ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA STROJNÍ

BAKALÁŘSKÁ PRÁCE

NÁVRH, TVORBA A OPTIMALIZACE MASKY PRO HLEDÁNÍ VZORŮ VE STROJOVÉM VIDĚNÍ

2022

PETR
KOHL

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kohl** Jméno: **Petr** Osobní číslo: **482387**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídicí techniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Návrh, tvorba a optimalizace masky pro hledání vzorů ve strojovém vidění

Název bakalářské práce anglicky:

Proposal, creation and mask optimization for searching patterns in machine vision

Pokyny pro vypracování:

- Rešerše problematiky hledání vzorů v rozpoznávání obrazu
- Příprava hledaných vzorů pomocí masek a jejich optimalizace
- Experimentální srovnání představených postupů

Seznam doporučené literatury:

1) BHUYAN, Manas Kamal. Computer vision and image processing fundamentals and applications: fundamentals and applications. Boca Raton: CRC Press/Taylor & Francis Group, [2020]. ISBN 978-0-8153-7084-0
2) RUSS, John C, a F. Brent NEAL. The image processing handbook. Seventh Edition. Boca Raton: CRC Press, [2017]. ISBN 978-1-138-74749-4

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Matouš Cejnek, Ph.D. U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **29.04.2022** Termín odevzdání bakalářské práce: **09.06.2022**

Platnost zadání bakalářské práce: _____

Ing. Matouš Cejnek, Ph.D.		doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

5.5.2022 _____

Datum převzetí zadání Podpis studenta

PROHLÁŠENÍ

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně s tím, že její výsledky mohou být dále použity podle uvážení vedoucího bakalářské práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků bakalářské práce nebo její podstatné části, pokud budu uveden jako její spoluautor.

Dne 8. 5. 2022 Petr Kohl

ABSTRAKT

Tato bakalářská práce se zabývá návrhem, tvorbou a optimalizací masky pro hledání vzorů ve strojovém vidění. Realizace tohoto úkonu je navržena v programovacím jazyku Python za pomoci knihovny OpenCV a dalších. Cílem této práce je osvětlit neznalému člověku procesy návrhu, tvorby, optimalizace masek, a to jak v psané formě programu, tak vizuální ve formě obrázků. Funkce této práce má být vzdělávací a sloužit jako návod.

KLÍČOVÁ SLOVA

Python, OpenCV, detekce, strojové vidění, maska, rozpoznávání předmětu, vzor, zpracování obrazu, programovací jazyk

ABSTRACT

This bachelor thesis deals with the design, creation and optimization of a mask for searching patterns in machine vision. The implementation of this task is designed in the Python programming language with the help of the OpenCV library and others. The aim of this work is to explain to the unfamiliar person the processes of design, creation, optimization of masks, both in the written form of the program and visual in the form of images. The function of this work is to be educational and to serve as a guide.

KEY WORDS

Python, OpenCV, detection, machine vision, mask, object recognition, pattern, image-processing, programming language

1 OBSAH

2 POUŽITÉ ZKRATKY A SYMBOLY

RGB.....Red, Blue, Green

HD .....High Definition

AI.....Artificial Intelligence – Umělá Inteligence

ML .....Machine Learning – Strojové Učení

IDE .....Integrated Development Environment – Integrované Vývojové Prostředí

GIT .....Global Information Tracker – Distribuovaný systém řízení

API .....Application Programming Interface – Aplikační programovací rozhraní

SPZ .....Státní poznávací značka

3 ÚVOD

Strojové vidění. Pojem, který v dnešní době spolu s automatizací a umělou inteligencí zaznívá čím dál častěji a hlasitěji. Dříve většinově zastoupený pouze v průmyslovém odvětví, dnes součástí každodenního života.

Vzhledem k velkému posunu kvality obrazových snímačů, jejich výstupů a enormnímu nárůstu procesních možností je strojové vidění velice efektivní a přesné. Jako příklady bych si dovolil uvést bezpečnostní systém face ID od společnosti Apple Inc. založený na několika způsobech rozpoznání obličeje a jeho částí. Sám výrobce uvádí pravděpodobnost na selhání rozpoznání tváře jedna ku milionu. [29] Jako druhý příklad bych rád uvedl čínský systém pro rozpoznání identity, kdy je identita občana rozpoznávána z každé veřejné kamery v reálném čase. [37]

Cílem této práce je vyobrazit způsoby návrhu, tvorby masek strojového vidění a jejich optimalizace pro různé aplikace.

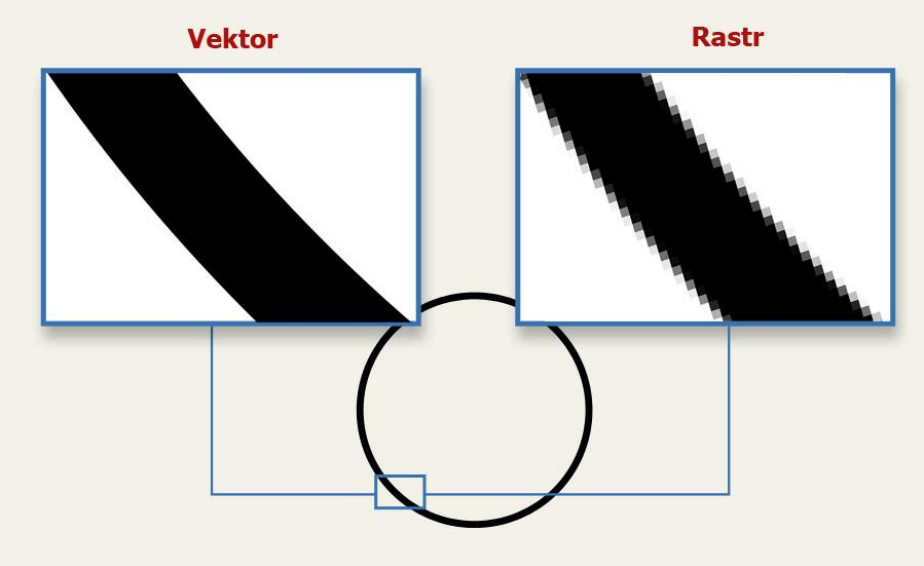
Tato práce má zároveň sloužit jako manuál pro začínající programátory strojového vidění.

4 VSTUPY A JEJICH OPTIMALIZACE

Pro co nejvyšší úspěšnost rozpoznání objektu hraje velkou roli kvalita vstupního obrazu. Tato závislost je přímo úměrná, tzn. čím kvalitnější vstup, tím přesnější výstup.

4.1 Obraz

Zpracování obrazu zahrnuje mnoho různých technik. Ty se mezi sebou mohou kombinovat za účelem získání co nejlepšího výstupu. Výstup je možné formulovat buď jako obraz nebo jako data popisující vlastnosti tohoto obrazu. S tímto výstupem je možné nadále pracovat a podrobovat ho různým analýzám podle potřeb zadání. Digitální obraz se rozděluje podle typu zápisu na dva druhy, a to vektorový a rastrový. Rozdíl mezi nimi je vyobrazen na *Obr. 1*. [1]



Obr. 1, Vektorový a rastrový zápis obrazu [14]

4.1.1 Vektorový obraz

Vektorový obraz je složen z velkého množství geometrických objektů, ty jsou předepsány funkcemi (F (x, y), kde x a y jsou souřadnice v 2D poli). Díky takovémuto způsobu zápisu je možné obraz libovolně zmenšovat a zvětšovat bez jakékoliv změny kvality obrazu. Jelikož je takovýto obraz ve své podstatě jen soubor funkcí, je v celku nenáročný na kapacitu paměti. [1]

4.1.2 Rastrový obraz

Rastrový obraz funguje na bázi jednotlivých bodů. Obraz je rozdělen do matice, kdy každé jedno pole této matice je označováno jako pixel. Jednotlivé pixely mají pevně určenou polohu a barvu. Kvalita takového obrazu je přímo určená počtem pixelů. Z toho plyne, že je paměťová náročnost přímo úměrná obrazové kvalitě (relativní velikosti obrazu). Jednoduchým relativním zvětšením rastrového obrazu se zvětšují i samotné pixely, tím se ztrácí ostrost obrazu. Tu je posléze možné zlepšit některými algoritmy obrazového zpracování. Rastrový obraz můžeme dále rozdělit dle barevného modelu. [1]

4.1.2.1 Stupně šedi

Obraz je jednokanálový. Je složen pouze z odstínů černé a bílé.

