Szkoła Główna Gospodarstwa Wiejskiego

w Warszawie

Wydział Zastosowań Informatyki i Matematyki

Sylwester Turski

167497

Inteligentny algorytm do rozpoznawania i zliczania miejsc parkingowych na podstawie serii obrazów z kamery

Intelligent algorithm to recognize and count the parking spaces on the basis of a series of images from the camera

Praca dyplomowa inżynierska

na kierunku informatyka

Praca wykonana pod kierunkiem

Dra Pawła Hosera

Katedra Zastosowań Informatyki

Warszawa, 2017 rok

**Oświadczenie promotora pracy**

Oświadczam, że niniejsza praca\*/wskazane przez autora rozdziały pracy dyplomowej przygotowanej zespołowo\* została/zostały\* przygotowana pod moim kierunkiem   
i stwierdzam, że spełnia\*/spełniają\* warunki do przedstawienia tej pracy w postępowaniu   
o nadanie tytułu zawodowego.

Data .................................... Podpis promotora pracy ...................................................

**Oświadczenie autora pracy**

Świadom odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że niniejsza praca dyplomowazostała napisanaprzeze mnie samodzielnie i nie zawieratreści uzyskanych   
w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności ustawą z dnia   
4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. Nr 90 poz. 631 z późn. zm.)

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data ..................................... Podpis autora pracy .............................................

**Streszczenie**

**Inteligentny algorytm do rozpoznawania i zliczania miejsc parkingowych na podstawie serii obrazów z kamery**

Streszczenie pracy - do 1000 znaków (12)

Słowa kluczowe - do 6 (12)

**Summary** (12 bold)

**Title** (12 bold)

Summary - 1000 words max (12)

Keywords – 6 max (12)

Spis Treści

[1. Wstęp 8](#_Toc471390539)

[2. Cel i zakres pracy 8](#_Toc471390540)

[3. Przegląd piśmiennictwa 8](#_Toc471390541)

[4. Projekt rozwiązania 8](#_Toc471390542)

[4.1. Biblioteki i technologie użyte podczas pisania algorytmu 8](#_Toc471390543)

[4.2. Badane cechy obrazu 9](#_Toc471390544)

[5. Omówienie i dyskusja wyników 14](#_Toc471390545)

[6. Spis piśmiennictwa 14](#_Toc471390546)

# Wstęp

# Cel i zakres pracy

Celem pracy jest stworzenie algorytmu, który będzie w stanie sklasyfikować miejsce parkingowe zaznaczone na zdjęciu jako wolne lub zajęte. Do algorytmu będzie dostarczany również zbiór uczący, składający się ze zdjęć parkingu wraz z oznaczeniami konturu gdzie znajdują się miejsca parkingowe i czy są one zajęte czy wolne.

Dodatkowo w zakres pracy wchodzi przygotowanie makiety parkingu i przygotowanie testowych zdjęć. Przygotowanie programu ułatwiającego oznaczanie konturów miejsc parkingowych i ich statusu. Przygotowanie programu testującego skuteczność algorytmu.

# Przegląd piśmiennictwa

# Projekt rozwiązania

## Biblioteki i technologie użyte podczas pisania algorytmu

OpenCV (Open Source Computer Vision) – Popularna biblioteka funkcji do rozpoznawania obrazów w czasie rzeczywistym. Posiada między innymi funkcje do manipulacji obrazami, wykrywania cech obrazu i uczenia maszynowego. W 1999 roku projekt OpenCV został zainicjalizowany przez firmę Intel. Biblioteka jest napisana w języku C++. Jako wrapper w języku C# wybrałem bibliotekę OpenCvSharp. API tej biblioteki jest bardzo zbliżone do oryginału, dodatkowo posiada fluent API.

WPF (Windows Presentation Foundation) – Framework do tworzenia interfejsu użytkownika (ang. user interface) dla platformy .NET, stworzony przez firmę Microsoft. WPF kładzie nacisk na grafikę wektorową, dzięki której większość kontrolek można skalować bez utraty jakości, czy pixelizacji. Wybór na tą technologię zapadł ponieważ można w łatwy sposób projektować wygląd aplikacji jednocześnie zachowując pełną kontrolę nad zachowaniem aplikacji podczas zmiany wielkości okna. Rozważaną alternatywą do tej technologii były wbudowane w bibliotekę OpenCV funkcję do tworzenie interfejsu użytkownika. Jednak nie pozwalały one zarządzać położeniem i wielkością kontrolek, przez co ostateczny wybór padł na technologię WPF

