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](#_Toc471583696)

[2. Cel i zakres pracy 8](#_Toc471583697)

[3. Przegląd piśmiennictwa 8](#_Toc471583698)

[4. Projekt rozwiązania 8](#_Toc471583699)

[4.1. Biblioteki i technologie użyte podczas pisania algorytmu 8](#_Toc471583700)

[4.2. Makieta parkingu 9](#_Toc471583701)

[4.3. Klasyfikator 10](#_Toc471583702)

[4.4. Badane cechy obrazu 11](#_Toc471583703)

[5. Omówienie i dyskusja wyników 17](#_Toc471583704)

[6. Literatura 17](#_Toc471583705)

[7. Spis obrazków 17](#_Toc471583706)

[8. Spis listingów 18](#_Toc471583707)

# 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

## Makieta parkingu

Pierwsza wersja makiety parkingu była zrobiona z czarnej bibuły zawiniętej i oklejonej na sztywnej podkładce. Pasy miejsc parkingowych zostały narysowane korektorem w taśmie. Faktura bibuły miała odwzorowywać fakturę asfaltu. Jako modelu samochodów zostały użyte popularne zabawkowe resoraki w kilku kolorach, i jeden resorak w kolorze czarnym, który z założenia miał zlewać się z kolorem podłoża i stanowić przypadek pesymistyczny. Zdjęcia robione były przy użyciu kamery internetowej w rozdzielczości 640x480, model ustawiany był pod różnymi obrotami względem kamery. Zdjęcia były tak dobierane aby były przypadki optymistyczne, gdzie samochód pozostaje w swoim miejscu parkingowym i takie gdzie kawałek samochodu zasłania kawałek miejsca parkingowego obok.

Takie podejście powodowało problemy. Faktura bibuły była wyraźnie widoczna ze względy na bliską odległość kamery od modelu, pomięta faktura bibuły utrudniała również rysowanie pasów korektorem w taśmie. Słabe oświetlenie sceny powodowało iż samochody o kolorze czarnym były słabo widoczne dla ludzkiego oka na obrazach. Doświetlanie sceny powodowało że samochody o kolorze czarnym były zaczynały być widoczne, a algorytm wykrywania krawędzi zaczynał wykrywać krawędzie na samochodzie, Jednakże algorytm wykrywania krawędzi znacząco więcej wykrywał pomarszczoną fakturę bibuły, a samochody kolorowe zaczynały być prześwietlone. Kamera przy słabej jakości oświetlenia zaczęła generować szum cyfrowy. Całość tych niedogodności wymagała bardzo precyzyjnego ustawienia oświetlenia sceny.

Druga wersja makiety powstała na brystolu o jednolitym ciemnoszarym kolorze. Z narysowanymi pasami korektorem w taśmie, zestaw samochodów resorków pozostał ten sam. Zdjęcia zostały wykonane aparatem telefonu komórkowego o rozdzielczości 3264x2448. Zdjęcia zostały zrobione pod różnymi kontami względem modelu, w przeciwieństwie do poprzedniego zdjęcia mają również różne nachylenie względem modelu. Są zdjęcia robione z góry, gdzie samochody idealnie się wpasowują w miejsca parkingowe, jak i zdjęcia robione pod niskim kontem, gdzie samochody zasłaniają oboczne miejsca parkingowe.

Drugie drugi model poprawiał znacząco jakość danych. Jednolita faktura brystolu eliminowała problem wykrywanych krawędzi. Dzięki sztywności brystolu nie potrzebna dodatkowych elementów usztywniających model, również rysowanie pasów korektorem w taśmie nie powodowało problemów. Dzięki zastosowaniu kamery o lepszych parametrach i lepszego źródła oświetlenia pomieszczenia, scena nie wymagała dodatkowego doświetlenia, a samochody o kolorze czarnym są wyraźnie widoczne na zdjęciach. Nie występuje również problem szumu cyfrowego.