4.1.2.2 RGB

Z angličtiny Red, Blue, Green. Má tři kanály a jak už název napovídá, je to kanál červené, modré a zelené barvy. Kombinací odstínů těchto tří barev vznikají výsledné barvy obrazu. Jedná se o nejpoužívanější barevný model pro digitální obraz.

4.1.2.3 Další barevné modely

CMYK – běžný při tisku fotografií

HSV – využití při editaci fotografií

YUV – použití v televizním vysílání

4.2 Objektiv

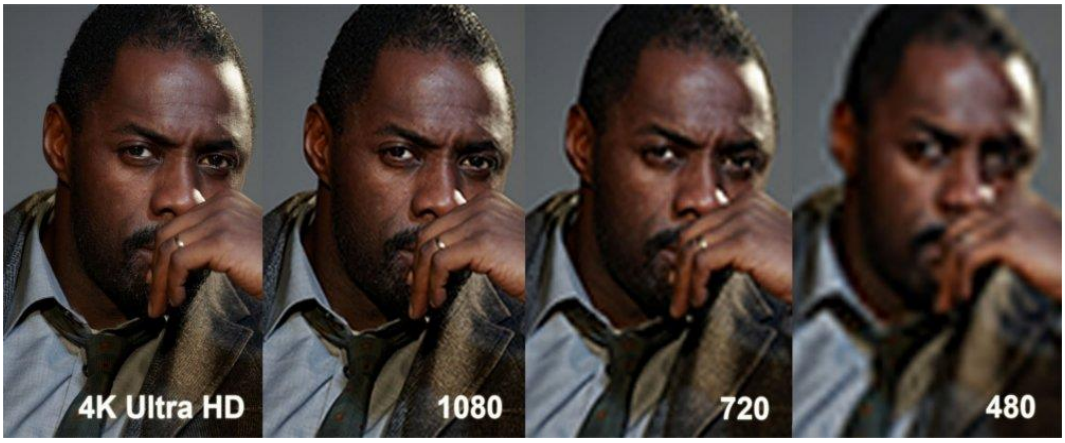
Pro potřeby každé aplikace bude potřeba lehce jiný objektiv. Nejdůležitější proměnou pro tuto aplikaci je ohnisková vzdálenost. Ta se pohybuje v rámci jednotek až stovek milimetrů. Normální objektiv, jehož zobrazení odpovídá zornému úhlu lidského oka, má ohniskovou vzdálenost 50 mm. Pro účely strojového vidění jsou vhodnější spíše objektivy s vyšší ohniskovou vzdáleností. Důvodem je nižší deformace perspektivy. [2]

4.3 Rozlišení

V dnešní době jsou nejrozšířenější čtyři rozlišení (níže uvedení číselní hodnoty jsou uvedeny v pixelech). [8]

- 1280 x 720 – HD
- 1920 x 1080 – Full HD
- 2560 x 1440 – QHD
- 3840 x 2160 – 4K

Vyšší rozlišení přináší lepší výsledky rozpoznání objektů, avšak klade si větší nároky na paměť a pořizovací cenu zařízení. Proto je zde důležité najít správnou rovnováhu mezi cenou a kvalitou zařízení. Pro představu rozdílů mezi rozlišeními je uveden *Obr. 2.*



Obr. 2, Vizuální srovnání rozlišení [15]

4.4 Formáty

4.4.1 Joint Photographic Experts Group (JPEG)

Také označovaný JPG je ztrátový formát, ale i přesto si drží vcelku slušnou kvalitu s dobrou kompresí. Tento formát nepodporuje průhlednost, animace a není vhodný ani pro vektorovou grafiku. [1] [10]

4.4.2 Portable Network Graphics (PNG)

Formát PNG nahrazuje zastaralý formát GIF. Využívá bezztrátové komprese a nejčastěji se vyskytuje ve webové grafice. Tento formát se dále dělí na 8bitový (obraz se skládá z 256barev) a 24bitový (obraz se skládá z 16,7milionu barev). Nepodporuje animace, ale dokáže pracovat s průhledností. [1] [10]

4.4.3 Tag Image File Format (TIFF)

Bezztrátový 24bitový formát, jež se nejvíce využívá na fotografie, a to za účelem kvalitativních možností změny rozlišení. TIFF je vcelku náročný na paměť. [1] [10]

4.4.4 RAW

Název pochází z anglického slova raw („syrový“ ve smyslu neupravený). Jedná se o velice univerzální formát, ze kterého lze převádět obraz do mnoha formátů. Jeho nejobvyklejší využití je práce s fotografiemi. Kvalitu a surová data tohoto formátu vyvažuje obrovská paměťová náročnost. [10]

5 PROGRAMOVACÍ JAZYKY

5.1 C++

C++ je jeden z nejběžnějších a nejpoužívanějších programovacích jazyků. V roce 1985 ho vytvořil dánský programátor Bjarne Stroustrup, a to rozšířením programovacího jazyka C. [3] Tento jazyk je multiplatformní a podporuje několik paradigmat[1] jako je generické programování[2] a objektově orientované programování[3]. Zástupci aplikací psaných v C++: produkty od Adobe, Mozilla Firefox, software od společnosti Microsoft. [7]

5.2 Java

Java se řadí mezi nejpopulárnější programovací jazyky. Vyvinula jej firma Sun Microsystems a byl představen v roce 1995. Je to velice stabilní, objektově orientovaný jazyk, který je přenosný napříč operačními systémy. Java je oficiálním jazykem pro vytváření aplikací pro systém android a získala si značnou podporu společností Google, tudíž je v ní napsána většina aplikací z obchodu Google Play. Další velké množství aplikací psaných v Javě je zastoupeno webovými aplikacemi jako například Twitter nebo Amazon. Další zástupci jsou uvedeni v *Obr. 3*. [4] [7]

Obr. 3, Zástupci programovacího Jazyku Java [16]

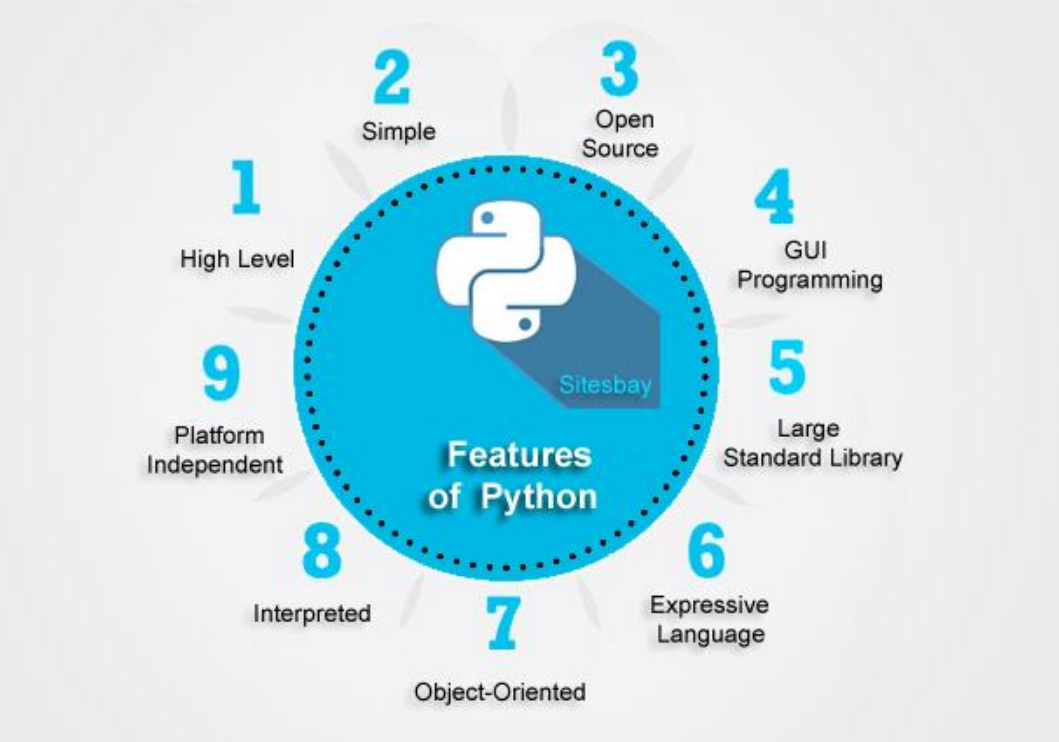
[1] Programovací styl.