## Badane cechy obrazu

Organoleptycznie można zauważyć że samochody od podłoża odróżnia kolor karoserii, więc pierwszą badaną cechą było badanie współczynnika pixeli z saturacją >100 do całości obszaru miejsca parkingowego. Na Listingu x,y,z pokazano sposób obliczania tej cechy. W bibliotece OpenCV zdarzają się pixele o kolorze zbliżonym do czarnego które jednak posiadają wysoką saturację, dzieje się tak dlatego że przestrzeń kolorów HSV w bibliotece jest prezentowana jako walec, dlatego w programie skaluje saturację z wartością co jest przedstawione na Listingu x

public static int CountSaturationPixels(Contour contour, Mat src)

{

var rect = GetContourRect(contour, src.Height, src.Width);

var mask = GetMask(contour, src.Size(), color: Scalar.White, background: Scalar.Black)

.Clone(rect)

.CvtColor(ColorConversionCodes.BGR2GRAY);

return src

.Clone(rect)

.CvtColor(ColorConversionCodes.BGR2HSV)

.ScaleSaturationWithValue() // returns only saturation layer

.BitwiseAnd(mask)

.Threshold(100, 255, ThresholdTypes.Binary)

.CountNonZero();

}

Listing Obliczanie liczności nasyconych pixeli

public static Mat ScaleSaturationWithValue(this Mat src)

{

var mats = src.Split();

return mats[1].Mul(mats[2], 1.0/255);

}

Listing Skalowanie nasycenia z wartością koloru

public static int CountMaskArea(Contour contour, Mat src)

{

var rect = GetContourRect(contour, src.Height, src.Width);

return GetMask(contour, src.Size(), color: Scalar.White, background: Scalar.Black)

.Clone(rect)

.CvtColor(ColorConversionCodes.BGR2GRAY)

.Threshold(200, 255, ThresholdTypes.Binary)

.CountNonZero();

}

Listing Obliczanie liczności pixeli w masce

public float SaturatedPixelsRatio => (float) SaturatedPixels/MaskPixels;

Listing Obliczanie współczynnika nasyconych pixeli względem obszaru

Rysunek x przedstawia wyniki walidacji sprawdzianem krzyżowym i [n-1]



Rysunek 11Wyniki testów klasyfikacji z cechą współczynnika pixeli nasyconych

Cecha nie jest skuteczna przy wykrywaniu samochodów o kolorze karoserii który posiada niską saturację (np. białym, czarnym, szarym) dając *false negative*. Również szum RGB generowany przez kamerę przy słabym oświetleniu może generować wyniki *false positive*.

Kolejną cechą którą można zauważyć organoleptycznie są krawędzie. Na obrazie na którym zastawano wykrywanie krawędzi np. metodą *Cannego*, można zauważyć że obszary na których znajdują się samochody dają widocznie więcej krawędzi od podłoża. Na podstawie tej obserwacji wyliczam stosunek pixeli z krawędziami do całości obszaru. Kod obliczający tą cechę prezentuję na Listingach X,Y. Do wykrywania krawędzi używana jest bezparametrowej implementacji metody *Cannego*, która automatycznie dobiera progi, na podstawie średniej wartości koloru, dla obrazu w skali szarości.

public static int CountEdgePixels(Contour contour, Mat src)

{

var rect = GetContourRect(contour, src.Height,src.Width);

var mask = GetMask(contour, src.Size(), color: Scalar.White, background: Scalar.Black)

.Clone(rect)

.CvtColor(ColorConversionCodes.BGR2GRAY);

return src

.Clone(rect)

.DetectEdges()

.BitwiseAnd(mask)

.Threshold(100, 255, ThresholdTypes.Binary)

.CountNonZero();

}

Listing Obliczanie liczności pixeli z krawędziami

public float EdgePixelsRatio => (float) EdgePixels/MaskPixels;

Listing Obliczanie stosunku pixeli z krawędziami do całego obszaru

public static Mat DetectEdges(this Mat src, double sigma = 0.33)

{

var graySrc = src.CvtColor(ColorConversionCodes.BGR2GRAY);

Cv2.MeanStdDev(graySrc, out var meanScalar, out var stddevScalar);

var mean = meanScalar[0];

var lower = (int) Math.Max(0, (1.0 - sigma) \* mean);

var upper = (int) Math.Min(255, (1.0 + sigma) \* mean);

return graySrc.Canny(lower, upper);

}

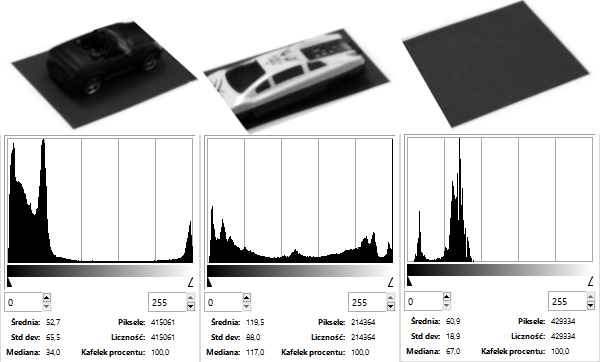
Listing Bezparametrowa detekcja krawędzi Cannego

