipulatorem konieczne jest również bazowanie osi. Manipulator po bazowaniu jest w stanie określić swoją aktualną pozycję zarówno w zmiennych zewnętrznych jak i wewnętrznych.

Za ustawienie manipulatora w pozycji bazowej odpowiada funkcja Home wraz z odpowiednimi parametrami aktualnymi.

Opis parametrów formalnych funkcji Home:
bool Home(unsigned char ucGroupAxis,CallBackFun fnHomingNotif)
Pierwszy parametr (ucGroupAxis) odpowiada za wybór osi, które mają zostać ustawione w pozycji bazowej. Parametr ten może przyjmować następujące wartości:
‘A’ – bazowaniu zostaną poddane wszystkie osie manipulatora
‘B’ – bazowaniu zostaną poddane osie przypisane urządzeniom peryferyjnym (7 oraz 8)
‘0’-‘7’ – bazowaniu zostaną poddane pojedyncze osie.

Należy zauważyć, iż w języku C++ pojedyncze osie numerowane są począwszy od 0.

Funkcja fnHomigNotif wywoływana jest przy rozpoczyniu bazowania i konczenia każdej osi. Parametr ten nie jest niezbędny do prawidłowego funkcjonowania procesu bazowania, może więc zostać ustawiony jako NULL.

Poniżej przedstawione są przykłady właściwego bazowania manipulatora Scorbot-er 4u. W przypadku bazowania manipulatora w oparciu o pojedyncze osie, należy określić odpowiednią kolejność, ponieważ niektóre osie są ze sobą sprzęgnięte. Prawidłowa kolejność bazowania osi jest następująca: 1,2,4,3,0,5. Nieuwzględnie prawidłowej kolejności przy bazowaniu pojedynczymi osiami powoduje niewłaściwe ustawienia manipulatora.

Przykład

```
//Bazowanie osi manipulatora
Home('A',NULL)
//Bazowanie manipulatora przy wykorzystaniu poszczególnych osi ‘0’-‘5’
int bazowanie[]={1,2,4,3,0,5};
for (int i=0; i<=5; i++)
{
Home(bazowanie[i],NULL);
}
```

4.3 Funkcje ruchu manipulatora w języku C++

Wykorzystując funkcję MoveManual istnieje możliwość poruszania manipulatora w zmiennych zewnętrznych oraz wewnętrznych. Przed jej wywołaniem konieczne jest jednak określenie typu układu, w którym ruch będzie wykonywany. Odpowiada za to funkcja EnterManual.

Opis parametrów formalnych funkcji EnterManual:
bool EnterManual(short sManualType)

Parametr sManual
Type może przyjmować następujące wartości:

MANUAL_TYPE_ENC=0, odpowiada za sterowanie manipulatora w zmiennych wewnętrznych.
MANUAL_TYPE_XYZ=1, odpowiada za sterowanie manipulatora w zmiennych zewnętrznych.
W celu wykonania ruchu przy wykorzystaniu funkcji: MoveLinear, MoveJoint oraz MoveCircularVect należy zdefiniować wektor przy użyciu funkcji DefineVector, w który umieszczone będą pozycje manipulatora. Pozycje dodawane są do wektora utworzonego przy pomocy funkcji DefineVector przy użyciu funkcji Teach z odpowiednimi parametrami.

Opis parametrów formalnych funkcji DefineVector:
bool DefineVector(unsigned char ucGroup, char *VectorName, short Dimension)

Parametr ucGroup może przyjmować następujące wartości:
‘A’ – jeżeli zapamiętywane pozycje będą dotyczyć jedynie manipulatora
‘B’ – jeżeli zapamiętywane pozycje będą dotyczyć urządzeń peryferyjnych
‘&’ – jeżeli zapamiętywane pozycje będą dotyczyć wszystkich osi (manipulatora i urządzeń peryferyjnych)

Parametr VectorName jest nazwa wektora, który będzie przechowywał określone pozycje manipulatora, natomiast parametr Dimension odpowiada za ilość przechowywanych pozycji.
Opis parametrów formalnych funkcji Teach:
bool Teach (char *VectorName, short PointNumber, long *plCoorArray, short splCoorArray, long lPointType).

Parametr VectorName jest nazwą wcześniej zdefiniowanego wektora, PointNumber odpowiada za numer pozycji w danych wektorze, której zostaną przydzielone współrzędne punku określone przez tablicę plCoorArray, splCoorArray jest rozmiarem tablicy plCoorArray, a lPointType może przyjmować wartości:

ABS_XYZ_A – zapamiętanie pozycji zmiennych zewnętrznych względem układu bazowego
ABS_JOINT – zapmiętanie pozycji zmiennych wewnętrznych względem układu bazowego

W języku C++ istnieje możliwość wykonywania ruchów z interpolacją: quasiliniową (MoveJoint), liniową (MoveLinear) oraz kołową (MoveCircuitVect).
Opis parametrów formalnych funkcji MoveJoint:
bool MoveJoint(char *VectorName, short PointNumber)
Parametr VectorName jest nazwą wektora przechowującego numer pozycji(PointNumber) do której manipulator się przemieści.

Opis parametrów formalnych funkcji MoveLinear:
bool MoveLinear(char *VectorName, short PointNumber)
Parametr VectorName jest nazwą wektora przechowującego numer pozycji(PointNumber) do której manipulator się przemieści z interpolacją linii prostej.

Opis parametrów formalnych funkcji MoveLinear:

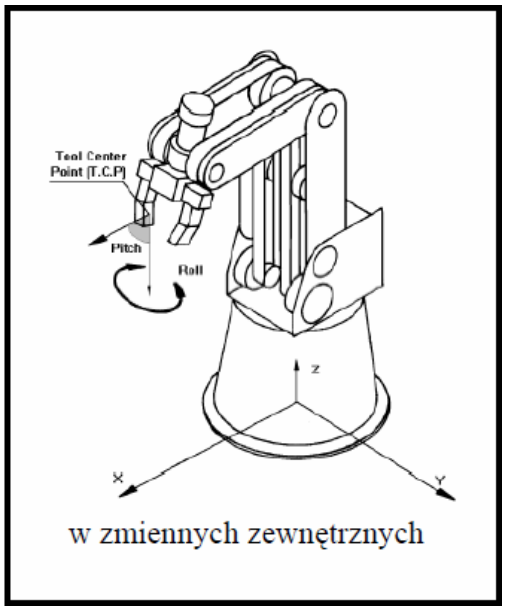
bool MoveCircularVect(char *VectorName, short ThroughtPointNumber, short toPointNumber)

Parametr VectorName jest nazwą wektora przechowującego numer pozycji (ThroughtPointNumber) przez którą manipulator ruchem z interpolacją kołową przemieści się do pozycje określonej przez parametr toPointNumber. Pozycje powinny znajdować się w wektorze o tej samej nazwie.

Przykład

```
//przykład dotyczy pozycjonowania quasiliniowego, reszta jest analogiczna
DefineVector('A',"pozycje",2);
long *plCoorArray=new long[5];
plCoorArray[0]=169000;
plCoorArray[1]=0;
plCoorArray[2]=500000;
plCoorArray[3]=-63000;
plCoorArray[4]=0;
Teach("pozycje",2,plCoorArray,5,ABS_XYZ_A);
MoveJoint("pozycje",1);
```

Za otwieranie oraz zamykanie chwytaka odpowiedzialne są bezparametrowe funkcje OpenGripper() oraz CloseGripper(). Istnieje także możliwość określania odstępu pomiędzy szczękami chwytaka.



Rys.1 kierunki osi wybazowanego manipulatora

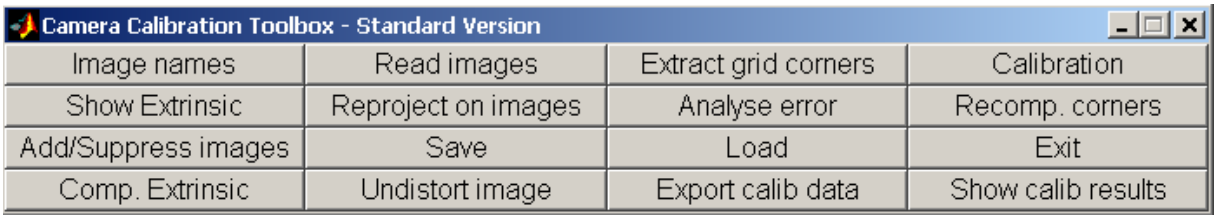
W naszym projekcie korzystaliśmy z możliwości sterowania danym manipulatorem w jego zmiennych zewnętrznych względem układu bazowego skojarzonego z jego nieruchomą podstawą (rys.2).

4.4 Kalibracja kamery

Kalibrację przeprowadziliśmy za pomocą programu Matlab oraz specjalnego pakietu narzędziowego Camera Calibration Toolbox for Matlab. Pakiet ten należy skopiować do katalogu toolbox.

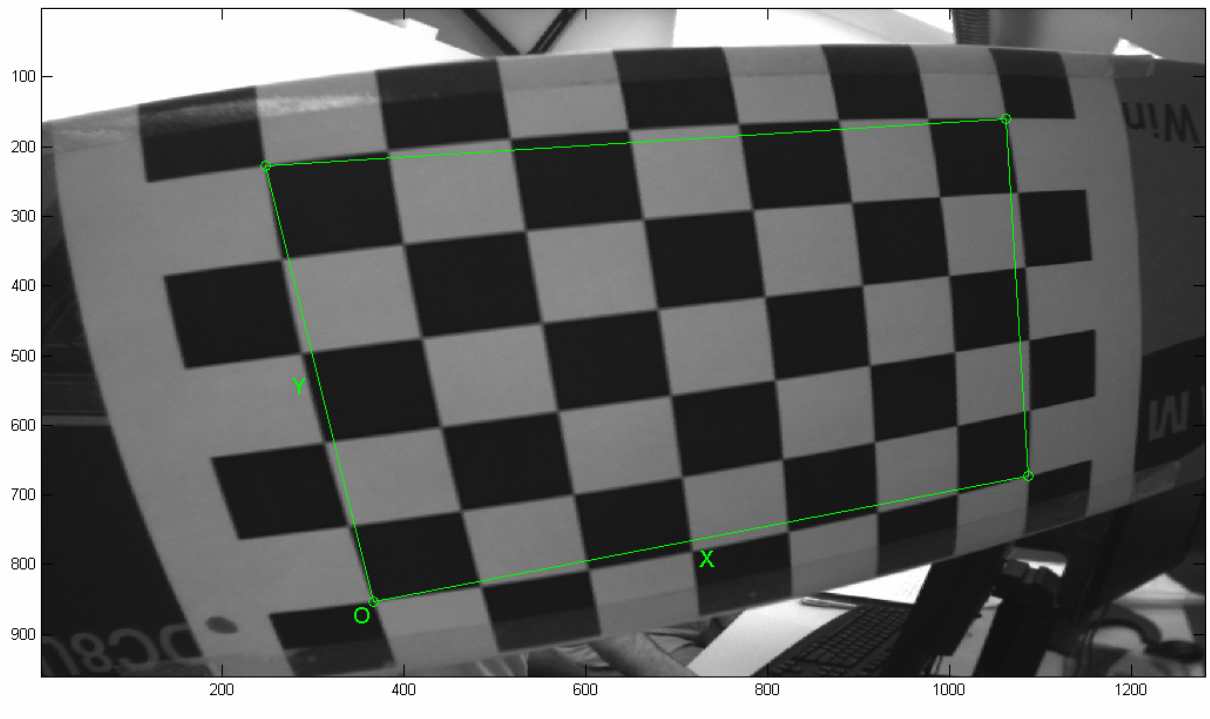
Do kalibracji niezbędne jest najpierw wykonanie jak największej liczby zdjęć szachownicy pod różnymi kątami względem kamery oraz z różnej odległości i w różnych częściach pola widzenia kamery.

Możemy teraz rozpocząć kalibrację kamery wywołując funkcję Calib.
Pokaże się menu, z wyborem funkcji (rys. 3):



Rys.3 menu wywołane przez funkcję Calib

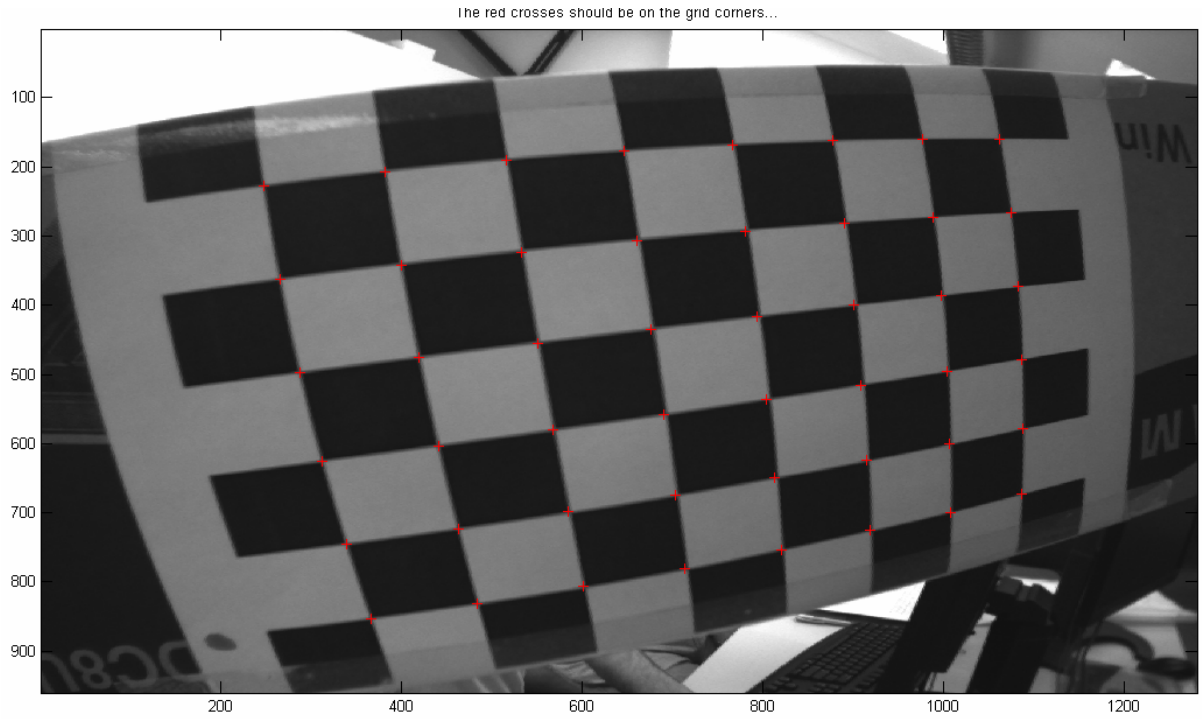
Nas będzie interesować funkcja Image names oraz Extract grid corners. Pierwsza z nich wczyta zrobione wcześniej zdjęcia.
Ta druga jest częścią zasadniczą kalibracji. Po jej wybraniu po kolei będą pojawiać się wczytane zdjęcia szachownicy. Naszym zadaniem będzie wybranie czterech narożników tak jak jest to pokazane na rys.4:



Rys.4 przebieg kalibracji – zaznaczanie narożników szachownicy

oraz podanie w linii komend: liczby kwadratów wzdłuż osi x oraz osi y, rzeczywisty rozmiar pojedynczego kwadratu wzdłuż osi x i y.

Matlab zaproponuje położenie pozostałych narożników rys.5:



Rys.5 proponowane przez Matlab położenia pozostałych narożników

Czynności te należy powtórzyć dla wszystkich wczytanych zdjęć.

Następnie wybieramy funkcję Calibration i automatycznie otrzymamy parametry kamery.