[2] Rozdělení kódu na algoritmus a datové typy. Účelem je zápis kódu algoritmu vyjádřit obecně a s minimální závislostí na datových strukturách.

[3] Takzvaně událostmi řízené programování. Kód programu není nezpracováván chronologicky odshora dolů, ale podle uživatelem vyvolávaných akcí. Jednotlivé funkce
a procedury nejsou uvedeny za sebou v jednom těle programu, ale jsou přiřazeny definovaným objektům.

5.3 Python

Python je hybridní vysokoúrovňový open-source[4] programovací jazyk, který podporuje rozmanitá programovací paradigmata. První verze pythonu byla v roce 1991 navržena Guidem van Rossumem, v dnešní době je používán Python verze 3. Python je velice oblíbený díky své jednoduchosti z hlediska dostupnosti, pochopení, učení práce v něm a samotné rychlosti práce. Nejčastější využití je pro stolní a webové aplikace a dále pro umělou inteligenci (AI[5]) a strojové učení (ML[6]). Zásadní vlastnosti jazyku Python jsou uvedeny v *Obr. 4.* [5] [7]



Obr. 4, Vlastnosti programovacího jazyku Python [17]

[4] Veřejně legálně dostupný software.

[5] Artificial Intelligence

[6] Machine Learning

5.4 Srovnání a vyhodnocení

C++	JAVA	PYTHON
Kompilovaný programovací jazyk	Kompilovaný programovací jazyk	Interpretovaný programovací jazyk
Podporuje přetížení operátoru	Neodporuje přetížení operátoru	Podporuje přetížení operátoru
Poskytuje jedno i vícenásobnou dědičnost	Poskytuje částečnou vícenásobnou dědičnost pomocí rozhraní	Poskytuje jedno i vícenásobnou dědičnost
Závisí na platformě	Nezávisí na platformě	Nezávisí na platformě
Nepodporuje vlákna	Má vestavěnou podporu multithreadingu	Podporuje multithreading[7]
Má omezený počet podporovaných knihoven	Má podporu knihoven pro mnoho konceptů, jako je uživatelské rozhraní	Má obrovskou sadu knihoven, díky kterým je vhodný pro AI, data science[8] atd.
Délka kódu je o něco menší než u Javy, a to asi 1,5x.	Java má poměrně dlouhý kód.	Menší délka kódu, 3-4x méně než Java.
Funkce a proměnné se používají mimo třídu	Každý kousek kódu je uvnitř třídy.	Funkce a proměnné lze deklarovat a používat i mimo třídu.
Program C++ je rychle kompilující programovací jazyk.	Kompilátor programu Java je o něco pomalejší než C++	Jde o interpretovaný jazyk, tudíž je kompilace pomalejší
Přísně používá normy syntaxe	Přísně používá normy syntaxe	Použití “;“ není povinné.

Tab. 1, Srovnání vlastností programovacích jazyků [6]

Pro potřeby řešení problematiky této práce je možné využít všechny tři srovnávané programovací jazyky. Některé z jejich bazálních vlastností jsou srovnány v *Tab. 1*. Z důvodů veřejné licence, jednoduché dostupnosti, množství knihoven, pochopitelnosti a pracnosti kódu volím Python jako programovací jazyk této práce.

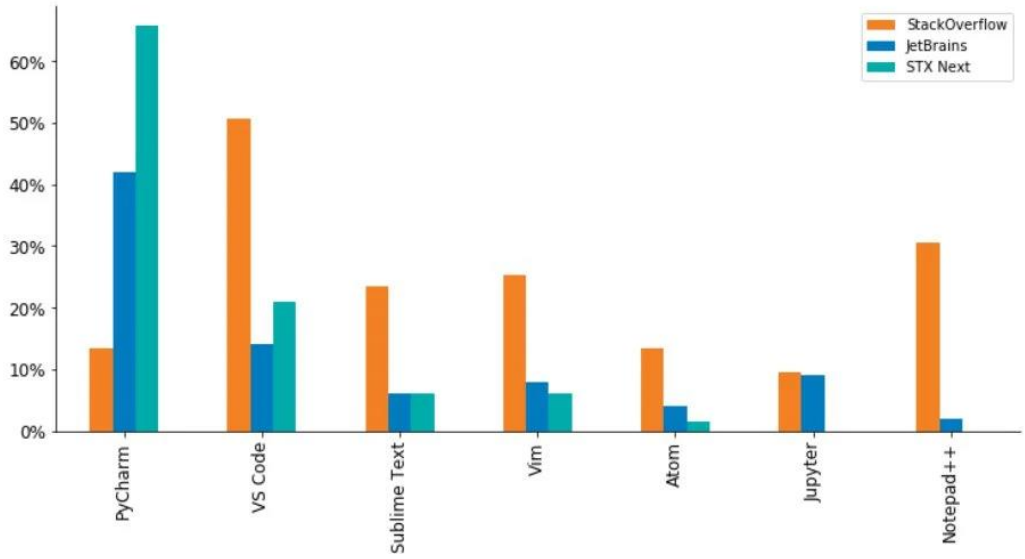
[7] Vytváření více vláken v jednom procesu

[8] Data science sjednocuje několik vědních oborů jako je statistika, strojové učení, analýza dat atd., za účelem vyčistit a analyzovat velké množství dat.

6 PROGRAMOVACÍ PROSTŘEDÍ (IDE)

V angličtině IDE (Integrated Development Environment). Jedná se o softwarovou aplikaci používanou vývojáři k vytváření programů. IDE mají vývojářům usnadnit práci tím, že kombinují nástroje, které jsou nezbytné při vývoji softwaru. [11] Jejich poměrové využívání profesionálními programátory je uvedeno v *Obr. 5*. Typické IDE bude obsahovat nástroje jako:

- Textový editor
- Kompilátor a/nebo interpret
- Debugger[9] a profilování kódu[10]
- Integrovaná správa verze
- Řada podpůrných nástrojů pro propojení s externími nástroji



Obr. 5, Graf preferencí IDE dle programátorů [18]

[9] Softwarový nástroj sloužící k hledání chyb a nedokonalostí v kódu.

[10] Profilování je technika, která zjišťuje čas řešení sekcí kódu a tím odhaluje úseky, které je možné zefektivnit přepsáním.