## 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. Sposób obliczania tej cechy przedstawia Listing 2, Listing 4 i Listing 5. 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 przedstawia Listing 3.

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 2 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 3 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 4 Obliczanie liczności pixeli w masce

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

Listing 5 Obliczanie współczynnika nasyconych pixeli względem obszaru

Rysunek 2 przedstawia wyniki walidacji sprawdzianem krzyżowym i [n-1]



Rysunek 2 Wyniki 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 Listing 6 i Listing 7. Do wykrywania krawędzi używana jest bezparametrowej implementacji metody *Cannego*, która automatycznie dobiera progi, na podstawie średniej wartości koloru, dla obrazu w skali szarości (Listing 8).

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 6 Obliczanie liczności pixeli z krawędziami

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

Listing 7 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);

}

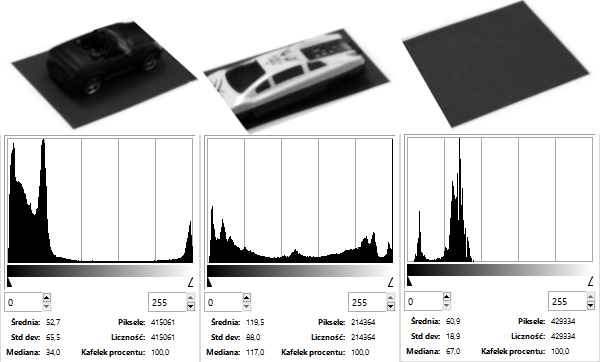
Listing 8 Bezparametrowa detekcja krawędzi Cannego

Rysunek 3 Pokazuje wyniki sprawdzianu klasyfikacji.



Rysunek 3 Wyniki sprawdzianu walidacji dla cech SaturatedPixelsRatio i EdgePixelsRatio

Kolejne badane cechy to badania histogramu dla obrazka w skali szarości. Rysunek 5 przedstawia histogramy dla pustego miejsca parkingowego i zajętego przez samochód o kolorze żółtym i samochód o kolorze czarnym. Łatwo zauważyć że histogram niezajętego miejsca parkingowego ma bardzo małe odchylenie standardowe. Wyliczana jest wartość oczekiwana i odchylenie standardowe dla obrazu w skali szarości.



Rysunek 4 Porównanie histogramów skali szarości dla pustego miejsca parkingowego, miejsca zajętego przez samochód kolorowy i czarny

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);

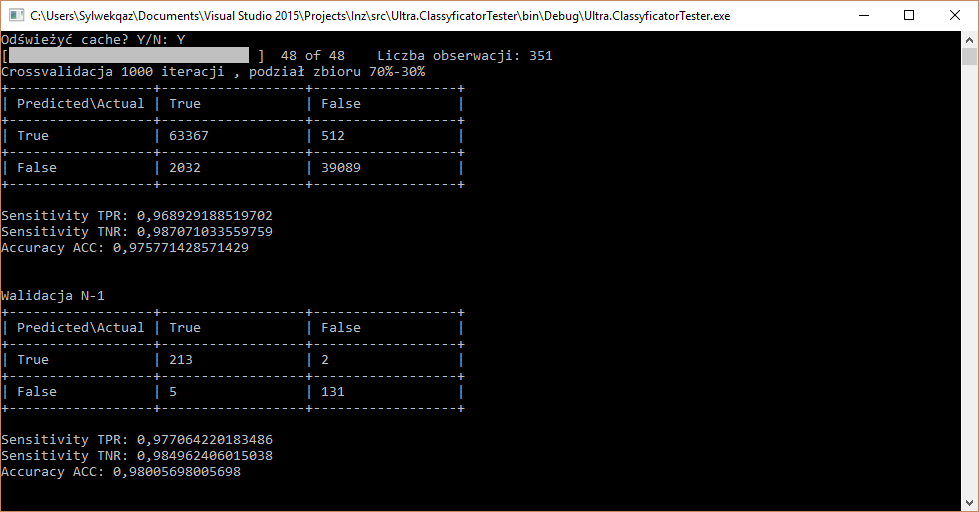
return (mean, stddev);