Np.:

Calibration parameters after initialization:

Focal Length:	fc = [869.54265 869.54265]
Principal point:	cc = [639.50000 479.50000]
Skew:	alpha_c = [0.00000] => angle of pixel = 90.00000 degrees
Distortion:	kc = [0.00000 0.00000 0.00000 0.00000 0.00000]

Parametry oznaczają po kolei:

Ogniskową, punkt środkowy, współczynnik krzywizny oraz współczynnik zniekształcenia obrazu (dystorsji).

Otrzymane dane można już wprowadzić do dwóch plików xml (`"intrinsic_duza.xml"` oraz `"distortion_duza.xml"`), a następnie wczytać do naszego projektu.

Parametry te posłużyły nam do usunięcia zniekształceń obrazu.

Następnie mając 2 pliki xml ze współczynnikami zniekształceń:
- intrinsic_duza.xml:

```
%YAML:1.0
intrinsic: !!opencv-matrix
   rows: 3
   cols: 3
   dt: f
   data: [ 1045.77069907845, 0, 640,
           0, 1041.8322312451442, 480,
           0, 0, 1]
```

- distortion_duza.xml:

```
%YAML:1.0
distortion: !!opencv-matrix
   rows: 1
   cols: 4
   dt: f
   data: [ -0.353375603162531, 0.145509746061277, -0.001095098768833,
0.000349118520763]
```

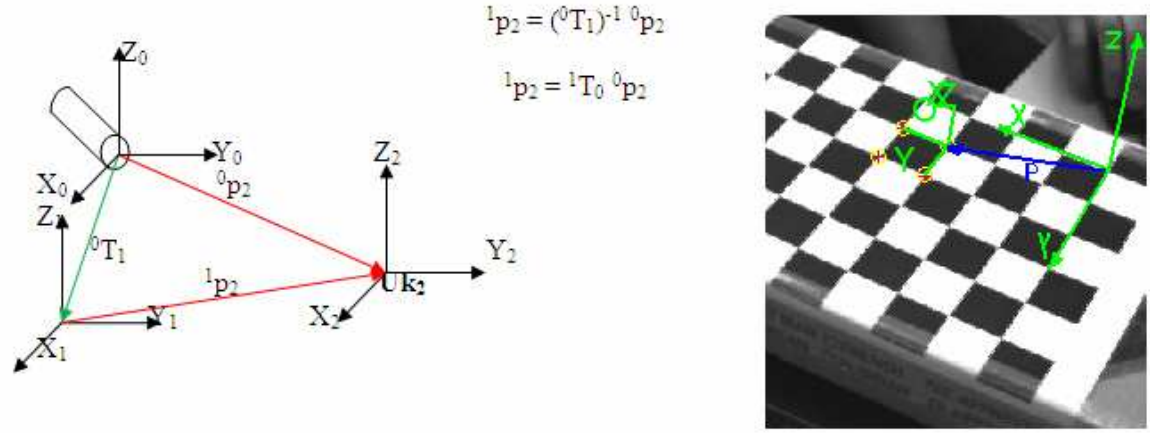
możemy poprawić obraz z kamery. Posłuży do tego funkcja: popraw_zdjecie(IplImage *src)

```
void popraw_zdjecie(IplImage *src){



	cvNamedWindow( "Oryginalny", 1 );
	cvShowImage( "Oryginalny", src ); // Show raw image

	CvMat* intrinsic = (CvMat*)cvLoad("intrinsic_duza.xml");//pobranie
	parametrow kamery
	CvMat* distortion = (CvMat*)cvLoad("distortion_duza.xml");

	//wyznaczenie map przeksztalcen korygujacych obraz kamery - raz dla
	danej kamery!
	IplImage* mapx = cvCreateImage( cvGetSize(src), IPL_DEPTH_32F, 1 );
	IplImage* mapy = cvCreateImage( cvGetSize(src), IPL_DEPTH_32F, 1 );
	cvInitUndistortMap(intrinsic,distortion,mapx,mapy);

	//usuniecie zniekształcen
	IplImage *t = cvCloneImage(src);
	cvRemap( t, src, mapx, mapy, CV_INTER_CUBIC | CV_WARP_FILL_OUTLIERS
	);
	cvReleaseImage(&t);
	cvReleaseImage(&mapx);
	cvReleaseImage(&mapy);
}
```

Kalibracja była przeprowadzana także za pomocą funkcji z OpenCV. W tym przypadku kalibracja była w znacznym stopniu łatwiejsza, ponieważ zaznaczanie narożników szachownicy wykonywał za nas gotowy algorytm. Otrzymane parametry układu optycznego z tej metody były zbliżone do otrzymanych z Matlaba.

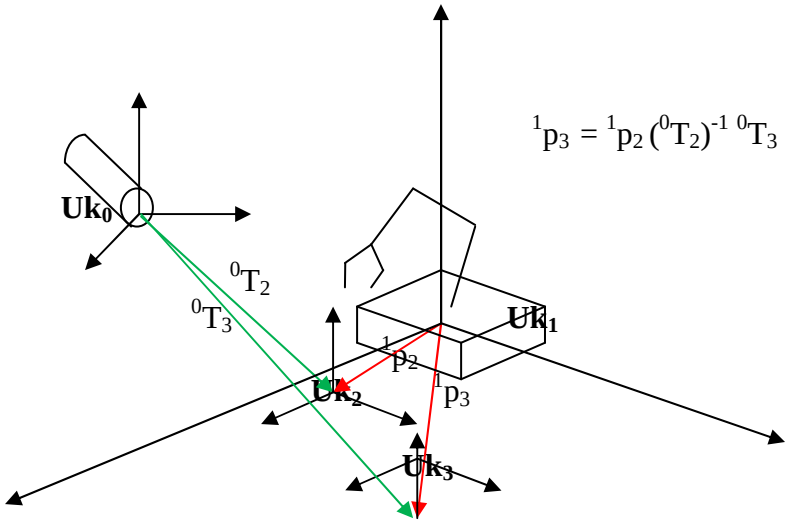
4.5 Transformacje układów współrzędnych

Kamera została umieszczona z boku, tak że obserwujemy planszę z pewnej perspektywy. Z kamerą związany jest układ 0, z robotem układ 1, natomiast z szachownicą układ 2. Wektor ${}^0p_2$ opisuje transformację układu kamery do układu szachownicy.
Matlab zwraca nam wektor ${}^0p_2$ (rys.6). Jeżeli chcemy uzyskać współrzędne punktu bazowego układu 2 (Uk_2) względem układu bazowego szachownicy (czyli uzyskać wektor ${}^1p_2$) należy przemnożyć odwrotność macierzy transformacji z układu 0 do 1 przez wektor ${}^0p_2$.



Rys.6 wykorzystanie transformacji układów współrzędnych

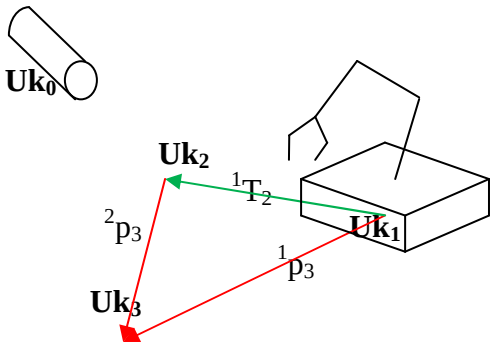
Mając współrzędne jednorodne początku układu współrzędnych związanego z płaskim obiektem oraz znając jego wymiary możemy zredukować zadanie lokalizacji obiektów na tymże obiekcie do zadania dwuwymiarowego. Pomocne tu będzie przekształcenie perspektywiczne (zwane przekształceniem homograficznym) obserwowanego prostokąta w prostokąt o zadanych wymiarach odpowiadających rzeczywistym wymiarom obiektu.



Rys.7 warunki przejścia do układu homograficznego

Korzystając z przekształcenia homograficznego algorytm się znacznie upraszcza (rys.8). Program komputerowy zwraca wartość położenia punktu na szachownicy (Uk_3) względem punktu bazowego szachownicy (Uk_2), a do tego mając macierz transformacji układu szachownicy względem układu bazowego robota (Uk_1), możemy łatwo znaleźć wektor przesunięcia punktu na szachownicy względem układu bazowego robota.

$$^1p_3 = {}^1T_2\ {}^2p_3$$



Rys.8 układ homograficzny

Tak wygląda fragment kodu, który wyznacza macierz przekształcenia homograficznego H i zapisuje do pliku H2.xml:

```
int found=0;
CvPoint2D32f objPts[4], imgPts[4];
float k_x=7*30, k_y=5*30; //rzeczywisty rozmiar prostokata
      objPts[0].x = 0; objPts[0].y = 0;
      objPts[1].x = k_x; objPts[1].y = 0;
      objPts[2].x = k_x; objPts[2].y = k_y;
      objPts[3].x = 0; objPts[3].y = k_y;

      H = cvCreateMat( 3, 3, CV_32F);
      p_img = cvCreateMat( 3, 1, CV_32F);
      p_obj = cvCreateMat( 3, 1, CV_32F);
while(found==0){
      flycaptureGrabImage2( context, &imagex );
      flycaptureConvertImage( context, &imagex, &imageConverted );
      popraw_zdjecie(image);
```

```
      cvNamedWindow( "Poprawiony", 1 );

//pobranie wspolrzednych 4 naroznikow planszy
      int board_w = 8;
      int board_h = 6;
      int corner_count;
      int board_n = board_w * board_h;
      CvSize board_sz = cvSize( board_w, board_h );
      CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
      IplImage *fota = cvCloneImage(image);
      IplImage *gray_image = cvCreateImage(cvGetSize(fota),8,1);//subpixel
      cvCvtColor(fota, gray_image, CV_BGR2GRAY);

      //Find chessboard corners:
      int found = cvFindChessboardCorners( image, board_sz, corners,
      &corner_count, CV_CALIB_CB_ADAPTIVE_THRESH |
      CV_CALIB_CB_NORMALIZE_IMAGE  );

      //Get Subpixel accuracy on those corners
      cvFindCornerSubPix(gray_image, corners, corner_count,
      cvSize(11,11),cvSize(-1,-1), cvTermCriteria(
      CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));

      imgPts[0].x = corners[40].x; imgPts[0].y = corners[40].y;
      imgPts[1].x = corners[47].x; imgPts[1].y = corners[47].y;
      imgPts[2].x = corners[7].x; imgPts[2].y = corners[7].y;
      imgPts[3].x = corners[0].x; imgPts[3].y = corners[0].y;
      printf("\n%f",corners[0].x);
      //Draw it
      cvDrawChessboardCorners(fota, board_sz, corners, corner_count,
      found);

      cvNamedWindow("narozniki",1);
      cvShowImage( "narozniki", fota );

      cvReleaseImage(&fota);
      cvReleaseImage(&gray_image);
//wyznaczenie przeksztalcenia przeliczajacego wsp. obrazu na wsp. obiektu
      cvGetPerspectiveTransform( imgPts,objPts, H);
      cvSave("H2.xml",H);
```

Program działa w pętli while, aż do wykrycia narożników (funkcja cvFindChessboardCorners zwróci wtedy wartość 1). Szachownicę ustawiliśmy tak, by poszczególne osie układu bazowego robota i układu związanego z początkiem szachownicy były do siebie równoległe. Dzięki takiemu ustawieniu macierz ${}^{1}T_{2}$ wiążąca te 2 układy będzie jedynie macierzą translacyjną, macierz rotacji będzie bowiem jednostkową.

$$ {}^{1}T_{2}= \begin{bmatrix} 1 & 0 & 0 & 301.31 \\ 0 & 1 & 0 & 25.48 \\ 0 & 0 & 1 & 109.75 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Samo obliczenie macierzy H, sprowadza się do następujących operacji:

- Wyznaczenie współrzędnych pikselowych czterech narożników szachownicy – w naszym programie zostaną one wpisane do struktury CvPoint2D32f imgPts[4].
- Wyznaczenie współrzędnych rzeczywistych tejże szachownicy, czyli jej rozmiarów – w naszym programie zostaną one wpisane do struktury CvPoint2D32f objPts[4].
- Wyznaczenie macierzy H, na podstawie danych z imgPts[] i objPts[] i funkcji cvGetPerspectiveTransform(imgPts,objPts, H);

4.6 Segmentacja obrazów kolorowych z wykorzystaniem biblioteki OpenCV

Segmentacja obrazów kolorowych z wykorzystaniem biblioteki OpenCV to metoda, która posłużyła do wydzielenia z analizowanego obrazu obiektów o różnych kolorach. Obiekty, które były konieczne do wydzielenia to pionki do gry w kółko i krzyżyk (czerwone i zielone) oraz pola na planszy.

Kolejne kroki do uzyskania oczekiwanych obiektów:

1.Wczytanie obrazu.

2. Wyznaczenie obszarów o interesującej nas jasności, barwie i nasyceniu (i odrzucenie wszystkiego innego) - utworzenie obrazu binarnego.

3. Znalezienie obszarów połączonych (blobów) i zaznaczenie ich na ekranie.

4. Analiza parametrów znalezionych blobów i odrzucenie blobów niepasujących do zadanych kryteriów.

5. Zaznaczenia na ekranie poszukiwanych obiektów i wyświetlenie ich wybranych parametrów.

Kod, realizujący powyższe czynności kolejno dla 3 kolorów:

```
for ( int i=0; i<3; i++ )
      {
            maske_wloz(image,maska,i);//tworzy obraz binarny(maska) na
kolory RGB

            wykryj_bloby(maska, image, i);//analizuje bloby i zaznacza na
obrazie oryginalnym
            wypisz_wspolrzedne(i);  //wypisuje wspolrzedne srodka ciezkosci
pikselowe i rzeczywiste poszczegolnych blobow

      }
```

Rozpoznawania kolorów jest znacznie łatwiejsze w przestrzeni HSV, której składowe oznaczają kolejno: barwę, nasycenie i jasność, niż w przestrzeni nominalnej RGB. W przestrzeni HSV np. kolor czerwony, niezależnie od nasycenia i jasności, będzie miał wartości H w okolicy 180 (w implementacji OpenCV), podczas gdy w przestrzeni RGB ten sam kolor (w zależności od jasności i nasycenia) będzie przyjmował najróżniejsze wartości wszystkich składowych.

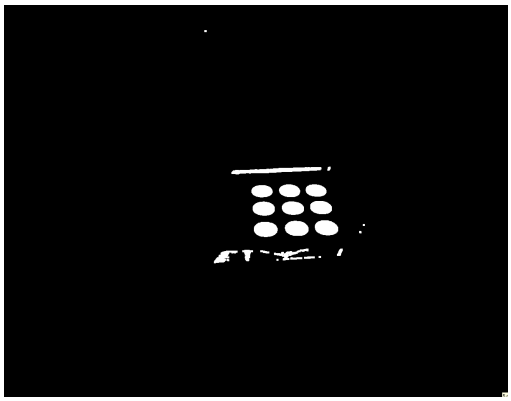
Tak wygląda funkcja, która założy maskę (utworzy obraz binarny: mask) na obraz oryginalny img, wydzielając odpowiedni kolor (0 – czerwony, 1 – żółty, 2 – zielony).

```
int maske_wloz (IplImage *img, IplImage *mask, int color )
{

      IplImage* hsv=cvCreateImage( cvGetSize(img), 8, 3 ); //obrazek dla
przestrzeni HSV
      cvCvtColor(img,hsv,CV_BGR2HSV); //konwersja na HSV - z tego bedzie
robiona maska kolorow

      //szybki sposob dostepu do pikseli obrazu
      uchar* data = (uchar *)hsv->imageData;
      uchar* dataMASK = (uchar *)mask->imageData;
      int step = hsv->widthStep;
      int stepMASK = mask->widthStep;

      for (int i = 0; i < hsv->height; i++)
            for (int j = 0; j < hsv->width; j++) {
                  int h=data[i*step+j*3+0];
                  int s=data[i*step+j*3+1];
                  int v=data[i*step+j*3+2];

      //utworzenie maski binarnej dla danego koloru
                  if(color==0)
                        if(v>=70 && (h>170 || h<6) && s>120) //czerwony
                              dataMASK[i*stepMASK+j] = 255;
                        else
                              dataMASK[i*stepMASK+j] = 0;
```

```
                else if(color==2)
                      if(v>=20 && h>60 &&  h<110 && s>30) //zielony
                            dataMASK[i*stepMASK+j] = 255;
                      else
                            dataMASK[i*stepMASK+j] = 0;


                else if(color==1)
                      if(v>=120 && (h>10 &&  h<25) && s>60) //zolty
                            dataMASK[i*stepMASK+j] = 255;
                      else
                            dataMASK[i*stepMASK+j] = 0;

          }
```

Na początku dokonujemy konwersji obrazu oryginalnego na obraz w przestrzeni HSV. Kolejne linie programu odpowiadają za pobieranie danych obrazka, czyli poszczególnych składowych H, S oraz V każdego piksela obrazu. Następnie instrukcja warunkowa sprawdza dane poszczególnego piksela czy znajdują się w przedziale wartości ustawionym przez nas. Jeżeli tak, wówczas piksel maski przyjmuje wartość 255 czyli barwę białą. Reszta obszaru zdjęcia zaś jest ustawiana na 0, czyli staje się czarna.