Wyniki sprawdzianu klasyfikacji zamieszczam na Rysunku x



Rysunek Wyniki sprawdzianu walidacji dla cech SaturatedPixelsRatio i EdgePixelsRatio

Kolejne badane cechy to badania histogramu dla obrazka w skali szarości. Na Rysunku X można łatwo zauważyć że badane histogramy różnią się rozkładem wartości na histogramie. Wyliczana jest wartość oczekiwana i odchylenie standardowe dla obrazu w skali szarości.



public static ((float mean, float stddev) saturation, (float mean, float stddev) value) GetHSVColorStats(

Contour contour, Mat src)

{

var rect = GetContourRect(contour, src.Height, src.Width);

var mask = GetMask(contour, src.Size(), color: Scalar.White, background: Scalar.Black)

.Clone(rect)

.CvtColor(ColorConversionCodes.BGR2GRAY);

var layers = src.Clone(rect)

.CvtColor(ColorConversionCodes.BGR2HSV)

.Split();

(float mean, float stddev) LocalMeanStdDev(Mat area)

{

Cv2.MeanStdDev(area, out var scalarMean, out var scalarStddev, mask);

var mean = (float) (scalarMean[0] / 255);

var stddev = (float) (scalarStddev[0] / 255);

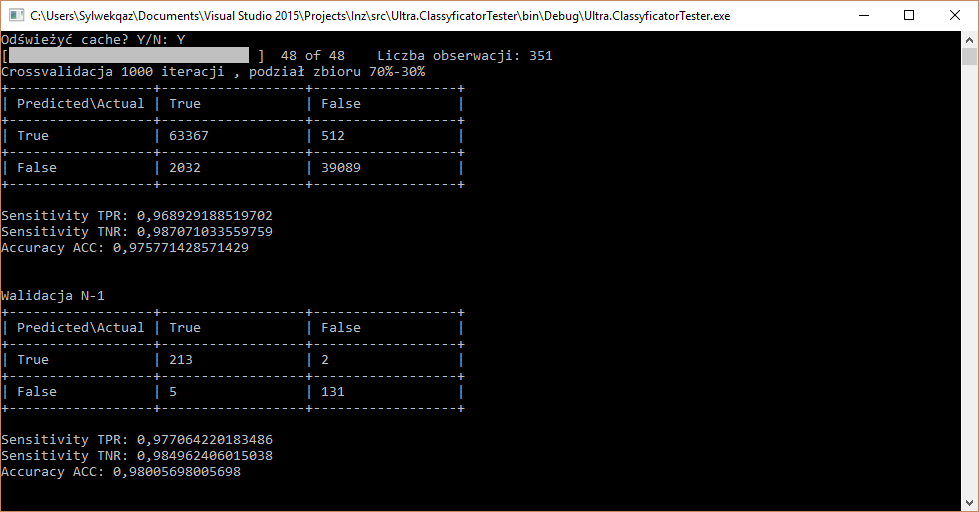
return (mean, stddev);

}

return (LocalMeanStdDev(layers[1])/\*saturation layer\*/, LocalMeanStdDev(layers[2])/\* value layer\*/);

}

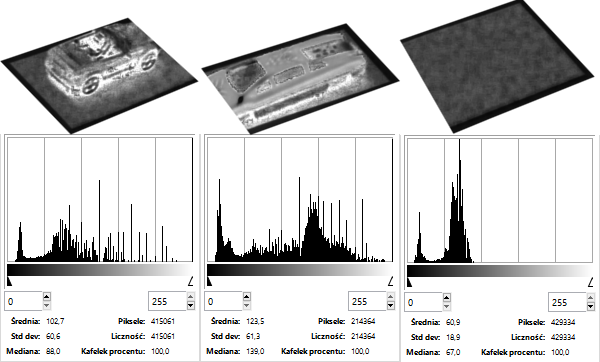
Listing Obliczanie statystyk średniej i odchylenia standardowego dla saturacji i wartości



Rysunek Wynikisprawdzianu klasyfikacji dla cech EdgePixelsRatio SaturatedPixelsRatio ValueMean ValueStddev

Sprawdzian wyników klasyfikacji zamieszczam na Rysunku x.

Kolejnymi badanymi cechami jest badanie histogramu dla warstwy saturacji w przestrzeni HSV, dla zadanego obszaru zdjęcia. Wyliczana jest wartość średnia i odchylenie standardowe.





# Omówienie i dyskusja wyników

# Spis piśmiennictwa

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW   
w tym w Archiwum Prac Dyplomowych SGGW

.................................................................

*(czytelny podpis autora pracy)*