Analiza dopasowania modelu do obiektu po etapie segmentacji obrazu wykorzystującego działy wodne (*ang. Watershed*) przy rozpoznawaniu barw obrazów.

Analysis of the pattern matching to the object after the Watershed image segmentation for image color recognition.

Paweł Iljaszewicz[1]

Streszczenie: W artykule przedstawiono segmentację obrazu bazująca na wykorzystaniu algorytmu działów wodnych (*ang. Watershed algorithm*), które zdobywa w ostatnich latach coraz szersze uznanie, jako skuteczna metoda detekcji obiektów. Wykorzystujemy fakt, iż poszukiwany obiekt powinien wykazywać podobieństwo geometryczne dotyczące kształtu, rozmiaru i położenia. Następnie połączenie tych cech z pobranym obrazem przetwarzanym na bieżąco pozwoli na zwrócenie współrzędnych prostokąta, który otoczy największy obiekt o wybranym kolorze. Celem proponowanego algorytmu będzie umożliwienie dokładnego dopasowania zadanego modelu do poszukiwanego obiektu, co w konsekwencji pozwoli na skuteczną detekcję na wszystkich obrazach z danej serii tematycznej. Omówiono również model wokselowy obiektu. Badania oparto na bibliotekach OpenCV.

Abstract: The article deals with the segmentation and recognition of the image based on the use of Watershed algorithm Watershed algorithm has gained ever greater recognition as an effective method of detecting objects in recent years. We use the fact that the object sought should show the geometric similarity of the shape, Size and position. Then combining these features with the downloaded image to be processed will allow you to return the coordinates of the rectangle, which will the largest object of the selected color. The purpose of the proposed algorithm will be to allow an exact match of the specified model to the object you are looking for, which will result in effective detection of all images from the given series. Also discusses the voxel object model. The study was based on OpenCV libraries.

Słowa kluczowe: OpenCV, analiza obrazu, działy wodne, thermography, dron, RGB, HSV, Model wokselowy obiektu.

Keywords: OpenCV, image analysis, watershed, termovision, drone, RGB, HSV, Voxel OpenCV model,