6.1 PyCharm

PyCharm je možné získat ve formě bezplatné komunitní edici. Její plná profesionální verze je ale placená. K dispozici je bezplatná zkušební verze. Pro účely studentů je možné využít PyCharm Educational Edition. Inspekce kódu PyCharm je jednou z nejpokročilejších ze všech Python IDE, takže je mnohem těžší zanést chyby, kterým lze předejít prostřednictvím statické a runtime analýzy kódu. PyCharm je kompatibilní se systémy: Windows, macOS, Linux. [11]

Funkce plné verze:

- Inteligentní editor kódu
- Navigace
- Refaktorovací nástroje[11]
- Debugger
- Funkční test kódu
- Python profiler
- Vzdálená správa
- Databázové nástroje
- Pomoc při vývoji webu (podpora frameworků jako Pyramid, Flask, JavaScript atd.)
- Vědecké nástroje (podpora balíčků jako Matplotlib a NumPy)
- Podpora VCS[12]

[11] Refaktoring je technika změny aplikace (buď kódu nebo architektury) tak, aby se navenek chovala stejně, ale interně se zlepšila. Těmito procesy může být zvýšena stabilita, výkon nebo snížena složitost.

[12] VCS (Version Control Systems) je abstraktní vrstva nad různými systémy správy verzí. Je navržena jako knihovna Pythonu bohatá na funkce s čistým API (API je zkratka pro "Application Programming Interface". Volně definované API popisuje vše, co potřebuje aplikační programátor vědět o části kódu, aby věděl, jak jej používat.).

6.2 Visual Studio Code

Visual Studio Code je kompletní editor kódu vyvinutý společností Microsoft. Je zdarma ke stažení a za příplatek ho je možné rozšířit o prémiové funkce. Je přizpůsobitelný, má jasné a snadné uživatelské rozhraní a instalace nových rozšíření je velice jednoduchá. Visual Studio Code je známé svými inovativními funkcemi jako je Live Share, která umožňuje provádět párové programování na dálku. Jeho jedinečná architektura také umožnila Microsoftu vytvořit z něj cloudovou službu, která poskytuje možnost využívat funkce VS Code odkudkoliv. Visual Studio Code je kompatibilní se systémy: Windows, macOS, Linux. [11]

Funkce:

- Zvýraznění syntaxe
- Párování závorek
- Automatické odsazení
- Vestavěná podpora pro dokončování kódu IntelliSense
- Bohaté porozumění sémantickému kódu
- Navigace
- nástroje pro Refaktorování kódu
- debugger
- podpora GIT[13]

[13] GIT (Global Information Tracker) je distribuovaný systém řízení s otevřeným zdrojovým kódem (také označovaný jako „řízení verzí“) běžně používaný ke sledování a správě změn souborů. Git se často používá jako systém správy verzí pro projekty Pythonu.

6.3 Sublime Text

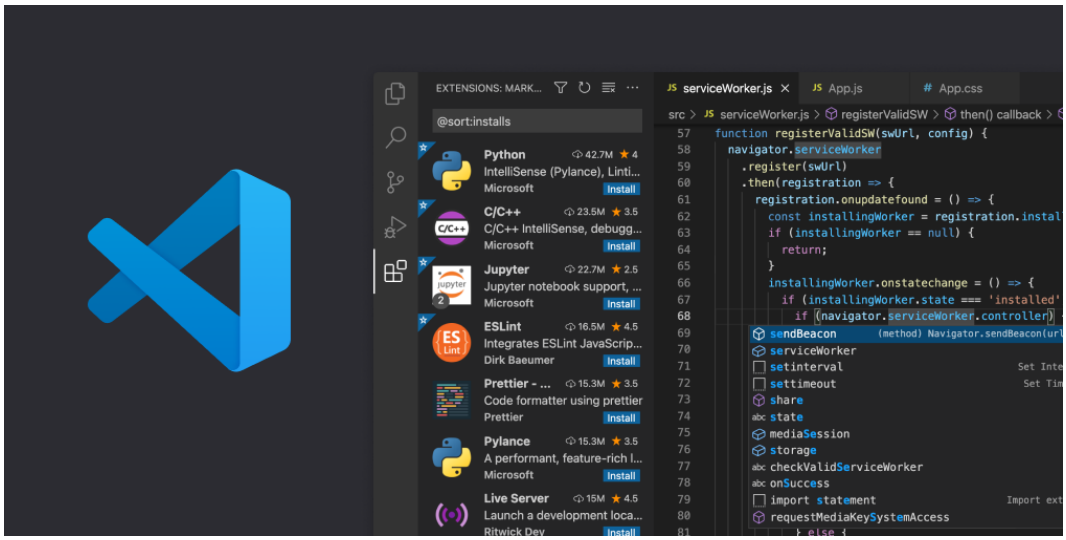
Sublime Text je sharewarový produkt, to znamená, že jej uživatel po určitou dobu může volně používat. Po uplynutí zkušební doby je potřeba zakoupit licenci. Je jednoduchý editor kódu pro více platforem, známý tím, že se snadno používá. Je rychlý, snadno přizpůsobitelný a má výkonné Python API, které zaručuje snadné přizpůsobení pomocí nástrojů a balíčků. Sublime Text je kompatibilní se systémy: Windows, macOS, Linux. [11]

Funkce:

- Úprava dělení
- Zvýraznění syntaxe
- Automatické dokončování
- Příkazová paleta
- Funkce “Goto”, ta má za cíl zvyšovat efektivitu

6.4 Vyhodnocení programovacího prostředí

Podobně jako u výběru programovacího jazyka i tentokrát splňují požadavky a potřeby zadání všechny tři srovnávaná programovací prostředí. U výběru IDE velice záleží na osobních preferencích a zkušenostech každého jednotlivého programátora. Z důvodu open-source licence, příjemného uživatelského rozhraní a předchozích zkušeností, volím jako programovací prostředí této práce Visual Studio Code. Prostředí je vyobrazeno na *Obr. 6*.



Obr. 6, Visual Studio Code [19]

7 KNIHOVNY PRO PRÁCI S OBRAZEM

7.1 OpenCV (Open Source Computer Vision)

OpenCV je open-source knihovna, která byla vyvinuta společností Intel v roce 2000. Většinou se používá v úlohách počítačového vidění jako je detekce objektů, detekce obličeje, rozpoznávání obličeje, segmentace obrazu atd. Obsahuje také mnoho užitečných funkcí používaných pro potřeby ML. [9] [12] [13]

Některé ze základních operací zpracování obrazu knihovny:

- Stupňování šedi
- Posuv obrázku
- Rotace obrázku
- Změna měřítka a velikosti jak celku, tak jednotlivých os

7.2 Scikit-image

Scikit-image je open-source knihovna pro zpracování obrázků založená na pythonu, která má některé části napsané v Cythonu[14] pro dosažení dobrého výkonu. [12] [13]

Některé ze základních operací zpracování obrazu knihovny:

- Segmentace
- Geometrické transformace
- Manipulace s barevným prostorem
- Analýza
- Filtrování
- Morfologické zpracování

[14] Cython je programovací jazyk, který je nadmnožinou programovacího jazyka Python navržený tak, aby měl výkon jako programovací jazyk C.