}

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

}

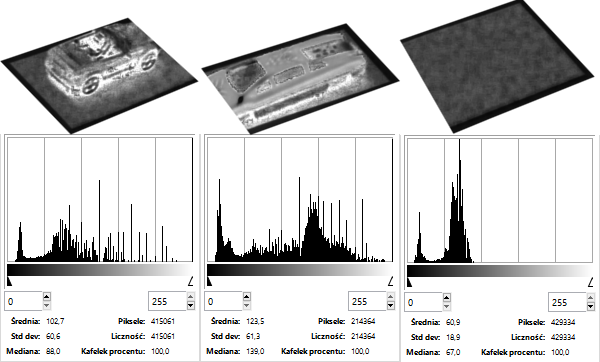
Listing 9 Obliczanie statystyk średniej i odchylenia standardowego dla saturacji i wartości



Rysunek 5 Wyniki sprawdzianu klasyfikacji dla cech EdgePixelsRatio SaturatedPixelsRatio ValueMean ValueStddev

Rysunek 5 Przedstawia wyniki sprawdzianu klasyfikacji.

Kolejnymi badanymi cechami są cechy histogramu dla warstwy saturacji w przestrzeni HSV (walcowatej), dla zadanego obszaru zdjęcia. Na Rysunek 6 widać że miejsca zajęte nie tylko mają większe odchylenie standardowe, ale również średnia jest wyższa. Do klasyfikatora wyliczana jest wartość średnia i odchylenie standardowe.



Rysunek 6 Porównanie histogramów saturacji dla pustego miejsca parkingowego, miejsca zajętego przez samochód kolorowy i czarny



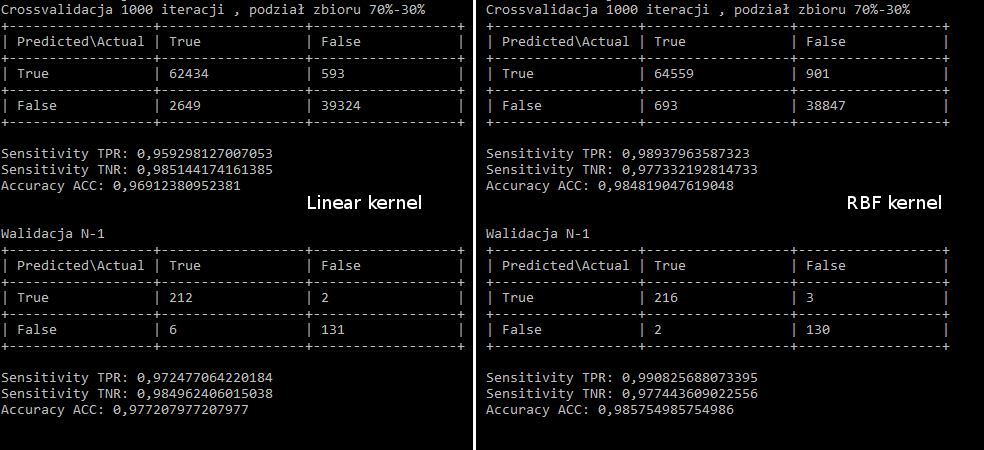
Rysunek 7 Wyniki sprawdzianu klasyfikacji z cechami SaturationRatio, EdgeRatio, ValueMean, ValueStdDev, SaturationMean, SaturationStdDev

Rysunek 7 Przedstawia wyniki sprawdzianu klasyfikatora z dodanymi cechami wyliczanymi z histogramu saturacji.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Współczynnik krawędzi |  |  |  |  |  |
|  | Współczynnik saturacji |  |  |  |  |
|  |  | Średnia saturacja |  |  |  |
|  |  |  | Odchylenie standardowe saturacji |  |  |
|  |  |  |  | Średnia wartość |  |
|  |  |  |  |  | Odchylenie standardowe wartości |