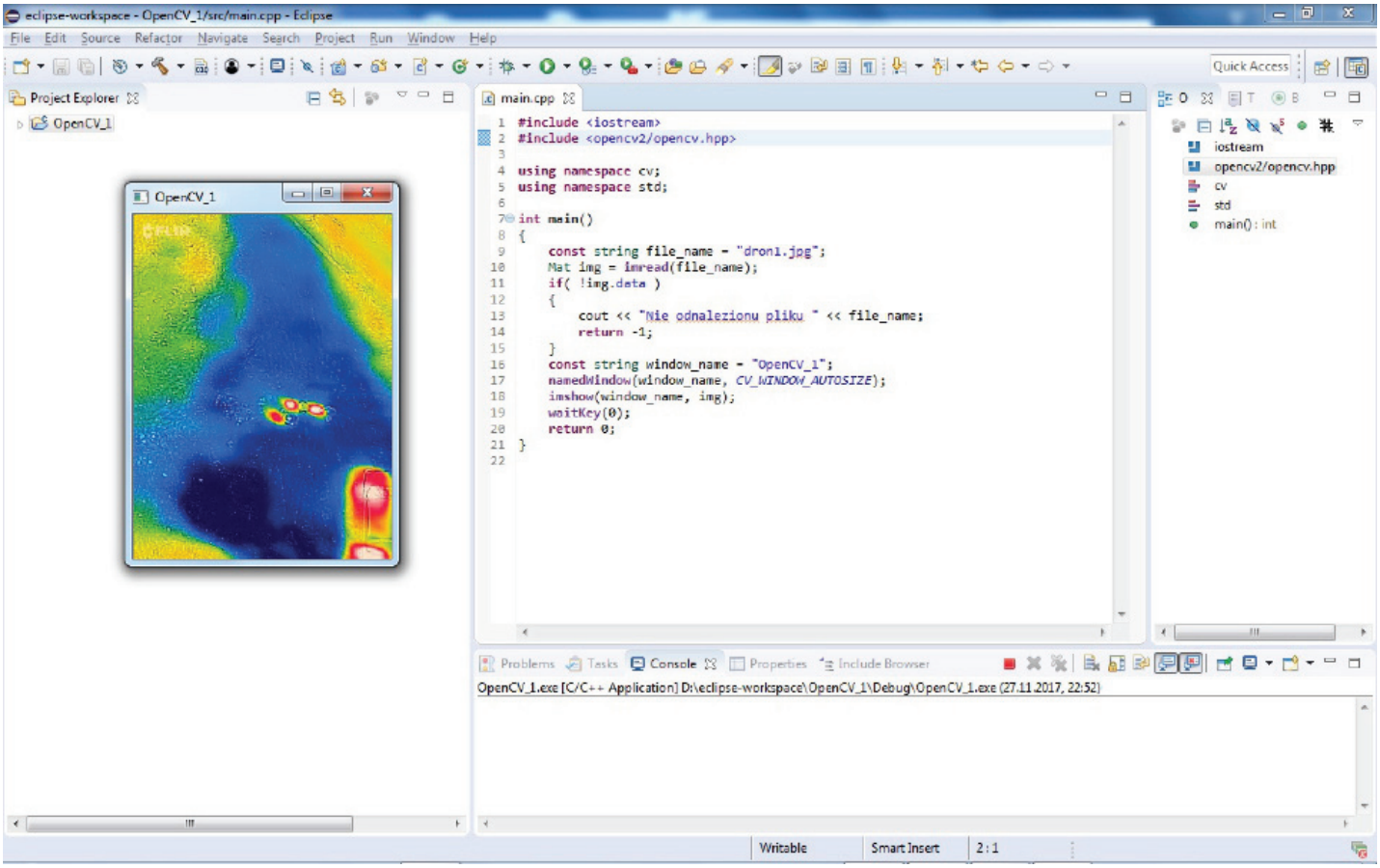
Wstęp

Celem niniejszej publikacji jest przedstawienie zastosowania bibliotek OpenCV wspomagających analizę barw obrazów uzyskiwanych podczas lotów dronem na różnych wysokościach w zależności od ich rozdzielczości. Film został wykonany kamerą termowizyjną.
Zaprezentowane zostaną przekształcenia RGB na HSV wykorzystujące bibliotekę opengl. Szczegółowo opisano mechanizm przejścia z jednej przestrzeni RGB do drugiej HSV. Przedstawiono wyniki konwersji na zdjęciu pobranym z filmu. Następnie
Zastosowanie modelu HSV i jego przekształcenie z RGB w OpenCV
OpenCV (*ang. Open Source Computer Vision Library*) - biblioteka algorytmów wizyjnych komputerowych, przetwarzania obrazu i algorytmów numerycznych do ogólnego przeznaczenia open source. Realizowane w C / C ++, jest również opracowany dla Python, Java, Ruby, Matlab, Lua i innych językach. [1] Można go swobodnie używać w celach akademickich i komercyjnych - rozpowszechniany jest na warunkach licencji BSD . Pozwala na optymalizowanie aplikacji w czasie rzeczywistym. Jest niezależna od systemu, sprzętu ani menadżera okienek. Zawiera nisko i wysokopoziomowy interfejs programowania aplikacji API (*ang. application programming interface*).[2]

1.1 Wczytanie obrazu

Korzystamy ze środowiska Eclipse[3] Pełny opis znajdziemy w dokumentacji OpenCV[4].
Program pisany jest w języku C++. Na rysunku poniżej przedstawiono wynik wczytania obrazu będącego klatką z filmu z drona.

1. Instytut Techniki Cieplnej Politechniki Śląskiej, ul. Konarskiego 22 44-100 Gliwice, email: pawel.iljaszewicz@polsl.pl
2. http://www.intel.com/technology/computing/opencv/
3. http://www.eclipse.org/
4. https://docs.opencv.org/2.4/doc/tutorials/introduction/linux_eclipse/linux_eclipse.html

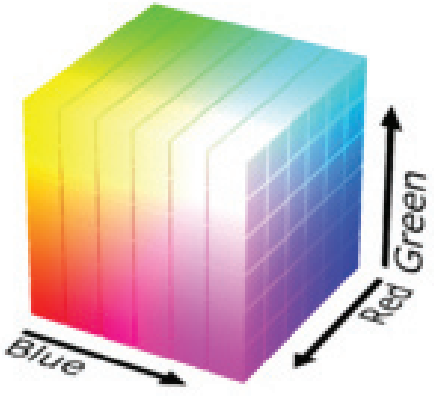


Ryc. 1 Wczytywanie obrazu OpenCV

W OpenCV obraz jest przechowywany w klasach Mat (Mat img = imread(file_name), które mają budowę wzorowaną na macierzach. W klasach można przechowywać również macierze a OpenCV udostępnia szereg funkcji pozwalających na wykonywanie działań na nich. Kolor zapisywany jest w formacie BGR odwrotnym od RGB (*ang. Red, Green, Blue*). Model ten jest rozpowszechniony przy zapisie i wyświetlaniu obrazów w telewizorach, kamerach aparatach itp.[4]