7.3 SciPy (Scientific Python)

SciPy je bezplatná a otevřená knihovna, která se používá pro matematické a vědecké výpočty. Může ale také provádět vícerozměrné zpracování obrazu pomocí submodulu scipy.ndimage. Dále poskytuje funkce pro provoz na n-rozměrných polích Numpy. [12]

Některé ze základních operací zpracování obrazu knihovny:

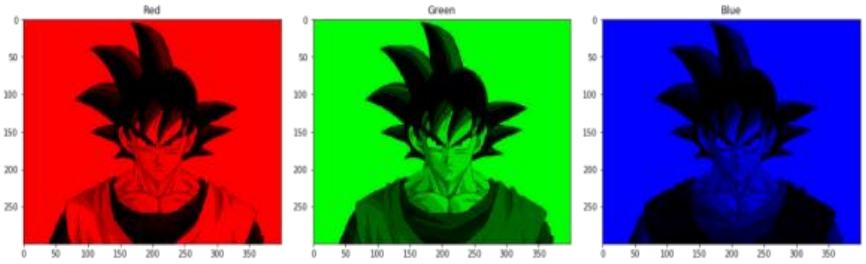
- Čtení obrázků
- Segmentace obrazu
- Konvoluce[15]
- Detekce obličeje

7.4 PIL (Python Imaging Library)

PIL (Python Imaging Library) je open-source knihovna pro úlohy zpracování obrázků, které vyžadují programovací jazyk python. PIL s obrázkem může provádět úkony, jako je čtení, změna měřítka a ukládání do různých formátů. Jinak tato knihovna ničím zvlášť nevyniká. [12] [13]

7.5 NumPy (Numerical Python)

Obrázek je v podstatě pole pixelů, kde každý pixel je reprezentován jednou (stupně šedi) nebo třemi (RGB) hodnotami. NumPy může snadno provádět úkoly, jako je ořezávání obrázků, maskování nebo manipulace s hodnotami pixelů. Zajímavou operací zpracování obrazu knihovny je rozdělení barevného obrázku na jednotlivé kanály (*Obr. 7*). [12] [13]



Obr. 7, Barevné kanály [20]

[15] Rozmazání obrazu

7.6 Mahotas

Mahotas patří mezi další open-source knihovny pro zpracování obrazu. Tato knihovna byla navržena zejména pro potřeby bioinformatiky, jako je obrazová analýza a rozpoznávání buněk a molekul. Je v ní zahrnuto mnoho implementovaných algoritmů psaných v C++. V dnešní době zahrnuje přes 100 funkcí pro zpracování obrazu. [12] [13]

Některé ze základních operací zpracování obrazu knihovny:

- Výpočty konvexních bodů
- Morfologické zpracování
- Přizpůsobení šablony
- Funkce hit-and-miss[16]

7.7 SimpleITK

Tato open-source knihovna se většinově používá pro úpravu obrazu, jako je segmentace, či registraci obrazu. Registrace obrazu je proces, při kterém jsou překryty dva nebo více obrazů. Takto překryté mohou být dále zpracovávány. [12]

7.8 Pgmagick

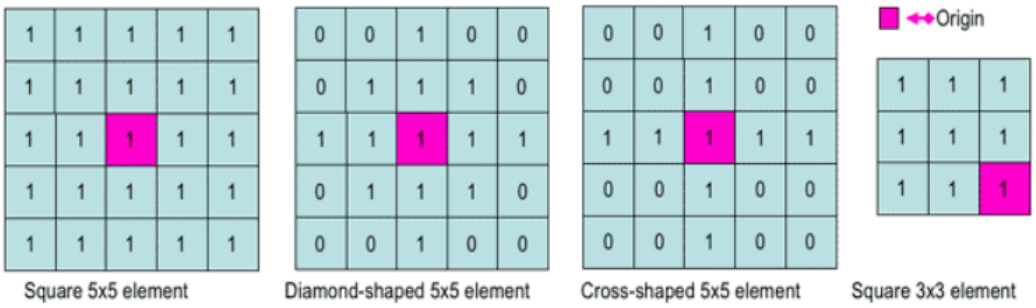
Pgmagick je open-source knihovna, využívána převážně ke grafické úpravě obrazu jako je kreslení textu, rotace, doostření, změna velikosti nebo obrazy s přechodem. [12]

[16] Je to obecná binární morfologická operace. Slouží k vyhledání konkrétních vzorů pixelů popředí a pozadí obrazu.

8 TECHNIKY ZPRACOVÁNÍ OBRAZU

8.1 Morfologické zpracování obrazu

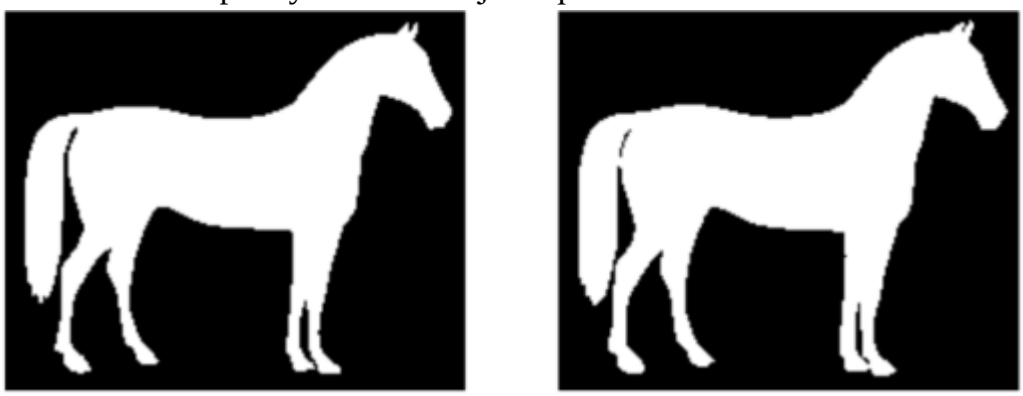
Morfologické zpracování se využívá pro odstranění nedokonalostí z binárních obrazů. Důvod je ten, že při vytváření binárního obrazu jednoduchým prahováním, může v obraze vznikat šum. Také pomáhá při vyhlazení obrazu. Princip funkce stojí na nelineárních operacích související se strukturou vlastností obrazu. Přesněji řečeno analyzuje obraz pomocí malé šablony ve formě matice, která obsahuje hodnoty 0 a 1. Ta se nazývá strukturovací prvek. Strukturovací prvek (*Obr. 8*) je umisťován na různá místa obrazu a je porovnáván se sousedními pixely dle rozvržení šablony. Tento proces se ještě dále dělí na dilataci (*Obr. 9*) a erozi (*Obr. 10*). [9] [13]



Obr. 8, Strukturovací prvky [21]

8.1.1 Dilatace

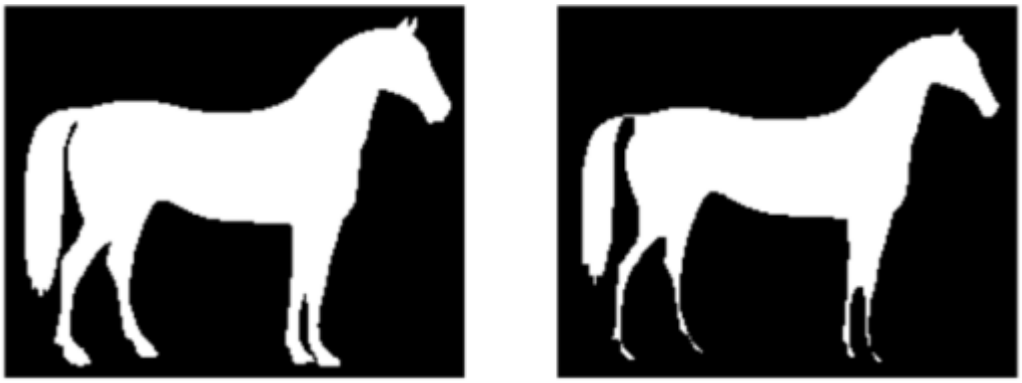
Dilatace pixely k hranici objektu přidává.



Obr. 9, Úprava obrazu dilatací [22]

8.1.2 Eroze

Eroze pixely z hranice objektu odebírá.