Np.: dla koloru żółtego:



Rys.9 segmentacja obrazu

Skoro mamy już obraz binarny, w którym obecne są tylko i wyłącznie interesujące nas piksele, chcemy teraz wykryć czy owe piksele łączą się w jakieś większe obszary i wyznaczyć ich pewne własności geometryczne, abyśmy mogli je od siebie odróżnić.

Jednym ze sposobów na wykrycie takich obszarów, zwanych często blobami (z ang. blob=Binary Large Object), jest wykorzystanie funkcji BlobAnalysis (z zewnętrznej biblioteki, tej funkcji NIE MA w OpenCV). Aby było to możliwe dodaliśmy do naszego projektu pliki zawierające funkcje związane z wykrywaniem blobów (blobs.cpp oraz blobs.h).

Tak wygląda funkcja segmentująca bloby na obrazie binarnym (w tym przypadku parametr img) oraz zaznaczająca na obrazie oryginalnym wykryte obiekty:

```
int wykryj_bloby(IplImage *img, IplImage * oryginalny, int color )
{

      blobs.BlobAnalysis(img, 0, 0, 1280, 960, 0, 0);
      blobs.BlobInclude(BLOBCIRCULARITY,0.8,1.10);
      blobs.BlobExclude(BLOBAREA,0,450);

if (color==0){
      printf("\nZnaleziono %d blobow czerwonych\n",blobs.BlobCount);
}
else if (color==2){
      printf("\nZnaleziono %d blobow zielonych\n",blobs.BlobCount);
      liczba_zielonych = blobs.BlobCount;
}
else if (color==1){
      printf("\nZnaleziono %d blobow zoltych\n",blobs.BlobCount);
}

//zaznaczenie wykrytych blobow

for (int i=1;i<=blobs.BlobCount;i++){

cvRectangle(oryginalny,cvPoint(blobs.RegionData[i][BLOBMINX],blobs.RegionDa
ta[i][BLOBMINY]),cvPoint(blobs.RegionData[i][BLOBMAXX],blobs.RegionData[i][
BLOBMAXY]),CV_RGB(0,0,255));

cvEllipse(oryginalny,cvPoint(blobs.RegionData[i][BLOBSUMX],blobs.RegionData
[i][BLOBSUMY]),cvSize(5,5),360, 0, 360, CV_RGB(255,255,255));

    }
return 0;
}
```

Parametrami metody BlobAnalysis są kolejno: współrzędne początku i rozmiar obszaru poddawanego analizie (najczęściej będzie to po prostu cały ekran), następnie kolor ramki wokół obrazu oraz rozmiar (w pikselach) poniżej którego bloby nie będą wykrywane (np. mogą nas nie interesować pojedyncze piksele będące szumem).

Warto tu przybliżyć **wybrane** parametry blobów jakie są wyznaczane przez metodę BlobAnalysis. Są to:

- BLOBPARENT - nr bloba nadrzędnego, czyli tego w środku którego znajduje się aktualny blob
- BLOBCOLOR - kolor bloba (w zasadzie wartość piksela z obrazu binarnego) - 1 oznacza biały, 0 czarny
- BLOBAREA - pole powierzchni bloba (liczba pikseli stanowiących bloba)
- BLOBPERIMETER - obwód bloba (liczba pikseli krawędziowych bloba)
- BLOBSUMX - współrzędna X środka ciężkości bloba
- BLOBSUMY - współrzędna Y środka ciężkości bloba
- BLOBMINX - najmniejsza współrzędna X bloba
- BLOBMAXX - największa współrzędna X bloba
- BLOBMINY - najmniejsza współrzędna Y bloba
- BLOBMAXY - największa współrzędna Y bloba
- BLOBMAJORAXIS - długość większej osi głównej
- BLOBMINORAXIS - długość mniejszej osi głównej
- BLOBORIENTATION - kąt nachylenia większej osi głównej względem osi X
- BLOBECCENTRICITY - odchylenie od kształtu koła
- BLOBCOMPACTNESS - zawartość obszaru rozumiana jako stosunek jego powierzchni do powierzchni prostokąta otaczającego
- BLOBCIRCULARITY - inna miara podobieństwa do koła

Parametry te można wykorzystać do określenia właściwości geometrycznych obserwowanych obszarów połączonych, a następnie do ich sklasyfikowania (np. do wybrania kół czy kwadratów).

Naszym celem jest wyłącznie owali. W tym celu wykorzystaliśmy jedną z miar kolistości i pozostawiliśmy bloby mieszczące się w zadanym przedziale liczbowym. Jaki to przedział dla danego parametru? Najlepiej sprawdzić to eksperymentalnie z różnymi obiektami, a dopiero potem zastosować go na docelowym obrazie. Metodę BlobInclude, wykorzystaliśmy do pozostawienia do dalszego przetwarzania tylko owali. Do odrzucenia niepasujących obszarów (pod względem jednej z ich własności) wykorzystaliśmy metodę BlobExclude – do usunięcia blobów o małym obszarze.

Kolejny krok to zwrócenie współrzędnych. Najpierw w pikselach, następnie rzeczywistych (obliczonych na podstawie macierzy H), oraz współrzędnych względem bazy robota (po dodaniu wektora translacji).

Fragment funkcji wykonującej te czynności (dla obiektów czerwonych):

```
void wypisz_wspolrzedne(int color)
{
      for (int i=1; i<= blobs.BlobCount; i++)
            {
            float xx=blobs.RegionData[i][BLOBSUMX];
            float yy=blobs.RegionData[i][BLOBSUMY];

            CV_MAT_ELEM(*p_img, float,0,0) = (float)xx;
            CV_MAT_ELEM(*p_img, float,1,0) = (float)yy;
            CV_MAT_ELEM(*p_img, float,2,0) = 1.0;
            cvMatMul(H,p_img,p_obj);

if (color==0){
printf("\n");
printf("wsp. ekranowe obiektu czerwonego nr.%d: %.3fx%.3f",i,xx,yy);

printf("\nwsp. na obiekcie: %.3fx%.3f",CV_MAT_ELEM(*p_obj,
float,0,0)/CV_MAT_ELEM(*p_obj, float,2,0),CV_MAT_ELEM(*p_obj,
float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0));

printf("\nwsp. dla robota: %.3fx%.3f",CV_MAT_ELEM(*p_obj,
float,0,0)/CV_MAT_ELEM(*p_obj, float,2,0)+301.31,CV_MAT_ELEM(*p_obj,
float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0)+25.48);
printf("\n");

XYZ_red[i].x=CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+301.31;

XYZ_red[i].y=CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+25.48;

                  }

}
```

Struktury XYZ_red[i] zapamiętują współrzędne dla robota. Posłużą one do zaprogramowania ruchów robota.

4.7 Implementacja algorytmu gry: „kółko i krzyżyk.”

Ta część pracy polegała jedynie na znalezieniu gotowego algorytmu, zrozumieniu go, a następnie odpowiednim wykorzystaniu.

Wywołanie tego programu znajduję się w funkcji main():

```
//zagraj w kółko i krzyżyk

       while(1)
    {
      init_board();
      if (user_first())
      {
        computer = 'O';
        user = 'X';
      }
      else
      {
        computer = 'X';
        user = 'O';
      }
      play_game();
      if (!play_again())
      break;
    }
```

Pozostała część znajduje się w pliku ttt.cpp.

Szereg funkcji znajdujących się w tym programiku doprowadza do zasadniczej czynności: wyboru odpowiedniego pola tablicy char board [3] [3] jak wiadomo od 1 do 9.

W przypadku przeciwnika, czyli komputera należało zatem dopisać funkcję, która wywoła nam program do wykonywania ruchów robota.

Tak wygląda jej najważniejszy fragment:

```
int ruch1(int square){
      .
      .
      .
      long* plCoorArray1 = new long[5];//pozycja nad pionkiem czerwonym
      plCoorArray1[0]=XYZ_red[zmienna].x*1000;
```

```
        plCoorArray1[1]=XYZ_red[zmienna].y*1000;
        plCoorArray1[2]=109750;
        plCoorArray1[3]=-85700;
        plCoorArray1[4]=4730;

        Teach("wektorVS",2,plCoorArray1,5,ABS_XYZ_A);

        .
        .
        .
        long* plCoorArray3 = new long[5];
        plCoorArray3[0]=XYZ_yellow[square].x*1000;//pozycja nad polem zoltym
        plCoorArray3[1]=XYZ_yellow[square].y*1000;
        plCoorArray3[2]=109750;
        plCoorArray3[3]=-85700;
        plCoorArray3[4]=4730;

        Teach("wektorVS",4,plCoorArray3,5,ABS_XYZ_A);
        .
        .
        .
}
```

Współrzędne poszczególnych czerwonych pionków (takich będzie używał robot), są pobrane z funkcji wypisz_wspolrzedne() znajdującej się w pliku wizja.cpp i zapisane w nowym wektorze „wektorVS”. Teraz robot „wie” gdzie znajdują się jego pionki.
Podobnie rzecz ma się z położeniem pionka w odpowiednie żółte pole.

Jeżeli robot wykona swój ruch czas byśmy my teraz wykonali ruch.
W programie kółko i krzyżyk działo się to w ten sposób, że wpisywaliśmy na klawiaturze wybrany numer pola.
W naszym projekcie dokonaliśmy jednak zmiany, tak żeby kładąc pionek (zielony) kolejny algorytm mógł zwrócić numer pola na który położyliśmy ten pionek.
Dzieje się to w następujący sposób:

```
      .
      .
      .
  zrob_zdjecie(image,context,imagex,imageConverted);
    for (int p=1; p <= liczba_zielonych; p++){
       sprawdz_gdzie_pionek(p);
  }

   square=x;
```

Najpierw wywołujemy funkcję zrob_zdjecie(), która dokona analizy zielonych obiektów (zwróci położenie). Następnie w wywołujemy funkcję sprawdz_gdzie_pionek() tyle razy ile zostało wykrytych zielonych obiektów.

Funkcja sprawdz_gdzie_pionek():

```
int sprawdz_gdzie_pionek(int square2){

 if((XYZ_green[square2].x < XYZ_yellowp[1].x+6)  &&  (XYZ_green[square2].x
> XYZ_yellowp[1].x-6) &&(XYZ_green[square2].y < XYZ_yellowp[1].y+10)  &&
(XYZ_green[square2].y > XYZ_yellowp[1].y-10) && (board[0][0]==' '))
        x=1;

  else if((XYZ_green[square2].x < XYZ_yellowp[2].x+6)  &&
(XYZ_green[square2].x > XYZ_yellowp[2].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[2].y+10)  &&  (XYZ_green[square2].y > XYZ_yellowp[2].y-10)&&
(board[0][1]==' '))
        x= 2;

  else if((XYZ_green[square2].x <  XYZ_yellowp[3].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[3].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[3].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[3].y-10)&&
(board[0][2]==' '))
        x= 3;

  else if((XYZ_green[square2].x <  XYZ_yellowp[4].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[4].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[4].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[4].y-10) &&
(board[1][0]==' '))
        x=4;
```

```
  else if((XYZ_green[square2].x <  XYZ_yellowp[5].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[5].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[5].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[5].y-10) &&
(board[1][1]==' '))
        x= 5;

  else if((XYZ_green[square2].x <  XYZ_yellowp[6].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[6].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[6].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[6].y-10) &&
(board[1][2]==' '))
        x= 6;

  else if((XYZ_green[square2].x <  XYZ_yellowp[7].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[7].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[7].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[7].y-10) &&
(board[2][0]==' '))
        x= 7;

  else if((XYZ_green[square2].x <  XYZ_yellowp[8].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[8].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[8].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[8].y-10)&&
(board[2][1]==' '))
        x= 8;

  else if((XYZ_green[square2].x <  XYZ_yellowp[9].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[9].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[9].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[9].y-10) &&
(board[2][2]==' '))
        x= 9;
 return 0;}
```

Funkcja ta działa w ten sposób, że sprawdza po kolei czy wykryty zielony obiekt znajduje się na którymś z żółtych pól poprzez porównanie współrzędnych pikselowych (x i y). Jeżeli tak zwraca numer pola tablicy char board [3] [3]. Aby tak się stało dane pole tablicy char board [3] [3] musi być puste. Dodanie sprawdzania tego warunku rozwiązało problem ponownego przypisania numeru pola wcześniej już wykrytemu pionkowi. Zatem po wykonaniu pętli for zmiennej x zostanie przypisana nowa wartość tylko raz bez względu na liczbę wykrytych zielonych obiektów.

5 Podsumowanie i wnioski

Nasz projekt zakładający możliwość prowadzenia z robotem rozgrywki w kółko i krzyżyk zakończył się powodzeniem. Całość kodu sterującego została sporządzona w języku C++.

Częstym pojawiającym się błędem podczas bazowania było przekroczenie dostępnego czasu przeznaczonego na bazowanie czwartej osi. Problem ten został rozwiązany poprzez zwiększenie domyślnego czasu na wykonanie bazowania tejże osi w pliku konfiguracyjnym.

Kolejnymi błędami jest przekroczenie przez manipulator przestrzeni ruchowej podczas bazowania, oraz przerwanie prawidłowo wykonywanego programu. Manipulator mimo nieprawidłowego bazowania przechodzi do dalszego wykonywania instrukcji, dlatego konieczne jest programowanie ograniczenia ruchowego.

Podczas bazowania manipulator nie ma ograniczeń ruchowych i jest w stanie uderzyć napędem piątej osi we własny korpus.
Zmiany oświetlenia w znacznym stopniu utrudniają wykrywanie barw w przestrzeni hsv. Nasz układ działa poprawnie jedynie przy określonych założeniach. Mianowicie nie możemy poruszać planszą podczas gry. Plansza musi znajdować się poziomo, na jednej, stałej i ustalonej wysokości. Program musi na początku wykryć 9 obiektów żółtych (pól planszy) oraz 5 obiektów czerwonych (pionków dla robota). Nie jest przewidziana obsługa tego typu błędów.

6 Możliwości rozbudowy

W przyszłości możemy wzbogacić nasz projekt o nowe funkcje:

- Zastąpienie pionków znakami graficznymi.
- Możliwość poruszania planszą podczas gry.
- Lokalizacja punktów w 3D (zmiana wysokości planszy).
- Zmiana algorytmu kółka i krzyżyk na bardziej wyrafinowaną grę (np.: szachy) w której proces rozpoznawania figur będzie bardziej złożony.

Bibliografia

[1] Gary Bradski; Adrian Kaehler, *Learning OpenCV*, O'Reilly Media 2008.