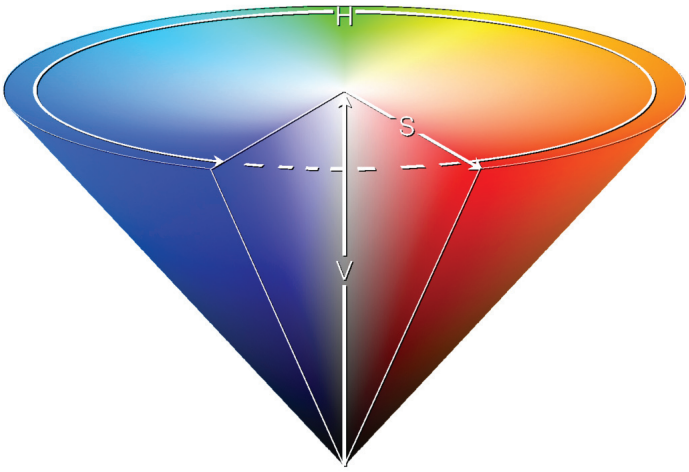
1.2 Przejście z przestrzeni RGB do HSV

Model RBG to przestrzeń o trzech wymiarach, gdzie każdemu punktowi odpowiada współrzędna definiowana względem osi R, G, B. Na rysunku poniżej przedstawiono osie i odpowiadające im punkty kolorów.



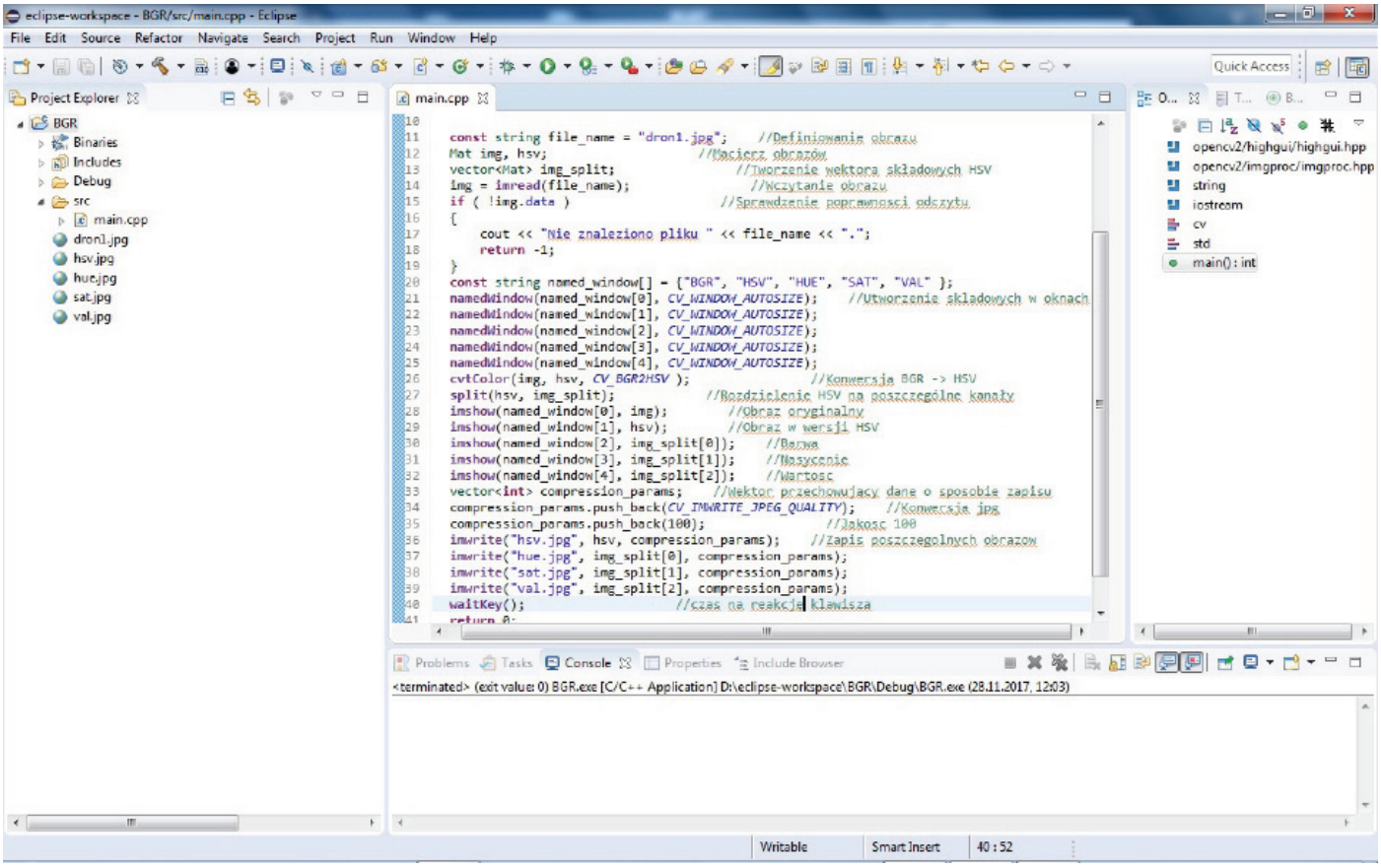
Ryc. 2 Model RGB

Widzimy, że aby zdefiniować kolor elementy macierzy Mat są definiowane przez 3 wartości odpowiadające kolorom B, G, R.
W przypadku modelu HSV (*ang. Hue Saturation Value*) przedstawionym na rysunku poniżej