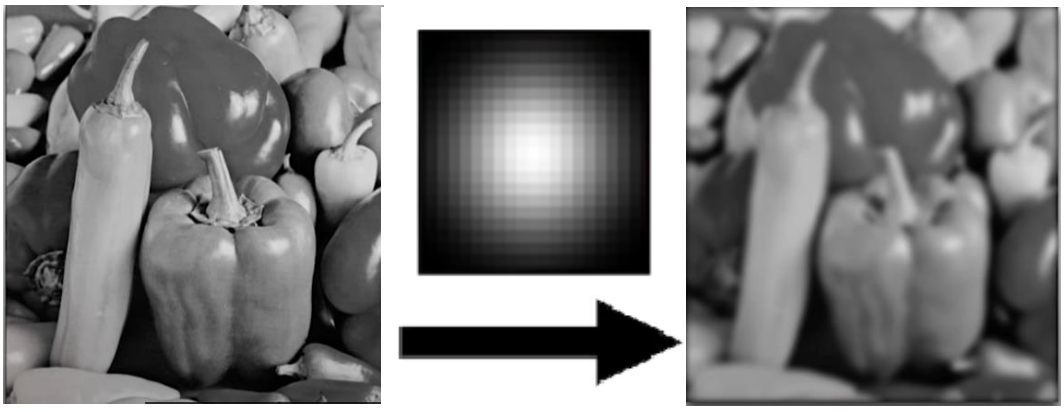
Obr. 10, Úprava obrazu erozí [23]

8.2 Gaussovské rozostření

Tato technika slouží ke snížení detailů a redukci obrazového šumu. Vizuálně se projevuje jako rozmazání. Proces se řídí Gaussovou funkcí (*rovnice(1*), a je možné ho použít samostatně ve vertikální i horizontální ose nebo v kombinaci obou. U potřeb počítačového vidění se někdy využívá jako vylepšení obrazu v různých měřítcích. Jeho efekt je vyobrazen na *Obr. 11*. [9] [13]

$$G(s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

(1)



Obr. 11, Gaussovské rozostření [24]

8.3 Zpracování obrazu Fourierovou transformací

Fourierova transformace rozkládá obraz na sinusovou a kosinovou složku. Velikost složky je spojena s kontrastem, prostorová frekvence souvisí s jasem a fáze se spojuje s informací o barvě. Tento proces se využívá zejména pro kompresi, rekonstrukci nebo filtrování obrazu. Pro obrazové procesy se používá diskrétní Fourierova transformace (rovnice (*2*) (*Obr*. 12). [1] [9] [13]

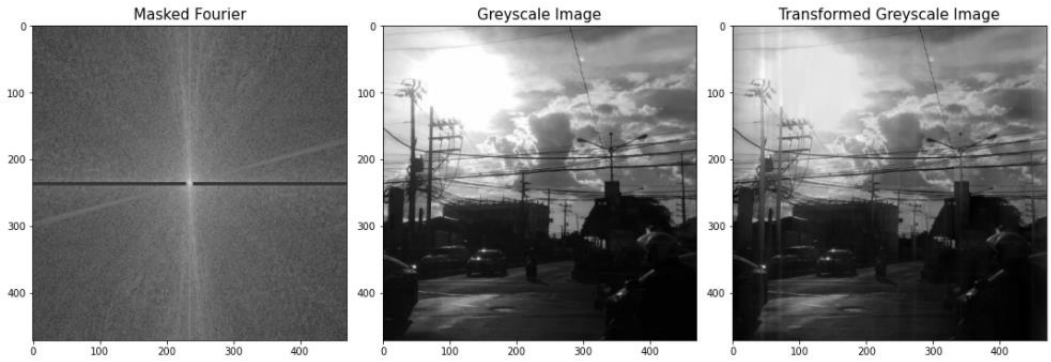
$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{u_x}{M}+\frac{v_y}{N}\right)}$$

(2)

Inverzní Fourierova transformace (*rovnice*(*3*) převádí transformaci zpět na obraz. [1]

$$f(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} F(u,v)e^{j2\pi\left(\frac{u_x}{M}+\frac{v_y}{N}\right)}$$

(3)



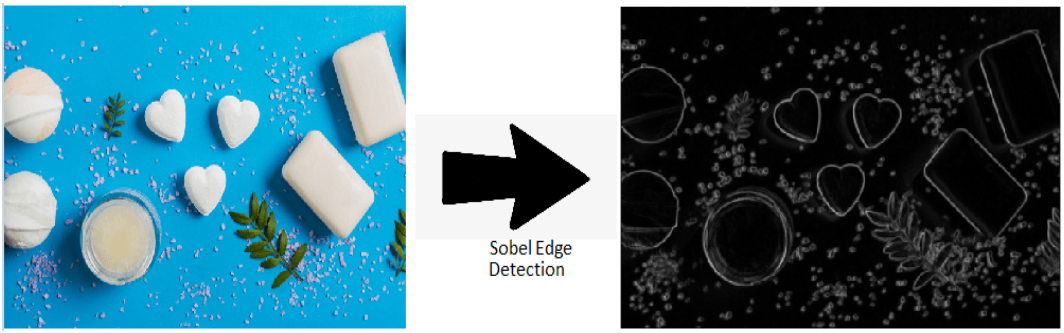
Obr. 12, Zpracování obrazu Fourierovou transformací [25]

8.4 Proces detekce hran

Proces detekce hran (*Obr. 13*) funguje na principu detekce nespojitosti jasu. Metoda je dost účinná díky tomu, že hrany definují tvar, který se posléze jednoduše a vcelku přesně v obraze hledá. Hrany jsou v obraze touto funkcí definovány jako lokální maxima gradientu. Nejběžněji se používá Sobelův algoritmus. Ten využívá dvou matic. Matice G_x (*rovnice(4)* a matice G_y (*5*), která je stejná jako matice G_x, ale otočená o 90°. Proces procházení matic obrazem probíhá jednotlivě. [9] [13]

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \cdot jednotlivé\ matice\ obrazu \quad (4)$$

$$G_y = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \cdot jednotlivé\ matice\ obrazu \quad (5)$$



Obr. 13, Proces detekce hran [26]

8.5 Wavelet (vlnková) transformace

Wavelet transformace (*Obr. 14*) je principem blízká transformaci Fourierově. Wavelet transformace bere v potaz mimo frekvence i čas. Tuto transformaci je vhodné použít pro nestacionární signály. Při použití tradičních filtrů je odstraněn šum, ale obraz se rozmaže. Tuto neduhu se vlnková transformace snaží řešit a je navržena tak, aby pro nízkofrekvenční složky bylo získáno dobré frekvenční rozlišení. [9] [13]

Obr. 14, Zpracování obrazu wavelet transformací [27]

9 NÁVRH A TVORBA MASKY OBRAZU

Veškeré návrhy a tvorba masek je prováděna v programovacím jazyku Python verze 4.5. Programování je realizováno v editoru zdrojového kódu Visual Studio Code.

9.1 Základní proces návrhu programu

9.1.1 Importace knihoven

Jako první je dobré naimportovat si veškeré potřebné knihovny pro danou problematiku. Importace je velice jednoduchá, slouží k ní příkaz „import“ za kterým následuje název knihovny. Ten je dobré si přejmenovat funkcí „as“ na zkratku nejlépe několika málo písmen (*Obr. 15*). To usnadňuje další psaní kódu. [9]

```
import cv2 as cv
import numpy as np
```

Obr. 15, Import knihoven

9.1.2 Rozbor příkazu

Jako příklad volím funkci pro načtení obrázku do programu (*Obr. 16*).

```
img = cv.imread('jablko.jpg')
```

Obr. 16, Načtení obrázku programem

„img“ představuje proměnou, kterou je možné zapsat textem a číslicemi téměř libovolně. Pod touto proměnou se s obrázkem dále pracuje.

„cv“ je pojmenování knihovny. Z té se následně volá potřebná funkce.

„imread“ představuje chtěnou funkci.

Za funkcí následuje prostor ohraničený kulatými závorkami, ten vyplňují funkce a proměnné (v tomto případě cestu k obrázku). Jelikož je obrázek uložen ve stejné složce jako program, postačí pouze název s příponou ohraničený apostrofy.