[2] Praca zbiorowa, *Podstawy robotyki. Teoria i elementy manipulatorów i robotów*, Wydawnictwa Naukowo Techniczne 1999.

[3] Jacob Cornelius Mocebo, The Scorbot-ER 4U. Function Reference and notes for the usbc.dll, http://kurser.iha.dk/eit/i4prj4/USBC-documentation.pdf , ostatni dostęp 12.06.2009.

Dodatek

Plik main.cpp zawiera:

```
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "pgrflycapture.h"
#include <conio.h>
#include "scorbot.h"
#include "ttt.h"
#include "wizja.h"


/*deklaracje tablic 4 elementowych typu 2D32f.
objPts do wpisania rzeczywistych długosci szachownicy,
a imgPts do wpisania wspolrzednych ekranowych z myszki */
CvPoint2D32f objPts[4], imgPts[4];

/*H - macierz transformacji homogenicznej,
p_img - macierz do pobrania wspolrzednych obiektu na ekranie w pikselach,
p_obj - macierz w której beda wspolrzedne danego obiektu w [mm]*/
CvMat *H, *p_img, *p_obj;

int prostokat_nr=-1;//zmienna pomocnicza

CvPoint2D32f XYZ_red[10],XYZ_green[10],XYZ_yellow[20],XYZ_yellowp[20];

char board[3][3];
char computer, user;
int zmienna =1;

IplImage* image;
FlyCaptureContext context;
FlyCaptureImage imagex;
FlyCaptureImage imageConverted;

int liczba_zielonych;
int x=0;


int main( void )
{
      bazuj();

      printf("przekrec szachownica i nacisnij przycisk");
      _getch();
      int found=0;

      float k_x=7*30, k_y=5*30; //rzeczywisty rozmiar prostokata
      objPts[0].x = 0; objPts[0].y = 0;
      objPts[1].x = k_x; objPts[1].y = 0;
      objPts[2].x = k_x; objPts[2].y = k_y;
      objPts[3].x = 0; objPts[3].y = k_y;

      H = cvCreateMat( 3, 3, CV_32F);
      p_img = cvCreateMat( 3, 1, CV_32F);
      p_obj = cvCreateMat( 3, 1, CV_32F);
```

```
	H = (CvMat*)cvLoad("H2.xml");


	image=cvCreateImage(cvSize(1280,960),8,3);

	//obsluga kamery w lab
	imageConverted.pData = (unsigned char*)image->imageData;
	imageConverted.pixelFormat = FLYCAPTURE_BGR;
	flycaptureCreateContext( &context );
	flycaptureInitialize( context, 0 );
	flycaptureStart(context, FLYCAPTURE_VIDEOMODE_ANY,
	FLYCAPTURE_FRAMERATE_ANY );
	flycaptureSetCameraRegister(context,0x884,0x82000002);


	//lokalizacja szachownicy

while(found==0){
		flycaptureGrabImage2( context, &imagex );
		flycaptureConvertImage( context, &imagex, &imageConverted );
		popraw_zdjecie(image);
		cvNamedWindow( "Poprawiony", 1 );


	//pobranie wspolrzednych 4 naroznikow planszy
	int board_w = 8;
	int board_h = 6;
	int corner_count;
	int board_n = board_w * board_h;
	CvSize board_sz = cvSize( board_w, board_h );
	CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
	IplImage *fota = cvCloneImage(image);
	IplImage *gray_image = cvCreateImage(cvGetSize(fota),8,1);//subpixel
	cvCvtColor(fota, gray_image, CV_BGR2GRAY);
	//Find chessboard corners:
	int found = cvFindChessboardCorners( image, board_sz, corners,
&corner_count, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_NORMALIZE_IMAGE
);
	//Get Subpixel accuracy on those corners

	cvFindCornerSubPix(gray_image, corners, corner_count,
cvSize(11,11),cvSize(-1,-1), cvTermCriteria(
CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));

	imgPts[0].x = corners[40].x; imgPts[0].y = corners[40].y;
	imgPts[1].x = corners[47].x; imgPts[1].y = corners[47].y;
	imgPts[2].x = corners[7].x; imgPts[2].y = corners[7].y;
	imgPts[3].x = corners[0].x; imgPts[3].y = corners[0].y;
	printf("\n%f",corners[0].x);
	//Draw it
	cvDrawChessboardCorners(fota, board_sz, corners, corner_count,
found);

	cvNamedWindow("narozniki",1);
	cvShowImage( "narozniki", fota );
	cvReleaseImage(&fota);
	cvReleaseImage(&gray_image);
	//wyznaczenie przeksztalcenia przeliczajacego wsp. obrazu na wsp.
obiektu
	cvGetPerspectiveTransform( imgPts,objPts, H);
```

```
	cvSave("H2.xml",H);
	//IplImage *image2=cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 3
);
	//cvWarpPerspective(image, image2, H, CV_INTER_LINEAR |
CV_WARP_FILL_OUTLIERS);//obraz po przeksztalceniu
	cvShowImage("Poprawiony", image);
	//cvSaveImage("homografia.bmp",image2 );
	if(found==1){
		printf("Znalazlem szachownice, moge teraz grac");
		break;
	}
	else {
		printf("Poloz szachownice tak zebym ja zobaczyl.\n");
		char c = cvWaitKey(500);
		if(c == 'p') {
			c = 0;
			while(c != 'p' && c != 27) {
				c = cvWaitKey(250);
			}
		}

		if( c == 27 ) break;
	}
}
	cvWaitKey(0);

	flycaptureGrabImage2( context, &imagex );
	flycaptureConvertImage( context, &imagex, &imageConverted );
	popraw_zdjecie(image);
	cvNamedWindow( "Poprawiony", 1 );
	// zmienne pomocnicze
	IplImage *binarny= cvCreateImage(cvGetSize(image),IPL_DEPTH_8U, 1);
	IplImage *maska =cvCreateImage(cvGetSize(image),IPL_DEPTH_8U,1);
	//cvCvtColor(image,szary,CV_BGR2GRAY);   //zrob szary

	cvSmooth( image, image, CV_GAUSSIAN, 3, 3 );   //filtr rozmywajacy

	for ( int i=0; i<2; i++ )
	{
		maske_wloz(image,maska,i);//tworzy obraz binarny(maska) na
kolory RGB

		wykryj_bloby(maska, image, i);//analizuje bloby i zaznacza na
obrazie oryginalnym
		wypisz_wspolrzedne(i);  //wypisuje wspolrzedne srodka ciezkosci
pikselowe i rzeczywiste poszczegolnych blobow
	}

	cvShowImage("Poprawiony", image);  //pokaz efekty


	//zagraj w kółko i krzyżyk

	 while(1)
    {
	init_board();
	if (user_first())
	{
	  computer = 'O';
	  user = 'X';
	}
```

```
      else
      {
        computer = 'X';
        user = 'O';
      }
      play_game();
      if (!play_again())
      break;
    }

      cvWaitKey(0);

      cvDestroyAllWindows();    //uwolnij pamiec
      cvReleaseImage(&image);
      cvReleaseImage(&binarny);
      cvReleaseImage(&maska);

      cvReleaseMat(&H);
      cvReleaseMat(&p_img);
      cvReleaseMat(&p_obj);

      return 0;
}
```

Plik wizja.cpp zawiera:

```
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "pgrflycapture.h"
#include <conio.h>
#include "scorbot.h"
#include "ttt.h"
#include "wizja.h"

Cblobs blobs;
/*deklaracje tablic 4 elementowych typu 2D32f.
objPts do wpisania rzeczywistych długosci szachownicy,
a imgPts do wpisania wspolrzednych ekranowych z myszki */
extern CvPoint2D32f objPts[4], imgPts[4];

/*H - macierz transformacji homogenicznej,
p_img - macierz do pobrania wspolrzednych obiektu na ekranie w pikselach,
p_obj - macierz w której beda wspolrzedne danego obiektu w [mm]*/
extern CvMat *H, *p_img, *p_obj;
extern CvPoint2D32f XYZ_red[10];
extern CvPoint2D32f XYZ_green[10];
extern CvPoint2D32f XYZ_yellow[20];
extern CvPoint2D32f XYZ_yellowp[20];
extern int liczba_zielonych;

void wypisz_wspolrzedne(int color)
{

            for (int i=1; i<= blobs.BlobCount; i++)
            {
                  float xx=blobs.RegionData[i][BLOBSUMX];
                  float yy=blobs.RegionData[i][BLOBSUMY];
```

```
                    CV_MAT_ELEM(*p_img, float,0,0) = (float)xx;
                    CV_MAT_ELEM(*p_img, float,1,0) = (float)yy;
                    CV_MAT_ELEM(*p_img, float,2,0) = 1.0;
                    cvMatMul(H,p_img,p_obj);


                    if (color==0){
                          printf("\n");
                          printf("wsp. ekranowe obiektu czerwonego nr.%d:
%.3fx%.3f",i,xx,yy);
                          printf("\nwsp. na obiekcie:
%.3fx%.3f",CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0),CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0));
                          printf("\nwsp. dla robota:
%.3fx%.3f",CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+301.31,CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+25.48);
                          printf("\n");
                          XYZ_red[i].x=CV_MAT_ELEM(*p_obj,
float,0,0)/CV_MAT_ELEM(*p_obj, float,2,0)+301.31;
                          XYZ_red[i].y=CV_MAT_ELEM(*p_obj,
float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0)+25.48;
                    }
                    else if (color==2){
                          printf("\n");
                          printf("wsp. ekranowe obiektu zielonego nr.%d:
%.3fx%.3f",i,xx,yy);
                          printf("\nwsp. na obiekcie:
%.3fx%.3f",CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0),CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0));
                          printf("\nwsp. dla robota:
%.3fx%.3f",CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+301.31,CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+25.48);
                          printf("\n");
                          XYZ_green[i].x=xx;
                          XYZ_green[i].y=yy;
                    }
                    else if(color==1){
                          printf("\n");
                          printf("wsp. ekranowe obiektu niebieskiego nr.%d:
%.3fx%.3f",i,xx,yy);
                          printf("\nwsp. na obiekcie:
%.3fx%.3f",CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0),CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0));
                          printf("\nwsp. dla robota:
%.3fx%.3f",CV_MAT_ELEM(*p_obj, float,0,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+301.31,CV_MAT_ELEM(*p_obj, float,1,0)/CV_MAT_ELEM(*p_obj,
float,2,0)+25.48);
                          printf("\n");
                          XYZ_yellow[i].x=CV_MAT_ELEM(*p_obj,
float,0,0)/CV_MAT_ELEM(*p_obj, float,2,0)+301.31;
                          XYZ_yellow[i].y=CV_MAT_ELEM(*p_obj,
float,1,0)/CV_MAT_ELEM(*p_obj, float,2,0)+25.48;
                          XYZ_yellowp[i].x=xx;
                          XYZ_yellowp[i].y=yy;
              }
        }
}
```

```
int wykryj_bloby(IplImage *img, IplImage * oryginalny, int color )
{

blobs.BlobAnalysis(img, 0, 0, 1280, 960, 0, 0);
blobs.BlobInclude(BLOBCIRCULARITY,0.8,1.10);
blobs.BlobExclude(BLOBAREA,0,450);

if (color==0){
      printf("\nZnaleziono %d blobow czerwonych\n",blobs.BlobCount);
}
else if (color==2){
      printf("\nZnaleziono %d blobow zielonych\n",blobs.BlobCount);
      liczba_zielonych = blobs.BlobCount;
}
else if (color==1){
      printf("\nZnaleziono %d blobow zoltych\n",blobs.BlobCount);
}

//zaznaczenie wykrytych blobow

for (int i=1;i<=blobs.BlobCount;i++){

cvRectangle(oryginalny,cvPoint(blobs.RegionData[i][BLOBMINX],blobs.RegionDa
ta[i][BLOBMINY]),cvPoint(blobs.RegionData[i][BLOBMAXX],blobs.RegionData[i][
BLOBMAXY]),CV_RGB(0,0,255));

cvEllipse(oryginalny,cvPoint(blobs.RegionData[i][BLOBSUMX],blobs.RegionData
[i][BLOBSUMY]),cvSize(5,5),360, 0, 360, CV_RGB(255,255,255));

    }

return 0;
}


int maske_wloz (IplImage *img, IplImage *mask, int color )
{

      IplImage* hsv=cvCreateImage( cvGetSize(img), 8, 3 ); //obrazek dla
przestrzeni HSV
      cvCvtColor(img,hsv,CV_BGR2HSV); //konwersja na HSV - z tego bedzie
robiona maska kolorow

      //szybki sposob dostepu do pikseli obrazu
      uchar* data = (uchar *)hsv->imageData;
      uchar* dataMASK = (uchar *)mask->imageData;
      int step = hsv->widthStep;
      int stepMASK = mask->widthStep;

      for (int i = 0; i < hsv->height; i++)
            for (int j = 0; j < hsv->width; j++) {
                  int h=data[i*step+j*3+0];
                  int s=data[i*step+j*3+1];
                  int v=data[i*step+j*3+2];

                  //utworzenie maski binarnej dla danego koloru
                  if(color==0)
                        if(v>=70 && (h>170 || h<6) && s>120) //czerwony
                              dataMASK[i*stepMASK+j] = 255;
                        else
                              dataMASK[i*stepMASK+j] = 0;
```

```
                else if(color==2)
                        if(v>=20 && h>60 &&  h<110 && s>30) //zielony
                                dataMASK[i*stepMASK+j] = 255;
                        else
                                dataMASK[i*stepMASK+j] = 0;


                else if(color==1)
                                if(v>=120 && (h>10 &&  h<25) && s>60)
//zolty
                                        dataMASK[i*stepMASK+j] = 255;
                                else
                                        dataMASK[i*stepMASK+j] = 0;


            }

            //usuniecie szumow i zamkniecie otworow w masce - ponizsze
parametry nie beda zawsze
            //dobrze dzialac, co wiecej nie jest to najszybszy sposob
            if (color==0){
                  cvErode(mask,mask,0,1);
                  cvDilate(mask, mask,0,1);//liczbe erozji i dylacji nalezy
dobierac!!!
                  cvNamedWindow("sama_maska_czerwona",1);
                  cvShowImage("sama_maska_czerwona",mask);
            }
            else if (color==2){
                  cvErode(mask,mask,0,2);
                  cvDilate(mask,mask,0,2);
                  cvNamedWindow("sama_maska_zielona",1);
                  cvShowImage("sama_maska_zielona",mask);
            }
            else if (color==1){
                  cvErode(mask,mask,0,1);
                  cvDilate(mask,mask,0,2);
                  cvNamedWindow("sama_maska_zolta",1);
                  cvShowImage("sama_maska_zolta",mask);
            }

            cvReleaseImage(&hsv);


return 0;
}


void popraw_zdjecie(IplImage *src){


      cvNamedWindow( "Oryginalny", 1 );
      cvShowImage( "Oryginalny", src ); // Show raw image

      CvMat* intrinsic = (CvMat*)cvLoad("intrinsic_duza.xml");//pobranie
parametrow kamery
      CvMat* distortion = (CvMat*)cvLoad("distortion_duza.xml");

      //wyznaczenie map przeksztalcen korygujacych obraz kamery - raz dla
danej kamery!
```

```
      IplImage* mapx = cvCreateImage( cvGetSize(src), IPL_DEPTH_32F, 1 );
      IplImage* mapy = cvCreateImage( cvGetSize(src), IPL_DEPTH_32F, 1 );
      cvInitUndistortMap(intrinsic,distortion,mapx,mapy);

      //usuniecie zniekształcen
      IplImage *t = cvCloneImage(src);
      cvRemap( t, src, mapx, mapy, CV_INTER_CUBIC | CV_WARP_FILL_OUTLIERS
);
      cvReleaseImage(&t);
      cvReleaseImage(&mapx);
      cvReleaseImage(&mapy);
}



void zrob_zdjecie (IplImage *image, FlyCaptureContext context,
FlyCaptureImage imagex, FlyCaptureImage imageConverted){

      ////obsluga kamery w lab

      flycaptureGrabImage2( context, &imagex );
      flycaptureConvertImage( context, &imagex, &imageConverted );
    popraw_zdjecie(image);
      cvNamedWindow( "Poprawiony2", 1 );
      cvShowImage("Poprawiony2", image);

      IplImage *binarny2= cvCreateImage(cvGetSize(image),IPL_DEPTH_8U, 1);
      IplImage *maska2 =cvCreateImage(cvGetSize(image),IPL_DEPTH_8U,1);


          cvSmooth( image, image, CV_GAUSSIAN, 3, 3 );     //filtr
rozmywajacy
            maske_wloz(image,maska2,2);                          //tworzy
obraz binarny(maska) na kolory RGB
      wykryj_bloby(maska2, image, 2);                    //analizuje
bloby i zaznacza na obrazie oryginalnym
            wypisz_wspolrzedne(2);                               //wypisuje
wspolrzedne srodka ciezkosci pikselowe i rzeczywiste poszczegolnych blobow



      cvShowImage("Poprawiony2", image);  //pokaz efekty


      cvReleaseImage(&binarny2);
      cvReleaseImage(&maska2);
}
```