Ryc. 3 Stożek przestrzeni barw HSV

składowa H barwa (hue) określana jest przez kąt od 0 do 360 stopni. Składowa S nasycenie (saturation), przez odległość od środka na promieniu podstawy, im mniejsza wartość tym bardziej biały kolor. Składowa V wartość (value) definiowana przez wysokość stożka, określa jasność koloru, mniejsza wartość odpowiada ciemniejszemu kolorowi. [3]



Ryc. 5 OpenCV konwertuje RGB na HSV

Jak widać, aby określić kolor wystarczy jedna składowa H. Pokazuje to rzut podstawy stożka.

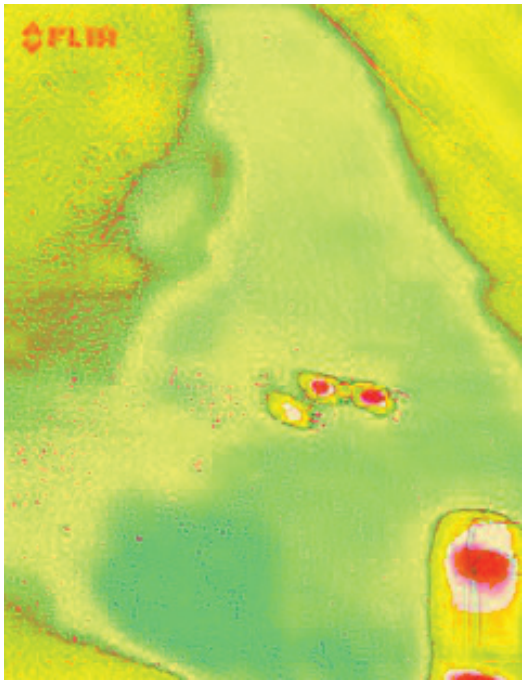
Ryc. 4 Definiowanie koloru żółtego, jako zakres kąta

Gdzie kolor żółty jest definiowany przez odpowiedni zakres kąta.

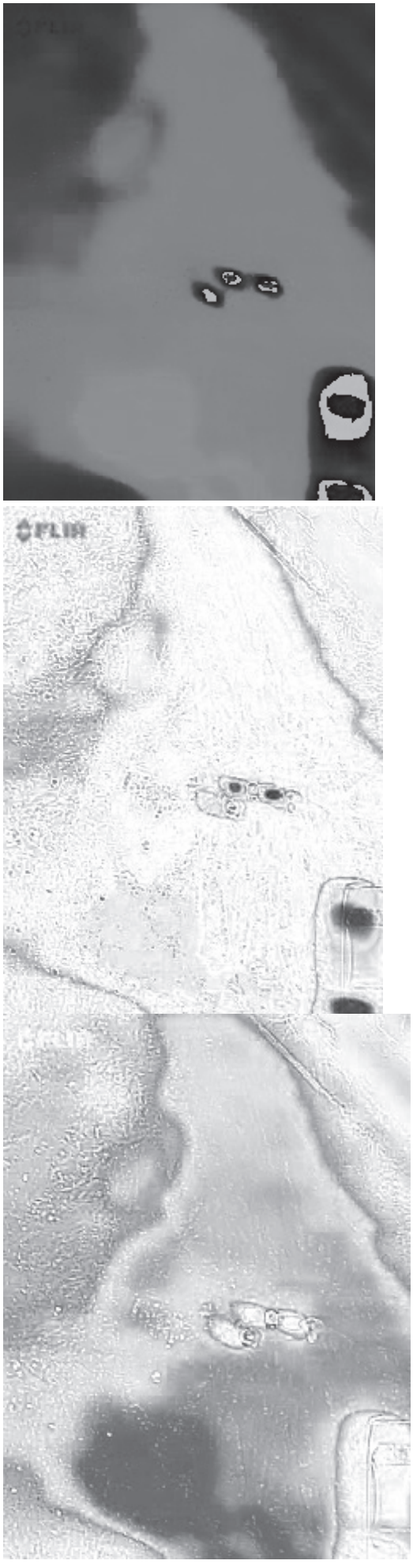
1.3 OpenCV przejście przestrzeni barw

W OpenCV w prosty sposób dokonuje się konwersji. Odpowiedzialna za to funkcja cvtColor(img, hsv, CV_BGR2HSV); pozwala przechodzić pomiędzy obrazami. Pierwszy argument img to obraz, który poddajemy przekształceniu, drugi argument oznacza obraz po konwersji natomiast trzeci CV_BGR2HSV oznacza przejście z BGR na HSV. Oczywiście istnieją inne konwersje szczegółowo opisane w dokumentacji[5]. Na rysunku powyżej pokazano zrzut ekranu pokazujący program konwertujący RGB na HSV.