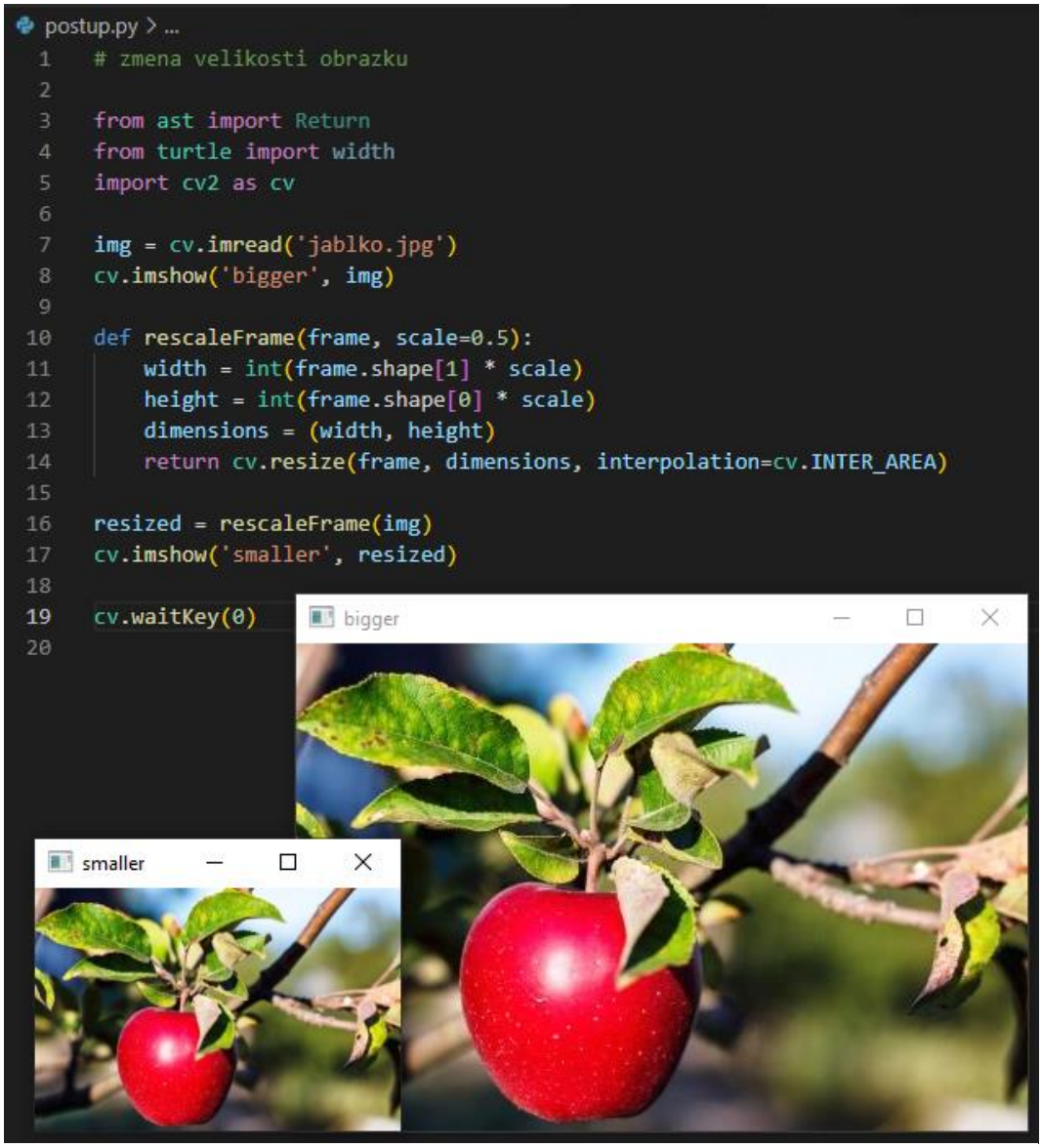
9.2 Načtení a zobrazení obrazu

Pro načtení obrazu do programu slouží funkce „imread“. Načtení obrazu je stěžejní pro jeho další zpracování. Po načtení je možné obraz zobrazit v okně za pomoci funkce „imshow“. Důležitá je také funkce „waitKey“, ta určuje čas zobrazení obrázku v milisekundách. Při hodnotě 0 se obraz ukazuje po celou dobu fungování programu. Algoritmus a výsledek je vyobrazen na *Obr. 17*. [9]

Obr. 17, Načtení a zobrazení obrázku [28]

9.3 Změna velikosti obrázku

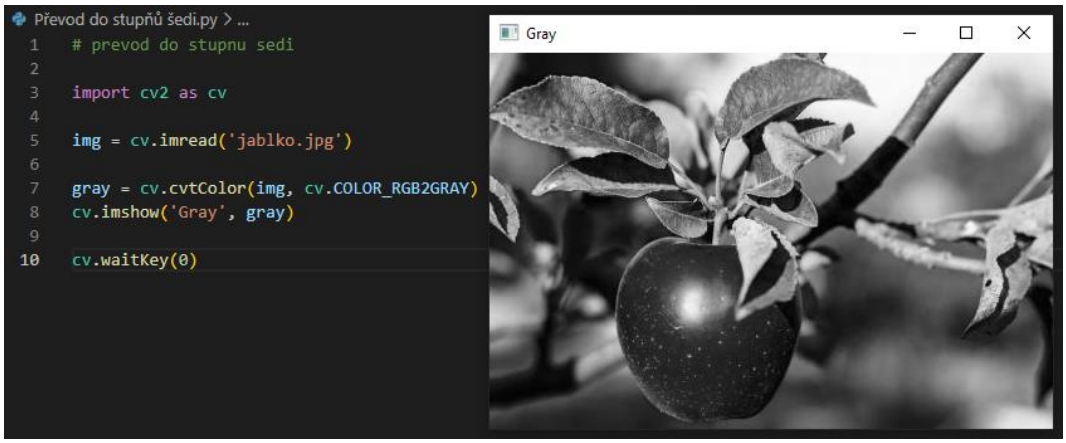
Relativní velikost obrázku je možné změnit funkcí „rescaleFrame“. Výška i šířka se mění podle zadaného násobku (*Obr. 18*). Obrázek pojmenovaný „bigger“ má původní velikost. Délka i výška obrázku „smaller“ je poloviční. [9]



Obr. 18, Změna velikosti obrázku

9.4 Převod do stupňů šedi

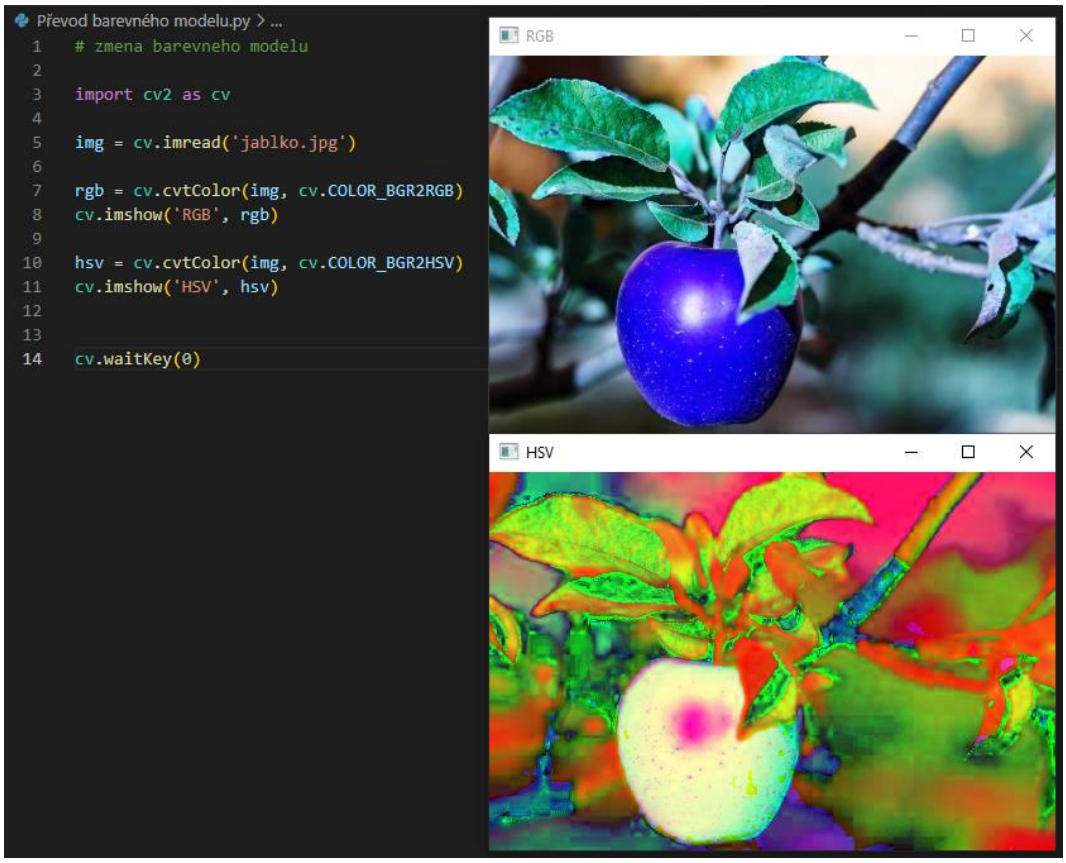
K tomuto pochodu slouží funkce „cvtColor“ (*Obr. 19*). [9]