Plik ttt.cpp zawiera:

```
#include <stdio.h>
#include "ttt.h"
#include "scorbot.h"
#include "cv.h"
#include "highgui.h"
#include <conio.h>
#include "wizja.h"
#include "pgrflycapture.h"

extern char board[3][3];
extern char computer, user;

extern CvPoint2D32f XYZ_green[10];
extern CvPoint2D32f XYZ_yellow[20];
extern CvPoint2D32f XYZ_yellowp[20];
      extern IplImage* image;
      extern FlyCaptureContext context;
      extern FlyCaptureImage imagex;
      extern FlyCaptureImage imageConverted;

extern int liczba_zielonych;
extern int x;

void init_board(void)
{
  int row, col;

  for (row = 0; row < 3; row++)
    for (col = 0; col < 3; col++)
      board[row][col] = ' ';

  return;
}

/* Display the board to standard output. */
void draw_board(void)
{
  int row, col;

  printf("\n");
  for (row = 0; row < 3; row++)
    {
      printf("   *   *   \n");
      printf(" %c * %c * %c \n", board[row][0], board[row][1],
board[row][2]);
      printf("   *   *   \n");
      if (row != 2)
      printf("***********\n");
    }
  printf("\n");

  return;
}

/* Ask if user wants to go first.  Returns 1 if yes, 0 if no. */
int user_first(void)
{
```

```
  char response;

  printf("Do you want to go first? (y/n) ");
  do
    {
      response = getchar();
    } while ((response != 'y') && (response != 'Y') &&
          (response != 'n') && (response != 'N'));

  if ((response == 'y') || (response == 'Y'))
    return 1;
  else
    return 0;
}

/* Loop through 9 turns or until somebody wins. */
void play_game(void)
{
  int turn;

  for (turn = 1; turn <= 9; turn++)
    {
      /* Check if turn is even or odd to determine which player should
move. */
      if (turn % 2 == 1)
      {
        if (computer == 'X')
          computer_move();
        else
          player_move();
        }
      else
      {
        if (computer == 'O')
          computer_move();
        else
          player_move();
        }

      draw_board();

      if (symbol_won(computer)) {
      printf("\nI WIN!!!\n\n");
      return;
      }
      else if (symbol_won(user)) {
      printf("\nCongratulations, you win!\n\n");
      return;
      }
  }

  printf("\nThe game is a draw.\n\n");
  return;
}

/* Ask if user wants to play again.  Returns 1 if yes, 0 if no. */
int play_again(void)
{
  char response;

  printf("Do you want to play again? (y/n) ");
```

```
  do
    {
      response = getchar();
    } while ((response != 'y') && (response != 'Y') &&
          (response != 'n') && (response != 'N'));

  if ((response == 'y') || (response == 'Y'))
    return 1;
  else
    return 0;
}

/* Choose a move for the computer. */
void computer_move(void)
{
  int square;
  int row, col;

  /* Use first strategy rule that returns valid square */
  square = find_win(computer);
  if (!square)
    square = find_win(user);
  if (!square)
    square = middle_open();
  if (!square)
    square = find_corner();
  if (!square)
    square = find_side();

  printf("\nI am choosing square %d!\n", square);

  row = (square - 1) / 3;
  col = (square - 1) % 3;

  board[row][col] = computer;//tu trzeba dodać kod który każe wstawić
robotowi pionek na wybrane pole
  ruch1(square);
  return;
}

/*
 * Find a win, if any, for the given symbol.
 * If a winning square exists, return the square;
 * Otherwise, return 0;
 */
int find_win(char symbol)
{
  int square, row, col;
  int result = 0;

  /*
   * Loop through the 9 squares.
   * For each, if it is empty, fill it in with the given symbol and check
   * if this has resulted in a win.  If so, keep track of this square in
result.
   * Either way, reset the square to empty afterwards.  After the loop, if
one or
   * more wins have been found, the last will be recorded in result.
   * Otherwise, result will still be 0.
   */
  for (square = 1; square <= 9; square++)
```

```
    {
      row = (square - 1) / 3;
      col = (square - 1) % 3;

      if (board[row][col] == ' ')
      {
        board[row][col] = symbol;
        if (symbol_won(symbol))
          result = square;
        board[row][col] = ' ';
      }
    }

  return result;
}

/* If middle square is empty, return 5; Otherwise return 0. */
int middle_open(void)
{
  if (board[1][1] == ' ')
    return 5;
  else
    return 0;
}

/* Return the number of an empty corner, if one exists; Otherwise return 0.
*/
int find_corner(void)
{
  if (board[0][0] == ' ')
    return 1;
  if (board[0][2] == ' ')
    return 3;
  if (board[2][0] == ' ')
    return 7;
  if (board[2][2] == ' ')
    return 9;

  return 0;
}

/* Return the number of an empty side square, if one exists; Otherwise
return 0. */
int find_side(void)
{
  if (board[0][1] == ' ')
    return 2;
  if (board[1][0] == ' ')
    return 4;
  if (board[1][2] == ' ')
    return 6;
  if (board[2][1] == ' ')
    return 8;

  return 0;
}

/* Check if the given symbol has already one the game. */
int symbol_won(char symbol)
{
  int row, col;
```

```
  for (row = 0; row < 3; row++)
    {
      if ((board[row][0] == symbol) && (board[row][1] == symbol) &&
(board[row][2] == symbol))
      return 1;
    }

  for (col = 0; col < 3; col++)
    {
      if ((board[0][col] == symbol) && (board[1][col] == symbol) &&
(board[2][col] == symbol))
      return 1;
    }

  if ((board[0][0] == symbol) && (board[1][1] == symbol) && (board[2][2] ==
symbol))
    return 1;

  if ((board[0][2] == symbol) && (board[1][1] == symbol) && (board[2][0] ==
symbol))
    return 1;

  return 0;
}

/* Have the user choose a move. */
void player_move(void)
{
  int square;
  int row, col;

  //do//tu trzeba to sprawdzanie pionkow dodac
  //  {
  //    printf("Enter a square: ");
  //    scanf("%d", &square);
  //  } while (!square_valid(square));

  printf("Poloz pionek i nacisnij przycisk... ");
  _getch();
  zrob_zdjecie(image,context,imagex,imageConverted);


  for (int p=1; p <= liczba_zielonych; p++){
       sprawdz_gdzie_pionek(p);
  }

   square=x;
   printf("\n wybralem pole:\t %d",square);
  row = (square - 1) / 3;
  col = (square - 1) % 3;

  board[row][col] = user;

  return;
}

/* Check if the given square is valid and empty. */
int square_valid(int square)
{
  int row, col;
```

```
  row = (square - 1) / 3;
  col = (square - 1) % 3;

  if ((square >= 1) && (square <= 9))
    {
      if (board[row][col] == ' ')
      return 1;
    }

  return 0;
}
int sprawdz_gdzie_pionek(int square2){



 if((XYZ_green[square2].x < XYZ_yellowp[1].x+6)  &&  (XYZ_green[square2].x
> XYZ_yellowp[1].x-6) &&(XYZ_green[square2].y < XYZ_yellowp[1].y+10)  &&
(XYZ_green[square2].y > XYZ_yellowp[1].y-10) && (board[0][0]==' '))
        x=1;

  else if((XYZ_green[square2].x < XYZ_yellowp[2].x+6)  &&
(XYZ_green[square2].x > XYZ_yellowp[2].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[2].y+10)  &&  (XYZ_green[square2].y > XYZ_yellowp[2].y-10)&&
(board[0][1]==' '))
        x= 2;

  else if((XYZ_green[square2].x <  XYZ_yellowp[3].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[3].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[3].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[3].y-10)&&
(board[0][2]==' '))
        x= 3;

  else if((XYZ_green[square2].x <  XYZ_yellowp[4].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[4].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[4].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[4].y-10) &&
(board[1][0]==' '))
        x=4;

  else if((XYZ_green[square2].x <  XYZ_yellowp[5].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[5].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[5].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[5].y-10) &&
(board[1][1]==' '))
        x= 5;

  else if((XYZ_green[square2].x <  XYZ_yellowp[6].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[6].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[6].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[6].y-10) &&
(board[1][2]==' '))
        x= 6;

  else if((XYZ_green[square2].x <  XYZ_yellowp[7].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[7].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[7].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[7].y-10) &&
(board[2][0]==' '))
        x= 7;

  else if((XYZ_green[square2].x <  XYZ_yellowp[8].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[8].x-6) && (XYZ_green[square2].y <
XYZ_yellowp[8].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[8].y-10)&&
(board[2][1]==' '))
```

```
        x= 8;

  else if((XYZ_green[square2].x <  XYZ_yellowp[9].x+6 ) &&
(XYZ_green[square2].x > XYZ_yellowp[9].x-6) &&(XYZ_green[square2].y <
XYZ_yellowp[9].y+10 ) &&  (XYZ_green[square2].y > XYZ_yellowp[9].y-10) &&
(board[2][2]==' '))
        x= 9;

 return 0;
}
```

Plik blobs.cpp zawiera:

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


//*************************************************************//
//* Blob analysis package  Version1.3 5 January 2008        *//
//* Added:                                                  *//
//* - BLOBCOLOR                                             *//
//* History:                                                *//
//* - Version 1.0 8 August 2003                             *//
//* - Version 1.2 3 January 2008                            *//
//* - Version 1.3 5 January 2008                            *//
//* - Version 1.4 13 January 2008                           *//
//*                                                         *//
//* Input: IplImage* binary image                           *//
//* Output: attributes of each connected region             *//
//* Author: Dave Grossman                                   *//
//* Email: dgrossman@cdr.stanford.edu                       *//
//* Acknowledgement: the algorithm has been around > 25 yrs *//
//*************************************************************//

// Transfer fields from subsumed region to correct one


void Cblobs::Subsume(float RegionData[BLOBTOTALCOUNT][BLOBDATACOUNT],
                     int HighRegionNum,
                     int SubsumedRegion[BLOBTOTALCOUNT],
                     int HiNum,
                     int LoNum)
{
      if(HiNum > BLOBTOTALCOUNT) return;

      int iTargetTest;
      int iTargetValid = LoNum;

      while(TRUE) // Follow subsumption chain to lowest number source
      {
            iTargetTest = SubsumedRegion[iTargetValid];
            if(iTargetTest < 0) break;
            iTargetValid = iTargetTest;
      }
      LoNum = iTargetValid;

      int i;
      for(i = BLOBAREA; i < BLOBDATACOUNT; i++)// Skip over BLOBCOLOR
      {
            if(i == BLOBMINX || i == BLOBMINY)
            {
                  if(RegionData[LoNum][i] > RegionData[HiNum][i]) {
RegionData[LoNum][i] = RegionData[HiNum][i]; }
            }
            else if(i == BLOBMAXX || i == BLOBMAXY)
```

```
            {
                  if(RegionData[LoNum][i] < RegionData[HiNum][i]) {
RegionData[LoNum][i] = RegionData[HiNum][i]; }
            }
            else // Area, Perimeter, SumX, SumY, SumXX, SumYY, SumXY
            {
                  RegionData[LoNum][i] += RegionData[HiNum][i];
            }
      }

      SubsumedRegion[HiNum] = LoNum;      // Mark dead region number for
future compression
}

/*float Cblobs::wspol_srodka_ciezkosci(int option)
{
      if(option=0)
      {
            for(ThisRegion = 1; ThisRegion <= BlobCount; ThisRegion++)
            {
                  return RegionData[ThisRegion][BLOBSUMX];

      else
            return RegionData[ThisRegion][BLOBSUMY];
}*/
void Cblobs::PrintRegionDataArray(int option)
{

int ThisRegion;

if(option!=0)
{
      printf("\n      Parent-Color-Area-Perimeter-X-Y-Circularity-
Centricity-Compatness \n");
      for(ThisRegion = 1; ThisRegion <= BlobCount; ThisRegion++)
      {
            printf("R=%3d: ", ThisRegion);
            printf("%3d  ", (int) RegionData[ThisRegion][BLOBPARENT]);
      //parent
            printf("%3d  ", (int) RegionData[ThisRegion][BLOBCOLOR]);
      //color
            printf("%6.1f  ",  RegionData[ThisRegion][BLOBAREA]);
      //area
            printf("%6.1f  ",  RegionData[ThisRegion][BLOBPERIMETER]);
      //perimeter
            printf("%6.1f  ",  RegionData[ThisRegion][BLOBSUMX]);      //X
coordinate of center of mass
            printf("%6.1f  ",  RegionData[ThisRegion][BLOBSUMY]);      //y
coordinate of center of mass
            printf("%6.2f  ",  RegionData[ThisRegion][BLOBCIRCULARITY]);
            //circularity
            printf("%6.2f  ",  RegionData[ThisRegion][BLOBECCENTRICITY]);
            printf("%6.2f  ",  RegionData[ThisRegion][BLOBCOMPACTNESS]);

            printf("\n");
      }
      printf("\n");

}
else
```

```
{

      printf("\nRegionDataArray\nParent-Color--Area---Perimeter---X-----Y--
---BoundingBox\n");
      for(ThisRegion = 0; ThisRegion < BLOBTOTALCOUNT; ThisRegion++)
      {
            if(ThisRegion > 0 && RegionData[ThisRegion][0] < 0) break;
            if(RegionData[ThisRegion][BLOBAREA] <= 0) break;
            printf("R=%3d: ", ThisRegion);
            printf("%3d  ", (int) RegionData[ThisRegion][BLOBPARENT]);
      //parent
            printf("%3d  ", (int) RegionData[ThisRegion][BLOBCOLOR]);
      //color
            printf("%6d  ", (int) RegionData[ThisRegion][BLOBAREA]);
      //area
            printf("%6d  ", (int) RegionData[ThisRegion][BLOBPERIMETER]);
      //perimeter
            printf("%6.1f  ", (float) RegionData[ThisRegion][BLOBSUMX]);
      //sumx
            printf("%6.1f  ", (float) RegionData[ThisRegion][BLOBSUMY]);
      //sumy

            printf("%6.1f  ", (float) RegionData[ThisRegion][BLOBMINX]);
      //minx
            printf("%6.1f  ", (float) RegionData[ThisRegion][BLOBMAXX]);
      //maxx
            printf("%6.1f  ", (float) RegionData[ThisRegion][BLOBMINY]);
      //miny
            printf("%6.1f  ", (float) RegionData[ThisRegion][BLOBMAXY]);
      //maxy
            printf("\n");
      }
      printf("\n");
}
}

int Cblobs::BlobAnalysis(IplImage* ImageHdr,    // input image

      int Col0, int Row0,                              // start of ROI
      int Cols, int Rows,                              // size of ROI (+2 for
Tran array)
      uchar Border,                                    // border color
      int MinArea)                                     // max trans in any
row
{