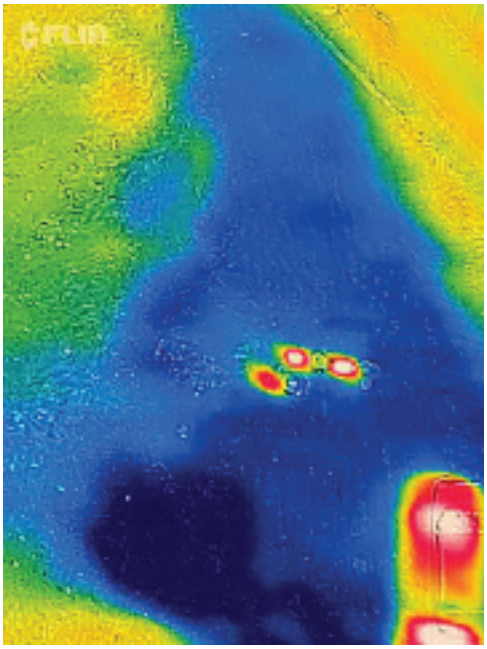
Funkcja split (hsv, img_split); rozdziela obraz na składowe wynikowe przedstawione na rysunkach poniżej. Widzimy obraz wyjściowy a następnie składowe po konwersji. Pierwszy obraz przedstawia obraz *HSV*, drugi składową *hue*, trzeci składową *sat*, czwarty składową *val*.



5. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html



Ryc. 6 Konwersja obrazu wejściowego na składowe HSV, hue, sat i val.



Ryc. 7 Obraz wejściowy img

Warto zwrócić uwagę, iż różne odcienie koloru niebieskiego stanowią ten sam odcień na obrazie reprezentującym barwę *hue*. Te formuły odzwierciedlają pewne osobliwości wartości HSV. Jeśli R= G = B, to wartość H nie ma znaczenia i jest ustalona przez definicję H = 0. Jest to oczywiste. Ponieważ jeśli S = 0 (kolor nienasycony), to chromatyczność znajduje się w środkowej szarej linii, Barwa jest tak nieistotna i nie może być określona w żaden znaczący sposób. Jeśli R = G = B = 0, to S nie ma znaczenia, a z definicji S = 0 jest ustalona. Jeśli wszystkie trzy wartości RGB są "zerowe", to jest obraz czarny i nasycenia koloru traci znaczenie. To samo dotyczy, jednak nie w przypadku MAX = MIN = 1, czyli biały, wartość podana tutaj przez formułę 0 ma znaczenie elementarne, jak widać w powyższej formie stożka. Niezdefiniowane wartości są wypełnione, jako "zero" ze względów obliczeniowych.
W równaniach poniżej przedstawiono podstawowe zależności pomiędzy RGB a HSV

$$H := \begin{cases} 0, & jeśli\ MAX = MIN \Leftrightarrow R = G = B \\ 60^{\circ} \cdot \left(0 + \dfrac{G-B}{MAX-MIN}\right), & jeśli\ MAX = R \\ 60^{\circ} \cdot \left(2 + \dfrac{B-R}{MAX-MIN}\right), & jeśli\ MAX = G \\ 60^{\circ} \cdot \left(4 + \dfrac{R-G}{MAX-MIN}\right), & jeśli\ MAX = B \end{cases} \quad (1)$$

$$S_{\mathrm{HSV}} := \begin{cases} 0, & jesli\ MAX = 0 \Leftrightarrow R = G = B = 0 \\ \dfrac{MAX-MIN}{MAX}, & inaczej \end{cases} \quad (2)$$

$$V := MAX$$
$$L := \frac{MAX + MIN}{2} \quad (3)$$

Możemy powiedzieć, że kolor przestrzeni jest kombinacją modelu kolorów i funkcji mapowania (ta definicja jest dobrze znana). Dla modelu kolorów, możemy zrozumieć każdy model matematyczny, który może być używany do reprezentowania kolory, jako liczby (np. RGB (255, 0, 0) reprezentuje kolor czerwony). Dla funkcji mapowania, możemy zrozumieć każdą funkcję, która może mapować model kolorów do absolutnej przestrzeni kolorów, w celu podłączenia tego systemu kolorów do rzeczywistego świata, dzięki czemu można go używać.
Naszym celem jest wizualizacja każdego z trzech kanałów tych przestrzeni kolorystycznych: RGB, HSV. Ogólnie żaden z nich nie jest w stanie bezwzględne odtworzyć rzeczywisty kolor. Są to jedynie inne systemy kodowania informacji RGB. Nasze obrazy będą odczytywane w BGR (niebiesko-zielono-czerwony), ze względu na domyślne ustawienia OpenCV. Dla każdej z tych przestrzeni kolorów jest funkcja mapowania i można je znaleźć dokumentacji OpenCV cvtColor.[6]