## Klasyfikator

SVM (z ang. support vector machine -maszyna wektorów nośnych) jest bla bla bla. Dla zwiększenia poprawności klasyfikacji została dodana funkcja jądra RBF (ang. radial basis function - Radialna funkcja bazowa). Jądro RBF przy użyciu gaussowskiej funkcji radialnej, dodaje dodatkowe wymiary w przestrzeni cech. Dzięki czemu pozwala na nieliniową separację zbiorów których nie da się rozdzielić hiperpłaszczyzną w klasycznym podejściu. Dzięki podejściu nieliniowemu dostajemy kilkuprocentowy zysk w poprawności klasyfikacji (Rysunek 1).



Rysunek 1Różnica poprawności klasyfikacji pomiędzy jądrem RBF i liniowymmm jądrem

public static SVMClassifier Create(List<ImageFeatures> trainingData)

{

var svm = SVM.Create();

svm.Type = SVM.Types.CSvc;

svm.KernelType = SVM.KernelTypes.Rbf;

svm.TermCriteria = TermCriteria.Both(maxCount: 1000, epsilon: 0.000001);

svm.Gamma = 100.0;

svm.C = 1.0;

svm.Train(trainingData.ToTrainingMat(), SampleTypes.RowSample, trainingData.ToResponseMat());

return new SVMClassifier(svm);

}

Listing 1 Konfiguracja klasyfikatora SVM

# Omówienie i dyskusja wyników

# Literatura

# Spis obrazków

[Rysunek 1Różnica poprawności klasyfikacji pomiędzy jądrem RBF i liniowymmm jądrem 10](#_Toc471583708)

[Rysunek 2 Wyniki testów klasyfikacji z cechą współczynnika pixeli nasyconych 12](file:///C:\Users\Sylwekqaz\Documents\Visual%20Studio%202015\Projects\Inz\Turski_Sylwester_INZ.docx#_Toc471583709)

[Rysunek 3 Wyniki sprawdzianu walidacji dla cech SaturatedPixelsRatio i EdgePixelsRatio 13](#_Toc471583710)

[Rysunek 4 Porównanie histogramów skali szarości dla pustego miejsca parkingowego, miejsca zajętego przez samochód kolorowy i czarny 14](#_Toc471583711)

[Rysunek 5 Wyniki sprawdzianu klasyfikacji dla cech EdgePixelsRatio SaturatedPixelsRatio ValueMean ValueStddev 15](#_Toc471583712)

[Rysunek 6 Porównanie histogramów saturacji dla pustego miejsca parkingowego, miejsca zajętego przez samochód kolorowy i czarny 16](#_Toc471583713)

[Rysunek 7 Wyniki sprawdzianu klasyfikacji z cechami SaturationRatio, EdgeRatio, ValueMean, ValueStdDev, SaturationMean, SaturationStdDev 16](#_Toc471583714)

# Spis listingów

[Listing 1 Konfiguracja klasyfikatora SVM 11](#_Toc471583715)

[Listing 2 Obliczanie liczności nasyconych pixeli 11](#_Toc471583716)

[Listing 3 Skalowanie nasycenia z wartością koloru 11](#_Toc471583717)

[Listing 4 Obliczanie liczności pixeli w masce 12](#_Toc471583718)

[Listing 5 Obliczanie współczynnika nasyconych pixeli względem obszaru 12](#_Toc471583719)

[Listing 6 Obliczanie liczności pixeli z krawędziami 13](#_Toc471583720)

[Listing 7 Obliczanie stosunku pixeli z krawędziami do całego obszaru 13](#_Toc471583721)

[Listing 8 Bezparametrowa detekcja krawędzi Cannego 13](#_Toc471583722)

[Listing 9 Obliczanie statystyk średniej i odchylenia standardowego dla saturacji i wartości 15](#_Toc471583723)

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

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

*(czytelny podpis autora pracy)*