      // Display Gray image
      //cvNamedWindow("BlobInput", CV_WINDOW_AUTOSIZE);
      //cvShowImage("BlobInput", ImageHdr);


      if(Cols > BLOBCOLCOUNT) { return(-1); } // Bounds check - Error in
column count
      if(Rows > BLOBROWCOUNT) { return(-2); }  // Bounds check - Error in
row count

      char* Image = ImageHdr->imageData;
      int WidthStep = ImageHdr->widthStep;

      // Convert image array into transition array. In each row
      // the transition array tells which columns have a color change
```

```
      int Trans = Cols;                   // max trans in any row

      // row 0 and row Rows+1 represent the border
      int iTran, Tran;                    // Data for a given run
      uchar ThisCell, LastCell;           // Contents (colors?) within this
row
      int ImageOffset = WidthStep * Row0 - WidthStep - 1;  // Performance
booster to avoid multiplication
      long int TransitionOffset = 0;            // Performance booster to
avoid multiplication
      int iRow, iCol;
      int i;

      // Initialize Transition array
      for(i = 0; i < (Rows+2)*(Cols+2); i++) { WorkingStorage[i] = 0; };
      WorkingStorage[0] = WorkingStorage[(Rows+1)*(Cols+2)] = Cols + 2;

      // Fill Transition array
      for(iRow = Row0 + 1; iRow < Row0 + Rows + 1; iRow++)       // Choose
a row of Bordered image
      {
            ImageOffset += WidthStep;     // Performance booster to avoid
multiplication
            TransitionOffset += Cols + 2;// Performance booster to avoid
multiplication
            iTran = 0;                          // Index into Transition
array
            Tran = 0;                           // No transitions at row
start

            if(TransitionOffset + Cols + 1 > WORKINGSTORAGE) break;    //
Bounds check

            LastCell = Border;
            for(iCol = Col0; iCol < Col0 + Cols + 2; iCol++)     // Scan
that row of Bordered image
            {
                  if(iCol == Col0 || iCol == Col0 + Cols + 1) ThisCell =
Border;
                  else ThisCell = Image[ImageOffset + iCol];

                  if(ThisCell != LastCell)
                  {
                        WorkingStorage[TransitionOffset + iTran] = Tran;
      // Save completed Tran
                        iTran++;                                    // Prepare
new index
                        LastCell = ThisCell;                // With this
color
                  }
                  Tran++;     // Tran continues
            }
            WorkingStorage[TransitionOffset + iTran] = Tran;     // Save
completed run
            WorkingStorage[TransitionOffset + iTran + 1] = -1;
      }
      // Process transition code depending on Last row and This row
      //
      // Last ---+++++++--+++++++++++++++++-----++++++++++++++++++++-----
++++++-------+++---
```

```
      // This -----+++-----+++----++++++++---+++++++---++---------------
---+++++++++--
      //
      // There are various possibilities:
      //
      // Case     1       2       3       4       5       6       7       8
      // Last |xxx    |xxxxoo |xxxxxxx|xxxxxxx|ooxxxxx|ooxxx  |ooxxxxx|
xxx|
      // This |    yyy|    yyy|  yyyy |  yyyyy|yyyyyyy|yyyyyyy|yyyy   |yyyy
|
      // Here o is optional
      //
      // Here are the primitive tests to distinguish these 6 cases:
      //   A) Last end < This start - 1 OR NOT       Note: -1
      //   B) This end < Last start OR NOT
      //   C) Last start < This start OR NOT
      //   D) This end < Last end OR NOT
      //   E) This end = Last end OR NOT
      //
      // Here is how to use these tests to determine the case:
      //   Case 1 = A [=> NOT B AND C AND NOT D AND NOT E]
      //   Case 2 = C AND NOT D AND NOT E [AND NOT A AND NOT B]
      //   Case 3 = C AND D [=> NOT E] [AND NOT A AND NOT B]
      //   Case 4 = C AND NOT D AND E [AND NOT A AND NOT B]
      //   Case 5 = NOT C AND E [=> NOT D] [AND NOT A AND NOT B]
      //   Case 6 = NOT C AND NOT D AND NOT E [AND NOT A AND NOT B]
      //   Case 7 = NOT C AND D [=> NOT E] [AND NOT A AND NOT B]
      //   Case 8 = B [=> NOT A AND NOT C AND D AND NOT E]
      //
      // In cases 2,3,4,5,6,7 the following additional test is needed:
      //   Match) This color = Last color OR NOT
      //
      // In cases 5,6,7 the following additional test is needed:
      //   Known) This region was already matched OR NOT
      //
      // Here are the main tests and actions:
      //   Case 1: LastIndex++;
      //   Case 2: if(Match) {y = x;}
      //           LastIndex++;
      //   Case 3: if(Match) {y = x;}
      //           else {y = new}
      //           ThisIndex++;
      //   Case 4: if(Match) {y = x;}
      //           else {y = new}
      //           LastIndex++;
      //           ThisIndex++;
      //   Case 5: if(Match AND NOT Known) {y = x}
      //           else if(Match AND Known) {Subsume(x,y)}
      //           LastIndex++;ThisIndex++
      //   Case 6: if(Match AND NOT Known) {y = x}
      //           else if(Match AND Known) {Subsume(x,y)}
      //           LastIndex++;
      //   Case 7: if(Match AND NOT Known) {y = x}
      //           else if(Match AND Known) {Subsume(x,y)}
      //           ThisIndex++;
      //   Case 8: ThisIndex++;

      // BLOBTOTALCOUNT is max num of regions incl all temps and background
      // BLOBROWCOUNT is the number of rows in the image
      // BLOBCOLCOUNT is the number of columns in the image
```

```
      // BLOBDATACOUNT is number of data elements for each region as
follows:
      // BLOBPARENT 0   these are the respective indices for the data
elements
      // BLOBCOLOR 1          0=background; 1=non-background
      // BLOBAREA 2
      // BLOBPERIMETER 3
      // BLOBSUMX 4           means
      // BLOBSUMY 5
      // BLOBSUMXX 6          2nd moments
      // BLOBSUMYY 7
      // BLOBSUMXY 8
      // BLOBMINX 9           bounding rectangle
      // BLOBMAXX 10
      // BLOBMINY 11
      // BLOBMAXY 12

      int SubsumedRegion[BLOBTOTALCOUNT];            // Blob result array
      int RenumberedRegion[BLOBTOTALCOUNT];          // Blob result array

      float ThisParent; // These data can change when the line is current
      float ThisArea;
      float ThisPerimeter;
      float ThisSumX;
      float ThisSumY;
      float ThisSumXX;
      float ThisSumYY;
      float ThisSumXY;
      float ThisMinX;
      float ThisMaxX;
      float ThisMinY;
      float ThisMaxY;
      float LastPerimeter;    // This is the only data for retroactive
change

      int HighRegionNum = 0;
      int RegionNum = 0;
      int ErrorFlag = 0;

      int LastRow, ThisRow;               // Row number
      int LastStart, ThisStart;           // Starting column of run
      int LastEnd, ThisEnd;               // Ending column of run
      int LastColor, ThisColor;           // Color of run

      int LastIndex, ThisIndex;           // Which run are we up to?
      int LastIndexCount, ThisIndexCount; // Out of these runs
      int LastRegionNum, ThisRegionNum;   // Which assignment?
      int LastRegion[BLOBCOLCOUNT+2];     // Row assignment of region number
      int ThisRegion[BLOBCOLCOUNT+2];     // Row assignment of region number

      long int LastOffset = -(Trans + 2);// For performance to avoid
multiplication
      long int ThisOffset = 0;                  // For performance to avoid
multiplication
      int ComputeData;
      int j;

      for(i = 0; i < BLOBTOTALCOUNT; i++)// Initialize result arrays
      {
            RenumberedRegion[i] = i;                // Initially no region
is renumbered
```

```
            SubsumedRegion[i] = -1;                  // Flag indicates
region is not subsumed
            RegionData[i][0] = (float) -1;           // Flag indicates null
region
            for(j = 1; j < BLOBDATACOUNT; j++)
            {
                  if(j == BLOBMINX || j == BLOBMINY) RegionData[i][j] =
(float) 1000000.0;
                  else RegionData[i][j] = (float) 0.0;
            }
      }
      for(i = 0; i < BLOBROWCOUNT + 2; i++)    // Initialize result arrays
      {
            LastRegion[i] = -1;
            ThisRegion[i] = -1;
      }

      RegionData[0][BLOBPARENT] = (float) -1;
      RegionData[0][BLOBAREA] = (float) WorkingStorage[0];
      RegionData[0][BLOBPERIMETER] = (float) (2 + 2 * WorkingStorage[0]);

      ThisIndexCount = 1;
      ThisRegion[0] = 0;      // Border region

      // Initialize left border column
      for(i = Row0 + 1; i < Row0 + Rows + 2; i++) { ThisRegion[i] = -1; }
// Flag as uninit

      // Loop over all rows
      for(ThisRow = Row0 + 1; ThisRow < Row0 + Rows + 2; ThisRow++)
      {
            ThisOffset += Trans + 2;
            ThisIndex = 0;

            LastOffset += Trans + 2;
            LastRow = ThisRow - 1;
            LastIndexCount = ThisIndexCount;
            LastIndex = 0;

            int EndLast = 0;
            int EndThis = 0;
            for(j = 0; j < Trans + 2; j++)
            {
                  int Index = ThisOffset + j;
                  int TranVal = WorkingStorage[Index];           // Run
length
                  if(TranVal > 0) ThisIndexCount = j + 1;  // stop at
highest

                  if(ThisRegion[j] == -1)  { EndLast = 1; }
                  if(TranVal < 0) { EndThis = 1; }

                  if(EndLast > 0 && EndThis > 0) { break; }

                  LastRegion[j] = ThisRegion[j];
                  ThisRegion[j] = -1;          // Flag indicates region is
not initialized
            }

            int MaxIndexCount = LastIndexCount;
```

```
            if(ThisIndexCount > MaxIndexCount) MaxIndexCount =
ThisIndexCount;


            // Main loop over runs within Last and This rows
            while (LastIndex < LastIndexCount && ThisIndex <
ThisIndexCount)
            {
                  ComputeData = 0;

                  if(LastIndex == 0) LastStart = 0;
                  else LastStart = WorkingStorage[LastOffset + LastIndex -
1];
                  LastEnd = WorkingStorage[LastOffset + LastIndex] - 1;
                  LastColor = LastIndex - 2 * (LastIndex / 2);
                  LastRegionNum = LastRegion[LastIndex];

                  if(ThisIndex == 0) ThisStart = 0;
                  else ThisStart = WorkingStorage[ThisOffset + ThisIndex -
1];
                  ThisEnd = WorkingStorage[ThisOffset + ThisIndex] - 1;
                  ThisColor = ThisIndex - 2 * (ThisIndex / 2);
                  ThisRegionNum = ThisRegion[ThisIndex];

                  if(ThisRegionNum >= BLOBTOTALCOUNT)// Bounds check
                  {
                        ErrorFlag = -2; // Too many regions found - You
must increase BLOBTOTALCOUNT
                        break;
                  }

                  int TestA = (LastEnd < ThisStart - 1);    // initially
false
                  int TestB = (ThisEnd < LastStart);        // initially
false
                  int TestC = (LastStart < ThisStart);      // initially
false
                  int TestD = (ThisEnd < LastEnd);
                  int TestE = (ThisEnd == LastEnd);

                  int TestMatch = (ThisColor == LastColor);       //
initially true
                  int TestKnown = (ThisRegion[ThisIndex] >= 0);   //
initially false

                  int Case = 0;
                  if(TestA) Case = 1;
                  else if(TestB) Case = 8;
                  else if(TestC)
                  {
                        if(TestD) Case = 3;
                        else if(!TestE) Case = 2;
                        else Case = 4;
                  }
                  else
                  {
                        if(TestE) Case = 5;
                        else if(TestD) Case = 7;
                        else Case = 6;
                  }
```

```
                  // Initialize common variables
                  ThisArea = (float) 0.0;
                  ThisSumX = ThisSumY = (float) 0.0;
                  ThisSumXX = ThisSumYY = ThisSumXY = (float) 0.0;
                  ThisMinX = ThisMinY = (float) 1000000.0;
                  ThisMaxX = ThisMaxY = (float) -1.0;
                  LastPerimeter = ThisPerimeter = (float) 0.0;
                  ThisParent = (float) -1;

                  // Determine necessary action and take it
                  switch (Case)
                  {
                        case 1: //|xxx    |
                                    //|    yyy|

                              ThisRegion[ThisIndex] = ThisRegionNum;
                              LastRegion[LastIndex] = LastRegionNum;
                              LastIndex++;
                              break;


                        case 2: //|xxxxoo |
                                    //|    yyy|

                              if(TestMatch)     // Same color
                              {
                                    ThisRegionNum = LastRegionNum;
                                    ThisArea = (float) ThisEnd - ThisStart +
1;
                                    LastPerimeter = (float) LastEnd -
ThisStart + 1;    // to subtract
                                    ThisPerimeter = 2 + 2 * ThisArea -
LastPerimeter;
                                    ComputeData = 1;
                              }

                              ThisRegion[ThisIndex] = ThisRegionNum;
                              LastRegion[LastIndex] = LastRegionNum;
                              LastIndex++;
                              break;


                        case 3: //|xxxxxxx|
                                    //|  yyyy |

                              if(TestMatch)     // Same color
                              {
                                    ThisRegionNum = LastRegionNum;
                                    ThisArea = (float) ThisEnd - ThisStart +
1;
                                    LastPerimeter = ThisArea;     // to
subtract
                                    ThisPerimeter = 2 + ThisArea;
                              }
                              else        // Different color => New region
                              {
                                    ThisParent = (float) LastRegionNum;
                                    ThisRegionNum = ++HighRegionNum;
                                    ThisArea = (float) ThisEnd - ThisStart +
1;
                                    ThisPerimeter = 2 + 2 * ThisArea;
```

```
                        }

                        ThisRegion[ThisIndex] = ThisRegionNum;
                        LastRegion[LastIndex] = LastRegionNum;
                        ComputeData = 1;
                        ThisIndex++;
                        break;


                  case 4:     //|xxxxxxx|
                              //|  yyyyy|

                        if(TestMatch)     // Same color
                        {
                              ThisRegionNum = LastRegionNum;
                              ThisArea = (float) ThisEnd - ThisStart +
1;
                              LastPerimeter = ThisArea;     // to
subtract
                              ThisPerimeter = 2 + ThisArea;
                        }
                        else        // Different color => New region
                        {
                              ThisParent = (float) LastRegionNum;
                              ThisRegionNum = ++HighRegionNum;
                              ThisArea = (float) ThisEnd - ThisStart +
1;
                              ThisPerimeter = 2 + 2 * ThisArea;
                        }

                        ThisRegion[ThisIndex] = ThisRegionNum;
                        LastRegion[LastIndex] = LastRegionNum;
                        ComputeData = 1;
                        LastIndex++;
                        ThisIndex++;
                        break;


                  case 5:     //|ooxxxxx|
                              //|yyyyyyy|

                        if(!TestMatch && !TestKnown)  // Different
color and unknown => new region
                        {
                              ThisParent = (float) LastRegionNum;
                              ThisRegionNum = ++HighRegionNum;
                              ThisArea = (float) ThisEnd - ThisStart +
1;
                              ThisPerimeter = 2 + 2 * ThisArea;
                        }
                        else if(TestMatch && !TestKnown)    // Same
color and unknown
                        {
                              ThisRegionNum = LastRegionNum;
                              ThisArea = (float) ThisEnd - ThisStart +
1;
                              LastPerimeter = (float) LastEnd -
LastStart + 1;    // to subtract
                              ThisPerimeter = 2 + 2 * ThisArea -
LastPerimeter;
                              ComputeData = 1;
```