Metoda modelu wokselowego

Woksel to najmniejszy element przestrzeni w grafice 3D (*ang. Voxel volumetric pixel element*) Voxel reprezentuje wartość na regularnej siatce w przestrzeni trójwymiarowej[7]. Podobnie jak piksele w bitmapie, same woksele zazwyczaj nie mają swojej pozycji (współrzędnych) wyraźnie zakodowanej wraz z ich wartościami. Zamiast tego systemy renderujące określają pozycję woksela na podstawie jego pozycji względem innych wokseli (tj. jego pozycja w strukturze danych, która tworzy pojedynczy obraz wolumetryczny). W przeciwieństwie do pikseli i wokseli, punkty i wielokąty często są jawnie reprezentowane przez współrzędne ich wierzchołków. Bezpośrednią konsekwencją tej różnicy jest to, że wielokąty mogą efektywnie reprezentować proste struktury 3D z dużą ilością pustej lub jednorodnie wypełnionej przestrzeni, podczas gdy woksele doskonale reprezentują regularnie próbkowane przestrzenie, które są niejednorodnie wypełnione.[2]
Metoda implementująca woksele jest techniką śledzenia wielosegmentowego obiektu w przestrzeni trójwymiarowej. Każdy z obiektów ma jednoznacznie określoną objętość. Pozwala to na jego wysegmentowanie, pomimo, iż może być przesłonięty przez inne obiekty. Znając przestrzeń zajmowaną przez obiekt łatwo można go rzutować na zdefiniowaną płaszczyznę. Łatwo można wyselekcjonować i śledzić obiekt gdyż ma przypisany do siebie indywidualny numer.
Obiekt jest na tej podstawie śledzony przez odpowiedni mechanizm, dopóki nie opuści zakładanego obszaru sceny. Segmentacja sceny z wykorzystaniem modelu wokselowego w prosty sposób identyfikuje obiekt i przypisuje mu indywidualny znacznik, w odróżnieniu do innych przekształceń macierzy homograficznych i klasycznych metodach detekcji obszaru poruszającego się i jego klasyfikacji.
Przykładowa implementacja używająca biblioteki OpenCV opengl suport gdzie chmura punktów jest reprezentowana w Voxel OpenCV. Każda z 8 pozycji definiuje trójwymiarowy wektor uzyskany podczas segmentacji obiektu.[8]

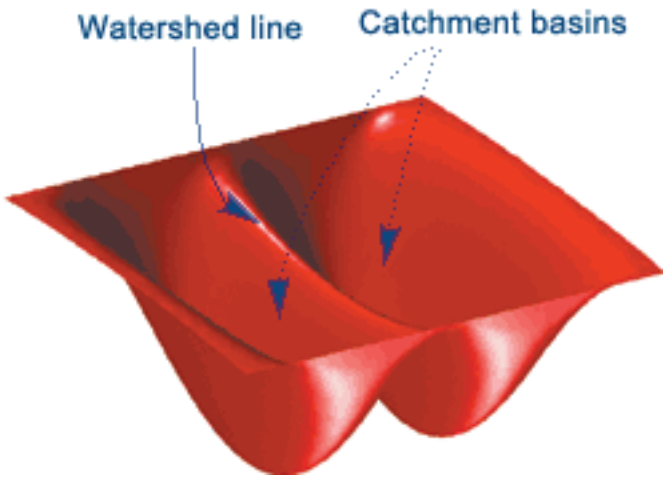
Tab. 1 Implementacja chmury punktów Voxel OpenCV

```
#include <opencv2/opencv.hpp>
// shame, needs windows.h on win, else problem with
APIENTRY
//#include <windows.h>
#include <GL/gl.h>

using namespace std;
using namespace cv;

float pts[8*3] = {
   0.3255598,0.2123329,0.4342422,
   0.234444,0.4323432,0.4324234,
   0.668886,0.387868,0.51884323,
   0.686878,0.567577,0.7675777,
   0.234565,0.2675675,0.356777,
   0.3456567,0.523435,0.3446577,
   0.237577,0.2343434,0.324344433,
   0.24353,0.35765756,0.2455345
};

void on_opengl(void* param)
{
   glLoadIdentity();
   glPointSize(3);
   glBegin(GL_POINTS);
   for ( int i=0; i<8; i++ )
   {
      glColor3ub( i*20, 100+i*10, i*42 );
      glVertex3fv(&pts[i*3]);
   }
   glEnd();
}

int main(int argc, char *argv[])
{
      namedWindow("3d",CV_WINDOW_OPENGL|CV_
WINDOW_NORMAL);
   setOpenGlDrawCallback("3d",on_opengl,0);
   waitKey(0);
   return 0;
}
```