```
                              }
                              else if(TestMatch && TestKnown)     // Same
color and known
                              {
                                    LastPerimeter = (float) LastEnd -
LastStart + 1;    // to subtract
                                    ThisPerimeter = - LastPerimeter;
                                    if(ThisRegionNum > LastRegionNum)
                                    {
                                          int iOld;
                                          Subsume(RegionData, HighRegionNum,
SubsumedRegion, ThisRegionNum, LastRegionNum);
                                          for(iOld = 0; iOld <
MaxIndexCount; iOld++)
                                          {
                                                if(ThisRegion[iOld] ==
ThisRegionNum) ThisRegion[iOld] = LastRegionNum;
                                                if(LastRegion[iOld] ==
ThisRegionNum) LastRegion[iOld] = LastRegionNum;
                                          }
                                          ThisRegionNum = LastRegionNum;
                                    }
                                    else if(ThisRegionNum < LastRegionNum)
                                    {
                                          int iOld;
                                          Subsume(RegionData, HighRegionNum,
SubsumedRegion, LastRegionNum, ThisRegionNum);
                                          for(iOld = 0; iOld <
MaxIndexCount; iOld++)
                                          {
                                                if(ThisRegion[iOld] ==
LastRegionNum) ThisRegion[iOld] = ThisRegionNum;
                                                if(LastRegion[iOld] ==
LastRegionNum) LastRegion[iOld] = ThisRegionNum;
                                          }
                                          LastRegionNum = ThisRegionNum;
                                    }
                              }

                              ThisRegion[ThisIndex] = ThisRegionNum;
                              LastRegion[LastIndex] = LastRegionNum;
                              LastIndex++;
                              ThisIndex++;
                              break;


                        case 6:     //|ooxxx  |
                                    //|yyyyyyy|

                              if(TestMatch && !TestKnown)
                              {
                                    ThisRegionNum = LastRegionNum;
                                    ThisArea = (float) ThisEnd - ThisStart +
1;
                                    LastPerimeter = (float) LastEnd -
LastStart + 1;    // to subtract
                                    ThisPerimeter = 2 + 2 * ThisArea -
LastPerimeter;
                                    ComputeData = 1;
                              }
                              else if(TestMatch && TestKnown)
```

```
                              {
                                    LastPerimeter = (float) LastEnd -
LastStart + 1;    // to subtract
                                    ThisPerimeter = - LastPerimeter;
                                    if(ThisRegionNum > LastRegionNum)
                                    {
                                          int iOld;
                                          Subsume(RegionData, HighRegionNum,
SubsumedRegion, ThisRegionNum, LastRegionNum);
                                          for(iOld = 0; iOld <
MaxIndexCount; iOld++)
                                          {
                                                if(ThisRegion[iOld] ==
ThisRegionNum) ThisRegion[iOld] = LastRegionNum;
                                                if(LastRegion[iOld] ==
ThisRegionNum) LastRegion[iOld] = LastRegionNum;
                                          }
                                          ThisRegionNum = LastRegionNum;
                                    }
                                    else if(ThisRegionNum < LastRegionNum)
                                    {
                                          Subsume(RegionData, HighRegionNum,
SubsumedRegion, LastRegionNum, ThisRegionNum);
                                          int iOld;
                                          for(iOld = 0; iOld <
MaxIndexCount; iOld++)
                                          {
                                                if(ThisRegion[iOld] ==
LastRegionNum) ThisRegion[iOld] = ThisRegionNum;
                                                if(LastRegion[iOld] ==
LastRegionNum) LastRegion[iOld] = ThisRegionNum;
                                          }
                                          LastRegionNum = ThisRegionNum;
                                    }
                              }

                              ThisRegion[ThisIndex] = ThisRegionNum;
                              LastRegion[LastIndex] = LastRegionNum;
                              LastIndex++;
                              break;


                        case 7:     //|ooxxxxx|
                                    //|yyyy   |

                              if(!TestMatch && !TestKnown)  // Different
color and unknown => new region
                              {
                                    ThisParent = (float) LastRegionNum;
                                    ThisRegionNum = ++HighRegionNum;
                                    ThisArea = (float) ThisEnd - ThisStart +
1;
                                    ThisPerimeter = 2 + 2 * ThisArea;
                              }
                              else if(TestMatch && !TestKnown)
                              {
                                    ThisRegionNum = LastRegionNum;
                                    ThisArea = (float) ThisEnd - ThisStart +
1;
                                    ThisPerimeter = 2 + ThisArea;
```

```
                                    LastPerimeter = (float) ThisEnd -
LastStart + 1;
                                    ThisPerimeter = 2 + 2 * ThisArea -
LastPerimeter;
                                    ComputeData = 1;
                              }
                              else if(TestMatch && TestKnown)
                              {
                                    LastPerimeter = (float) ThisEnd -
LastStart + 1;    // to subtract
                                    ThisPerimeter = - LastPerimeter;
                                    if(ThisRegionNum > LastRegionNum)
                                    {
                                          Subsume(RegionData, HighRegionNum,
SubsumedRegion, ThisRegionNum, LastRegionNum);
                                          int iOld;
                                          for(iOld = 0; iOld <
MaxIndexCount; iOld++)
                                          {
                                                if(ThisRegion[iOld] ==
ThisRegionNum) ThisRegion[iOld] = LastRegionNum;
                                                if(LastRegion[iOld] ==
ThisRegionNum) LastRegion[iOld] = LastRegionNum;
                                          }
                                          ThisRegionNum = LastRegionNum;
                                    }
                                    else if(ThisRegionNum < LastRegionNum)
                                    {
                                          Subsume(RegionData, HighRegionNum,
SubsumedRegion, LastRegionNum, ThisRegionNum);
                                          int iOld;
                                          for(iOld = 0; iOld <
MaxIndexCount; iOld++)
                                          {
                                                if(ThisRegion[iOld] ==
LastRegionNum) ThisRegion[iOld] = ThisRegionNum;
                                                if(LastRegion[iOld] ==
LastRegionNum) LastRegion[iOld] = ThisRegionNum;
                                          }
                                          LastRegionNum = ThisRegionNum;
                                    }
                              }

                              ThisRegion[ThisIndex] = ThisRegionNum;
                              LastRegion[LastIndex] = LastRegionNum;
                              ThisIndex++;
                              break;

                        case 8:     //|    xxx|
                                    //|yyyy   |

                              ThisRegion[ThisIndex] = ThisRegionNum;
                              LastRegion[LastIndex] = LastRegionNum;
                              ThisIndex++;
                              break;

                        default:
                              ErrorFlag = -1;   // Impossible case
                              break;
                  }     // end switch case
                  if(ErrorFlag != 0) break;
```

```
                    if(ComputeData > 0)
                    {
                          int k;
                          for(k = ThisStart; k <= ThisEnd; k++)
                          {
                                ThisSumX += (float) (k - 1);
                                ThisSumXX += (float) (k - 1) * (k - 1);
                          }
                          float ImageRow = (float) (ThisRow - 1);

                          ThisSumXY = ThisSumX * ImageRow;
                          ThisSumY = ThisArea * ImageRow;
                          ThisSumYY = ThisSumY * ImageRow;

                          if(ThisStart - 1 < (int) ThisMinX) ThisMinX =
(float) (ThisStart - 1);
                          if(ThisMinX < (float) 0.0) ThisMinX = (float) 0.0;
                          if(ThisEnd - 1 > (int) ThisMaxX) ThisMaxX = (float)
(ThisEnd - 1);

                          if(ImageRow < ThisMinY) ThisMinY = ImageRow;
                          if(ThisMinY < (float) 0.0) ThisMinY = (float) 0.0;
                          if(ImageRow > ThisMaxY) ThisMaxY = ImageRow;
                    }

                    if(ThisRegionNum >= 0)
                    {

                          if(ThisRegionNum >= BLOBTOTALCOUNT) // Too many
regions found - You must increase BLOBTOTALCOUNT
                          {
                                ErrorFlag = -2;
                                break;
                          }


                          if(ThisParent >= 0) {
RegionData[ThisRegionNum][BLOBPARENT] = (float) ThisParent; }
                          RegionData[ThisRegionNum][BLOBCOLOR] = (float)
ThisColor;  // New code
                          RegionData[ThisRegionNum][BLOBAREA] += ThisArea;
                          RegionData[ThisRegionNum][BLOBPERIMETER] +=
ThisPerimeter;

                          if(ComputeData > 0)
                          {
                                RegionData[ThisRegionNum][BLOBSUMX] +=
ThisSumX;
                                RegionData[ThisRegionNum][BLOBSUMY] +=
ThisSumY;
                                RegionData[ThisRegionNum][BLOBSUMXX] +=
ThisSumXX;
                                RegionData[ThisRegionNum][BLOBSUMYY] +=
ThisSumYY;
                                RegionData[ThisRegionNum][BLOBSUMXY] +=
ThisSumXY;
                                RegionData[ThisRegionNum][BLOBPERIMETER] -=
LastPerimeter;
                                if(RegionData[ThisRegionNum][BLOBMINX] >
ThisMinX) RegionData[ThisRegionNum][BLOBMINX] = ThisMinX;
```

```
                              if(RegionData[ThisRegionNum][BLOBMAXX] <
ThisMaxX) RegionData[ThisRegionNum][BLOBMAXX] = ThisMaxX;
                              if(RegionData[ThisRegionNum][BLOBMINY] >
ThisMinY) RegionData[ThisRegionNum][BLOBMINY] = ThisMinY;
                              if(RegionData[ThisRegionNum][BLOBMAXY] <
ThisMaxY) RegionData[ThisRegionNum][BLOBMAXY] = ThisMaxY;
                        }
                  }

            }     // end Main loop
            if(ErrorFlag != 0) break;

      }     // end Loop over all rows

      // Subsume regions that have too small area
      int HiNum;
      for(HiNum = HighRegionNum; HiNum > 0; HiNum--)
      {

            if(SubsumedRegion[HiNum] < 0 && RegionData[HiNum][BLOBAREA] <
(float) MinArea)
            {
                  Subsume(RegionData, HighRegionNum, SubsumedRegion, HiNum,
(int) RegionData[HiNum][BLOBPARENT]);
            }
      }

      // Compress region numbers to eliminate subsumed regions
      int iOld;
      int iNew = 0;
      for(iOld = 0; iOld <= HighRegionNum; iOld++)
      {
            if(SubsumedRegion[iOld] >= 0) {     continue; } // Region
subsumed, empty, no further action
            else
            {
                  int iTargetTest;
                  int iTargetValid = (int)  RegionData[iOld][BLOBPARENT];
                  while(TRUE) // Follow subsumption chain
                  {
                        iTargetTest = SubsumedRegion[iTargetValid];
                        if(iTargetTest < 0) break;
                        iTargetValid = iTargetTest;
                  }
                  RegionData[iOld][BLOBPARENT] = (float)
RenumberedRegion[iTargetValid];

                  // Move data from old region number to new region number
                  int j;
                  for(j = 0; j < BLOBDATACOUNT; j++) { RegionData[iNew][j]
= RegionData[iOld][j]; }
                  RenumberedRegion[iOld] = iNew;
                  iNew++;
            }
      }
      HighRegionNum = iNew - 1;                     // Update where the
data ends
      RegionData[HighRegionNum + 1][0] = -1;    // and set end of array flag

      // Normalize summation fields into moments
```

```
	for(ThisRegionNum = 0; ThisRegionNum <= HighRegionNum;
ThisRegionNum++)
	{
		// Extract fields
		float Area = RegionData[ThisRegionNum][BLOBAREA];
		float SumX = RegionData[ThisRegionNum][BLOBSUMX];
		float SumY = RegionData[ThisRegionNum][BLOBSUMY];
		float SumXX = RegionData[ThisRegionNum][BLOBSUMXX];
		float SumYY = RegionData[ThisRegionNum][BLOBSUMYY];
		float SumXY = RegionData[ThisRegionNum][BLOBSUMXY];

		// Get averages
		SumX /= Area;
		SumY /= Area;
		SumXX /= Area;
		SumYY /= Area;
		SumXY /= Area;

		// Create moments
		SumXX -= SumX * SumX;
		SumYY -= SumY * SumY;
		SumXY -= SumX * SumY;
		if(SumXY > -1.0E-14 && SumXY < 1.0E-14)
		{
			SumXY = (float) 0.0; // Eliminate roundoff error
		}
		RegionData[ThisRegionNum][BLOBSUMX] = SumX;
		RegionData[ThisRegionNum][BLOBSUMY] = SumY;
		RegionData[ThisRegionNum][BLOBSUMXX] = SumXX;
		RegionData[ThisRegionNum][BLOBSUMYY] = SumYY;
		RegionData[ThisRegionNum][BLOBSUMXY] = SumXY;

		RegionData[ThisRegionNum][BLOBSUMX] += Col0;
		RegionData[ThisRegionNum][BLOBMINX] += Col0;
		RegionData[ThisRegionNum][BLOBMAXX] += Col0;

		float h = (SumXX + SumYY) * .5;
		float h2= sqrt ( h*h - SumXX * SumYY + SumXY * SumXY);
		RegionData[ThisRegionNum][BLOBMAJORAXIS] = sqrt(h + h2);
		RegionData[ThisRegionNum][BLOBMINORAXIS] = sqrt(h - h2);


		float e=sqrt((SumXX - SumYY) * (SumXX - SumYY) + 4 *
SumXY*SumXY);

		RegionData[ThisRegionNum][BLOBECCENTRICITY]=
(SumXX+SumYY+e)/(SumXX+SumYY-e);

		float rect_area=(RegionData[ThisRegionNum][BLOBMAXX]-
RegionData[ThisRegionNum][BLOBMINX])*(RegionData[ThisRegionNum][BLOBMAXY]-
RegionData[ThisRegionNum][BLOBMINY]);
		RegionData[ThisRegionNum][BLOBCOMPACTNESS] =Area/rect_area;

		float angle=(atan(2*SumXY/(SumXX-SumYY))/2)*180/CV_PI;
		if( RegionData[ThisRegionNum][BLOBMAXX]-
RegionData[ThisRegionNum][BLOBMINX]>RegionData[ThisRegionNum][BLOBMAXY]-
RegionData[ThisRegionNum][BLOBMINY])
			angle=180-angle;
		else
			angle=90-angle;
		RegionData[ThisRegionNum][BLOBORIENTATION] = angle;
```

```
	}

	for(ThisRegionNum = HighRegionNum; ThisRegionNum > 0 ; ThisRegionNum-
-)
	{
		// Subtract interior perimeters
		int ParentRegionNum = (int)
RegionData[ThisRegionNum][BLOBPARENT];
		RegionData[ParentRegionNum][BLOBPERIMETER] -=
RegionData[ThisRegionNum][BLOBPERIMETER];
		RegionData[ThisRegionNum][BLOBCIRCULARITY]
=20.5*RegionData[ThisRegionNum][BLOBAREA]/(RegionData[ThisRegionNum][BLOBPE
RIMETER]*RegionData[ThisRegionNum][BLOBPERIMETER]);

	}

	RegionData[HighRegionNum+1][BLOBPARENT] = -2;

	if(ErrorFlag != 0) return(ErrorFlag);

	BlobCount=HighRegionNum;

	return(HighRegionNum);
 }



//Witek
int Cblobs::BlobInclude(int Criterion, double Low_threshold, double
High_threshold)
{
	BlobCountTmp=1;

	if(Criterion==BLOBPARENT)
	{
		for(int i=1;i<=BlobCount;i++)
		{
			if(RegionData[i][BLOBPARENT]>=Low_threshold &&
RegionData[i][BLOBPARENT]<=High_threshold)
			{
				for(int j=0;j<BLOBDATACOUNT;j++)

	RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
				BlobCountTmp++;
			}

		}
	}

	else if(Criterion==BLOBAREA)
	{
		for(int i=1;i<=BlobCount;i++)
			if(RegionData[i][Criterion]>Low_threshold &&
RegionData[i][Criterion]<High_threshold)
			{
				for(int j=0;j<BLOBDATACOUNT;j++)

	RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
				BlobCountTmp++;
```

```
                  }
      }
      else if(Criterion==BLOBCIRCULARITY)
      {
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][Criterion]>Low_threshold &&
RegionData[i][Criterion]<High_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
      }
      else if(Criterion==BLOBPERIMETER)
      {
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][Criterion]>Low_threshold &&
RegionData[i][Criterion]<High_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
      }
      else if(Criterion==BLOBCOMPACTNESS)
      {
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][Criterion]>Low_threshold &&
RegionData[i][Criterion]<High_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
      }
      else if(Criterion==BLOBECCENTRICITY)
      {
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][Criterion]>Low_threshold &&
RegionData[i][Criterion]<High_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
      }


            //copying filtered blobs to the original array
      BlobCountTmp-=1;
      BlobCount=0;
      for(int i=1;i<=BlobCountTmp;i++)
      {
            BlobCount++;
            for(int j=0;j<BLOBDATACOUNT;j++)
                  RegionData[BlobCount][j]=RegionDataTmp[i][j];
```

```
	}
	//after last correct record putting one that has a negative parent
	RegionData[BlobCount+1][0]=-2;


	return(1);

}

int Cblobs::BlobExclude(int Criterion, double Low_threshold, double
High_threshold)
{
	//filters found blobs based on given criteria
	//for a single parameter criterion only Low_threshold is taken into
consideration
	//fultering is done in a very simple but not optimal way - by
creating another array of filtered
	//blobs and copying it back ont the original - NEEDS OPTIMIZATION
	BlobCountTmp=1;

	switch(Criterion)
	{

	case BLOBAREA:
		for(int i=1;i<=BlobCount;i++)
			if(RegionData[i][Criterion]<Low_threshold ||
RegionData[i][Criterion]>High_threshold)
			{
				for(int j=0;j<BLOBDATACOUNT;j++)

	RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
				BlobCountTmp++;
			}
			break;
	case BLOBPERIMETER:
		for(int i=1;i<=BlobCount;i++)
			if(RegionData[i][Criterion]<Low_threshold ||
RegionData[i][Criterion]>High_threshold)
			{
				for(int j=0;j<BLOBDATACOUNT;j++)

	RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
				BlobCountTmp++;
			}
			break;
	case BLOBCOMPACTNESS:
		for(int i=1;i<=BlobCount;i++)
			if(RegionData[i][Criterion]<Low_threshold ||
RegionData[i][Criterion]>High_threshold)
			{
				for(int j=0;j<BLOBDATACOUNT;j++)

	RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
				BlobCountTmp++;
			}
			break;
	case BLOBPARENT:
		for(int i=1;i<=BlobCount;i++)
			if(RegionData[i][Criterion]<Low_threshold ||
RegionData[i][Criterion]>High_threshold)
			{
```

```
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
                  break;
      case BLOBECCENTRICITY:
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][Criterion]<Low_threshold ||
RegionData[i][Criterion]>High_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
                  break;
      case BLOBCIRCULARITY:
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][Criterion]<Low_threshold ||
RegionData[i][Criterion]>High_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
                  break;

      case BLOBCOLOR:
            for(int i=1;i<=BlobCount;i++)
                  if(RegionData[i][BLOBCOLOR]!=Low_threshold)
                  {
                        for(int j=0;j<BLOBDATACOUNT;j++)

      RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                        BlobCountTmp++;
                  }
                  break;

      case BLOBSIBLING:
            //najpierw wyznaczamy rodzenstwo odfiltrowanych blobow, a nie
wszystkich, tylko aktualnych
            //na poczatku kazdy jest jedynakiem
            for(int i=0;i<BlobCount;i++)
                  RegionData[i][BLOBSIBLING]=1;
            for(int i=1;i<=BlobCount;i++)
                  for(int j=1;j<=BlobCount;j++)
                  {
                        if(i==j) continue;

      if(RegionData[i][BLOBPARENT]==RegionData[j][BLOBPARENT])
                              RegionData[i][BLOBSIBLING]++;
                  }
                  //teraz nastepuje odfiltrowanie
                  for(int i=1;i<=BlobCount;i++)
                        if(RegionData[i][BLOBSIBLING]<Low_threshold ||
RegionData[i][BLOBSIBLING]>High_threshold)
                        {
                              for(int j=0;j<BLOBDATACOUNT;j++)
```

```
	RegionDataTmp[BlobCountTmp][j]=RegionData[i][j];
                                BlobCountTmp++;
                        }

                        break;


	}

	//copying filtered blobs to the original array
	BlobCountTmp-=1;
	BlobCount=0;
	for(int i=1;i<=BlobCountTmp;i++)
	{
		BlobCount++;
		for(int j=0;j<BLOBDATACOUNT;j++)
			RegionData[BlobCount][j]=RegionDataTmp[i][j];

	}
	//after last correct record putting one that has a negative parent
	RegionData[BlobCount+1][0]=-2;

	return(1);
}
```

Plik scorbot.cpp zawiera:

```
#include "error.h"
#include "usbcdef.h"
#include "usbc.h"
#include "extern.h"
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <process.h>
#include <time.h>
#include "cv.h"
#include "highgui.h"


#define HOME_NOTIF_START 0xff
#define HOME_NOTIF_FINISH 0x40


// Max velocity = 165
extern CvPoint2D32f XYZ_red[10];
extern CvPoint2D32f XYZ_green[10];
extern CvPoint2D32f XYZ_yellow[20];
extern int zmienna;
int error;
bool ruch = FALSE;
int baza_pas=0;

void InitEnd(ConfigData *pTheConfigData){
	printf("Inicializacja...\n");
}

void ErrorMessage(ErrorInfo *pTheErrorInfo){
	if(pTheErrorInfo->lNumber = 201){
		baza_pas=1;
	}
	else {
		printf("\nError: %d ", pTheErrorInfo -> lNumber);
		printf("Blad!");
		error = 1;
		printf("%s", pTheErrorInfo -> cOptional);
	}


}

void MotionStart(){
	ruch = TRUE;
}

void MotionEnd(){
	ruch = FALSE;
}

void fnHomingNotif(void* ){
	printf("Bazowanie osi\n");
}
```

```
void BazaPas(){
            Home(7, NULL);

            do {
                  EnterManual(MANUAL_TYPE_ENC);
                  Time('B', 1000);
                  MoveManual(7, 50);
                  Sleep(20000);
                  if(baza_pas==1)
                        break;
            } while(baza_pas != 1);

            printf("Wybazowano pas jezdny. Czekam na dalsze rozkazy, Panie
moj.\n");
      }

void bazuj(){
      //ConfigData Config;
      //int i=0;
      //GetConfig( Config);
      //    GetConfig( Config);
      //i=i+5;
//Config.m_sContrType=123;
      //int bazowaneOsie[] = {1,2,4,3,0,5};

      Initialization(INIT_MODE_ONLINE, ER4USB_SYSTEM_TYPE,
(CallBackFun)&InitEnd, (CallBackFun)&ErrorMessage);

      Control('A', TRUE);
      int bazowanie[]={1,2,4,3,0,5};

      for (int i=0; i<=5; i++)
      {
            Home(bazowanie[i],NULL);
      }

      //Home('A', NULL);
      printf("\nWybazowano\n");

}
int ruch1(int square){

      DefineVector('A', "wektorVS", 5);

      long* plCoorArray = new long[5];
      plCoorArray[0]=261310;
      plCoorArray[1]=25480;
      plCoorArray[2]=109750;
      plCoorArray[3]=-85700;
      plCoorArray[4]=4730;

      Teach("wektorVS",1,plCoorArray,5,ABS_XYZ_A);;


      long* plCoorArray1 = new long[5];
      plCoorArray1[0]=XYZ_red[zmienna].x*1000;
      plCoorArray1[1]=XYZ_red[zmienna].y*1000;
      plCoorArray1[2]=109750;
      plCoorArray1[3]=-85700;
      plCoorArray1[4]=4730;
```

```
	Teach("wektorVS",2,plCoorArray1,5,ABS_XYZ_A);

	long* plCoorArray2 = new long[5];
	plCoorArray2[0]=XYZ_red[zmienna].x*1000;
	plCoorArray2[1]=XYZ_red[zmienna].y*1000;
	plCoorArray2[2]=96750;
	plCoorArray2[3]=-85700;
	plCoorArray2[4]=4730;

	Teach("wektorVS",3,plCoorArray2,5,ABS_XYZ_A);

	long* plCoorArray3 = new long[5];
	plCoorArray3[0]=XYZ_yellow[square].x*1000;
	plCoorArray3[1]=XYZ_yellow[square].y*1000;
	plCoorArray3[2]=109750;
	plCoorArray3[3]=-85700;
	plCoorArray3[4]=4730;

	Teach("wektorVS",4,plCoorArray3,5,ABS_XYZ_A);

	long* plCoorArray4 = new long[5];
	plCoorArray4[0]=XYZ_yellow[square].x*1000;
	plCoorArray4[1]=XYZ_yellow[square].y*1000;
	plCoorArray4[2]=96750;
	plCoorArray4[3]=-85700;
	plCoorArray4[4]=4730;

	Teach("wektorVS",5,plCoorArray4,5,ABS_XYZ_A);

	MoveLinear("wektorVS", 1);
	printf("\n\nWykonalem ruch 1.\nWcisnij dowolny klawisz, aby
kontynuowac prace.");
	_getch();

	OpenGripper();
	Sleep(3000);



	MoveLinear("wektorVS", 2);
	printf("\n\nWykonalem ruch 2");
	Sleep(4000);

	MoveLinear("wektorVS", 3);
	printf("\n\nWykonalem ruch 3");
	Sleep(4000);

	CloseGripper();
	Sleep(3000);

	MoveLinear("wektorVS", 2);
	printf("\n\nWykonalem ruch 2");
	Sleep(2000);

	MoveLinear("wektorVS", 4);
	printf("\n\nWykonalem ruch 1");
	Sleep(6000);

	MoveLinear("wektorVS", 5);
	printf("\n\nWykonalem ruch 2");
```

```
	Sleep(2000);

	OpenGripper();
	Sleep(3000);

	MoveLinear("wektorVS", 4);
	printf("\n\nWykonalem ruch 1");
	Sleep(4000);

	MoveLinear("wektorVS", 1);
	printf("\n\nWykonalem ruch 1");
	Sleep(4000);


	printf("\n\nZakonczono wykonywanie programu.\nWcisnij dowolny
klawisz, aby zakonczyc prace.");
	zmienna++;
	return 0;
	//_getch();
	//ClearPointsAttributes( );
	//CloseUSBC();


}
```

plik wizja.h zawiera:

```
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "blobs.h" //plik nagłówkowy
#include "pgrflycapture.h"
#include <conio.h>
#include "scorbot.h"
#include "ttt.h"


void wypisz_wspolrzedne(int color);
int wykryj_bloby(IplImage *img, IplImage * oryginalny, int color );
int maske_wloz2(IplImage* gray, IplImage* mask, IplImage* binary, int
Threshold1, int Threshold2, int color);
int maske_wloz (IplImage *img, IplImage *mask, int color );
void mouse_callback_1(int event, int x, int y, int flags, void* param);
void popraw_zdjecie(IplImage *src);
void zrob_zdjecie ( IplImage *image, FlyCaptureContext context,
FlyCaptureImage imagex, FlyCaptureImage imageConverted);
```

plik ttt.h zawiera:

```
#include <stdio.h>

void init_board(void);
void draw_board(void);
int user_first(void);
void play_game(void);
int play_again(void);
void computer_move(void);
void player_move(void);
int find_win(char);
int middle_open(void);
int find_corner(void);
int find_side(void);
int symbol_won(char);
int square_valid(int);
int sprawdz_gdzie_pionek(int square);
```

plik scorbot.h zawiera:

```
#include "error.h"
#include "usbcdef.h"
#include "usbc.h"
#include "extern.h"
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <process.h>
#include <time.h>

#define HOME_NOTIF_START 0xff
#define HOME_NOTIF_FINISH 0x40


// Max velocity = 165



void InitEnd(ConfigData *pTheConfigData);

void ErrorMessage(ErrorInfo *pTheErrorInfo);


void MotionStart();

void MotionEnd();

void fnHomingNotif(void* );


void BazaPas();
int scorbot();
void bazuj();
int ruch1(int square);
```

plik blobs.h zawiera:

```
//*********************************************************//
//* Blob analysis package  Version1.41 13 January 2008      *//
//* Added:                                                  *//
//* - BLOBCOLOR                                             *//
//* History:                                                *//
//* - Version 1.0 8 August 2003                             *//
//* - Version 1.2 3 January 2008                            *//
//* - Version 1.3 5 January 2008                            *//
//* - Version 1.4 13 January 2008                                  *//
//* - Version 1.41 16 April 2009                            *//
//*                                                         *//
//* Input: IplImage* binary image                           *//
//* Output: attributes of each connected region             *//
//* Author: Dave Grossman                                   *//
//* Email: dgrossman@cdr.stanford.edu                       *//
//* Acknowledgement: the algorithm has been around > 25 yrs *//
//* - Version 1.41 is modified from original 1.4            *//
//* by Witold Czajewski
     *//
//* Email: W.Czajewski@isep.pw.edu.pl                              *//
//*********************************************************//

// defines for blob array sizes and indices


#ifdef ROWS
#define BLOBROWCOUNT ROWS
#endif

#ifdef COLS
#define BLOBCOLCOUNT COLS
#endif

#ifndef BLOBROWCOUNT
#define BLOBROWCOUNT 3040
#endif

#ifndef BLOBCOLCOUNT
#define BLOBCOLCOUNT 3040
#endif

#ifndef BLOBTOTALCOUNT
#define BLOBTOTALCOUNT (BLOBROWCOUNT + BLOBCOLCOUNT) * 5
#endif

#define WORKINGSTORAGE (BLOBROWCOUNT+2)*(BLOBCOLCOUNT+2)

#define BLOBPARENT 0
#define BLOBCOLOR 1
#define BLOBAREA 2
#define BLOBPERIMETER 3
#define BLOBSUMX 4
#define BLOBSUMY 5
#define BLOBSUMXX 6
#define BLOBSUMYY 7
#define BLOBSUMXY 8
#define BLOBMINX 9
#define BLOBMAXX 10
#define BLOBMINY 11
```

```
#define BLOBMAXY 12
#define BLOBMAJORAXIS 13
#define BLOBMINORAXIS 14
#define BLOBORIENTATION 15
#define BLOBECCENTRICITY 16
#define BLOBCOMPACTNESS 17
#define BLOBCIRCULARITY 18
#define BLOBSIBLING 19
#define BLOBDATACOUNT 20

class Cblobs
{

public:

// Global variables to avoid memory leak
int WorkingStorage[WORKINGSTORAGE];                         // Working
storage Note +2 +2 for image border
float RegionData[BLOBTOTALCOUNT][BLOBDATACOUNT];     // Blob result array

float RegionDataTmp[BLOBTOTALCOUNT][BLOBDATACOUNT]; //Witek - temporary
array for blob filtering
int BlobCount, BlobCountTmp;                                //Witek - global
counters to make things easier

// Subroutine prototypes
void PrintRegionDataArray(int option=0);
//float wspol_srodka_ciezkosci(int option);
void Subsume(float[BLOBTOTALCOUNT][BLOBDATACOUNT], int,
int[BLOBTOTALCOUNT], int, int);
int BlobAnalysis(IplImage*, int, int, int, int, uchar, int);

//Witek - blob filtering functions
int BlobInclude(int Criterion, double Low_threshold, double
High_threshold=0);
int BlobExclude(int Criterion, double Low_threshold, double
High_threshold=0);
};
```