6. www.opencv.org
7. https://github.com/clickteam-plugin/OpenCV/blob/master/Extension/Main/Voxel.h-
8. https://docs.opencv.org/2.4.13.2/modules/core/doc/basic_structures.html

Algorytm działów wodnych

Algorytm działu wodnego jest klasycznym algorytmem służącym do segmentacji i jest szczególnie przydatny przy wydobywaniu z obrazu obiektów, dotykających lub nakładających się. Każdy obraz w skali szarości może być postrzegany jako powierzchnia topograficzna, gdzie wysoka intensywność oznacza szczyty i wzgórza, a niska intensywność oznacza doliny. W algorytmie zaczyna się wypełniać każdą izolowaną dolinę (lokalne minima) różnokolorową wodą (etykiety) tworząc tzw. zlewiska (ang. catchment basins), gdy woda wznosi się, w zależności od pobliskich szczytów, wody z różnych dolin, oczywiście z różnymi kolorami zaczynają się łączyć. Aby tego uniknąć, buduje się bariery w miejscach, gdzie woda się łączy. Kontynuuje się pracę nad napełnianiem wody i budowaniem barier, dopóki wszystkie szczyty nie znajdą się pod wodą. Następnie utworzone bariery dają wynik segmentacji. To jest "filozofia" stojąca za działem wodnym.
Możesz odwiedzić stronę CMM w dziale wodnym, aby zrozumieć ją za pomocą animacji.
Ale to podejście daje wynik przekrojowy z powodu szumu lub innych nieprawidłowości w obrazie. Tak, więc OpenCV wdrożył algorytm wodny oparty na markerach, w którym określa się, które punkty doliny mają zostać połączone, a które nie. Jest to interaktywna segmentacja obrazu. To, co robimy, to dawanie różnych etykiet dla naszego znanego nam obiektu. Oznacza się region, którego jesteśmy pewni, że jest pierwszym planem lub obiektem o jednym kolorze (lub intensywności), oznacza się region, z którego jesteśmy pewni, że jest tłem lub nie jest obiektem o innym kolorze, i ostatecznie region, którego nie jesteśmy pewni, oznacz to, jako 0. To jest nasz marker. Następnie zastosuj algorytm działu wodnego. Następnie nasz marker zostanie zaktualizowany o etykiety, które daliśmy, a granice obiektów będą miały wartość -1.



Ryc. 8 Wypełnienia algorytmu działem wodnym i zlewiskiem

Poniżej przykład algorytmu[9]. Rozpatrujemy pojedynczy obraz. W pierwszej kolejności ładowany jest obraz „image”.

Tab. 2 Algorytm działów wodnych OpenCV

```
#include <opencv2/core/utility.hpp>
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <cstdio>
#include <iostream>
using namespace cv;
using namespace std;
static void help()
{
cout << "\nThis program demonstrates the famous water-
shed segmentation algorithm in OpenCV: watershed()\n"
"Usage:\n"
"./watershed [image_name -- default is ../data/fruits.jpg]\n"
<< endl;
cout << "Hot keys: \n"
"\tESC - quit the program\n"
"\tr - restore the original image\n"
"\tw or SPACE - run watershed segmentation
algorithm\n"
"\t\t(before running it, *roughly* mark the areas to seg-
ment on the image)\n"
"\t (before that, roughly outline several markers on the
image)\n";
}
Mat markerMask, img;
Point prevPt(-1, -1);
static void onMouse( int event, int x, int y, int flags, void*
)
{
if( x < 0 || x >= img.cols || y < 0 || y >= img.rows )
return;
if( event == EVENT_LBUTTONUP || !(flags & EVENT_
FLAG_LBUTTON) )
prevPt = Point(-1,-1);
else if( event == EVENT_LBUTTONDOWN )
prevPt = Point(x,y);
else if( event == EVENT_MOUSEMOVE && (flags &
EVENT_FLAG_LBUTTON) )
{
Point pt(x, y);
if( prevPt.x < 0 )
prevPt = pt;
line( markerMask, prevPt, pt, Scalar::all(255), 5, 8, 0 );
line( img, prevPt, pt, Scalar::all(255), 5, 8, 0 );
prevPt = pt;
imshow("image", img);
}
}
int main( int argc, char** argv )
{
cv::CommandLineParser parser(argc, argv, "{help h | | }{
@input | ../data/dron1.jpg | }");
if (parser.has("help"))
{
help();
```

9. https://docs.opencv.org/3.3.1/d8/da9/watershed_8cpp-example.html

```
return 0;
}
string filename = parser.get<string>("@input");
Mat img0 = imread(filename, 1), imgGray;
if( img0.empty() )
{
cout << "Couldn'g open image " << filename << ". Usage:
watershed <image_name>\n";
return 0;
}
help();
namedWindow( "image", 1 );
img0.copyTo(img);
cvtColor(img, markerMask, COLOR_BGR2GRAY);
cvtColor(markerMask, imgGray, COLOR_GRAY2BGR);
markerMask = Scalar::all(0);
imshow( "image", img );
setMouseCallback( "image", onMouse, 0 );
for(;;)
{
char c = (char)waitKey(0);
if( c == 27 )
break;
if( c == 'r' )
{
markerMask = Scalar::all(0);
img0.copyTo(img);
imshow( "image", img );
}
if( c == 'w' || c == ' ' )
{
int i, j, compCount = 0;
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
findContours(markerMask, contours, hierarchy, RETR_
CCOMP, CHAIN_APPROX_SIMPLE);
if( contours.empty() )
continue;
Mat markers(markerMask.size(), CV_32S);
markers = Scalar::all(0);
int idx = 0;
for( ; idx >= 0; idx = hierarchy[idx][0], compCount++ )
drawContours(markers, contours, idx, Scalar::all(comp-
Count+1), -1, 8, hierarchy, INT_MAX);
if( compCount == 0 )
continue;
vector<Vec3b> colorTab;
for( i = 0; i < compCount; i++ )
{
int b = theRNG().uniform(0, 255);
int g = theRNG().uniform(0, 255);
int r = theRNG().uniform(0, 255);
colorTab.push_back(Vec3b((uchar)b, (uchar)g, (uchar)r));
}
double t = (double)getTickCount();
watershed( img0, markers );
t = (double)getTickCount() - t;
printf( "execution time = %gms\n", t*1000./getTickFrequ-
ency() );
Mat wshed(markers.size(), CV_8UC3);
// paint the watershed image
for( i = 0; i < markers.rows; i++ )
for( j = 0; j < markers.cols; j++ )
{
int index = markers.at<int>(i,j);
if( index == -1 )
wshed.at<Vec3b>(i,j) = Vec3b(255,255,255);
else if( index <= 0 || index > compCount )
wshed.at<Vec3b>(i,j) = Vec3b(0,0,0);
else
wshed.at<Vec3b>(i,j) = colorTab[index - 1];
}
wshed = wshed*0.5 + imgGray*0.5;
imshow( "watershed transform", wshed );
}
}
return 0;
}
}
```

Wnioski

Zastosowany algorytm pozwala na wyodrębnienie poszczególnych obiektów. Niestety przy zdjęciach termowizyjnych obszary przedstawiające obiekty o tej samej temperaturze mają identyczną barwę. Należałoby więc połączyć segmentację opartą na badaniu konturu razem z zastosowanym algorytmem. Celem niniejszej publikacji było jedynie pokazanie zastosowań bibliotek OpenCV.

Literatura

[1]. Pulli, Kari; Baksheev, Anatoly; Kornyakov, Kirill; Eruhimov, Victor (1 April 2012). "Realtime Computer Vision with OpenCV". Queue. pp. 40:40–40:56. doi:10.1145/2181796.2206309 CHENG, H. D. et al. Color image segmentation: advances and prospects. Pattern Recognition, v. 34, p. 2259-228, 2001.

[2]. Kaufman, A. and Shimony, E., '3D Scan-Conversion Algorithms for Voxel-Based Graphics', Proc. ACM Workshop on Interactive 3D Graphics, Chapel Hill, NC, October 1986, 45-76.

[3]. FR patent 841335, Valensi, Georges, "Procédé de télévision en couleurs", published 1939-05-17, issued 1939-02-06 [4].THEODORIDIS, S.; K, K. Pattern Recognition. [S.l.]: Elsevier Inc, 2009.

[5]. PAL, N. R.; PAL, S. A review on image segmentation techniques. Pattern Recognition, v. 26, p. 1277-1294, 1993.

[6]. C. Rother, V. Kolmogorov, and A. Blake.“GrabCut”—Interactive foreground extraction using iterated graph cuts. In SIGGRAPH, pages 309–314, 2004.

[7]. A. K. Sinop and L. Grady. A seeded image segmentation framework unifying graph cuts and random walker

which yields a new algorithm. In ICCV, 2007
Strony internetowe:
[8]. https://docs.opencv.org/3.3.1/d8/da9/watershed_8cpp-
-example.html.
[9]. https://github.com/clickteam-plugin/OpenCV/blob/
master/Extension/Main/Voxel.h-
[10]. www.opencv.org
[11]. https://docs.opencv.org/2.4/doc/tutorials/introduc-
tion/linux_eclipse/linux_eclipse.html
[12]. http://www.eclipse.org
[13]. http://www.intel.com/technology/computing/
opencv/
[14]. https://docs.opencv.org/2.4.13.2/modules/core/doc/
basic_structures.html
[15]. https://docs.opencv.org/2.4/modules/imgproc/doc/
miscellaneous_transformations.html