Obr. 19, Stupně šedi

9.5 Změna barevného modelu

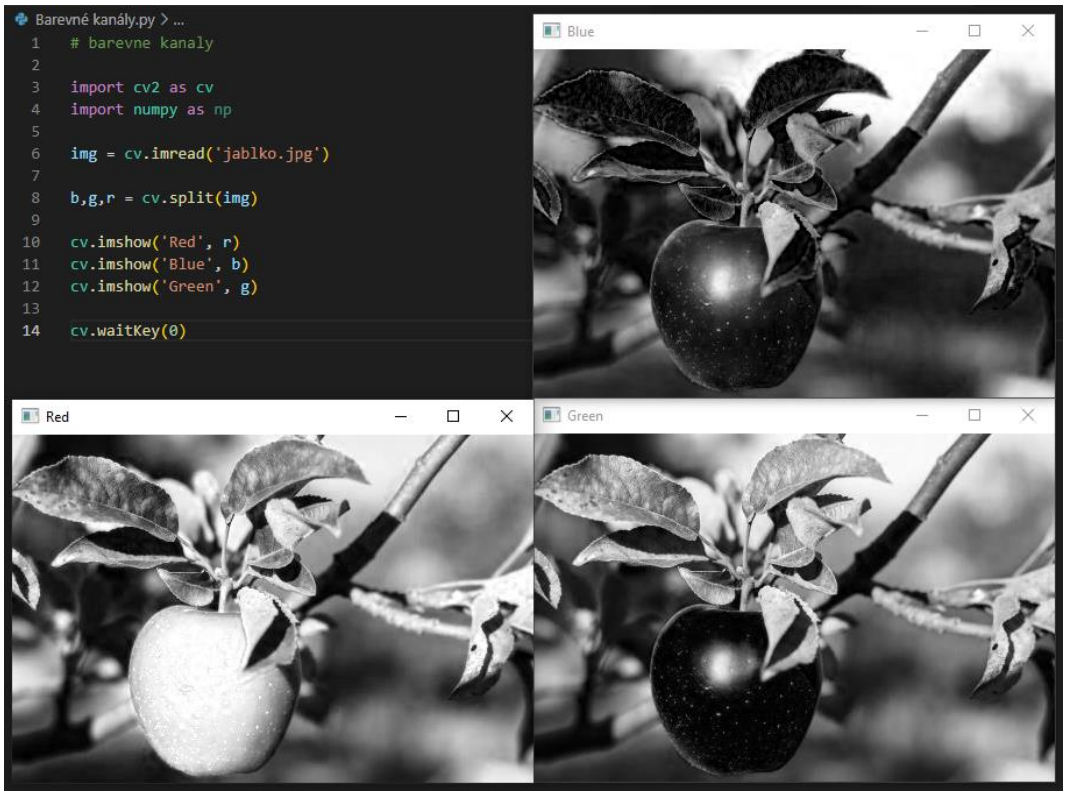
K transformaci barevného modelu slouží taktéž funkce „cvtColor“ (*Obr. 20*). Podporované modely je možné měnit jakkoliv. [9]



Obr. 20, Změna barevného modelu

9.6 Barevné kanály

Obrázek je možné rozdělit na jednotlivé barevné kanály do tří jednotlivých obrázků ve stupních šedi. Každý tento obrázek reprezentuje zastoupení barvy. Čím světlejší je oblast obrázku, tím více je zde barva zastoupena (*Obr. 21*). [9]



Obr. 21, Barevné kanály

9.7 Rozmazání

Typů rozmazání je několik. Liší se například strukturovacím prvkem nebo postupovou funkcí. Níže je uvedený kód základních rozmazávacích funkcí (*Obr. 22*) a jejich vizuální výsledek (*Obr. 23*). [9]

- Průměrové – funkce „blur“
- Mediánové – funkce „medianBlur“
- Gaussovské – funkce „GaussainBlur“
- Bilaterální – funkce „bilateralFilter“

Gaussovské rozmazání.py > ...

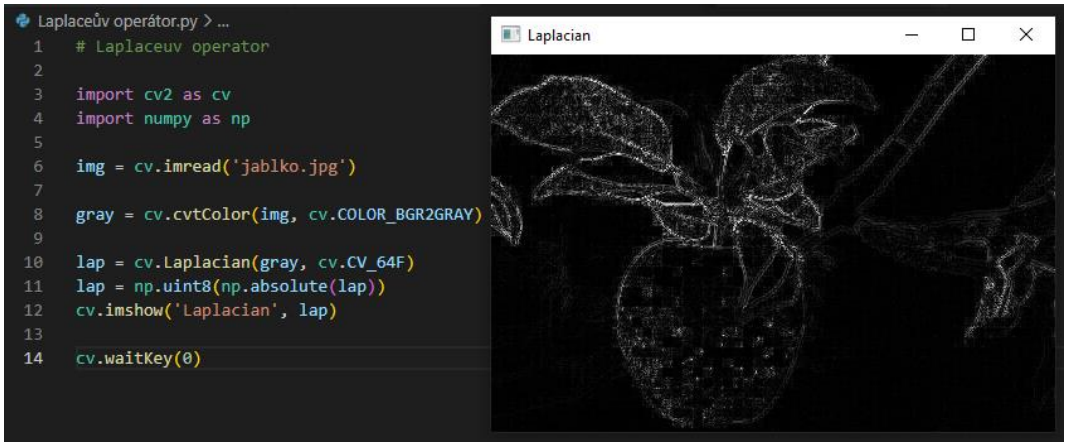
```python
# rozmazani

import cv2 as cv

img = cv.imread('jablko.jpg')

average = cv.blur(img, (11,11))
cv.imshow('Average', average)

median = cv.medianBlur(img, 11)
cv.imshow('Median', median)

gaussain = cv.GaussianBlur(img, (11,11), 0)
cv.imshow('Gaussain', gaussain)

bilateral = cv.bilateralFilter(img, 11, 11, 11)
cv.imshow('Bilateral', bilateral)


cv.waitKey(0)
```

Obr. 22, Kód funkcí rozmazání

Obr. 23, Výsledné rozmazání

9.8 Úprava za pomoci Laplaceova operátoru

Jedná se o kombinovanou metodu, kdy je nejprve potřeba převést obraz do stupňů šedi, jinak v něm zůstávají oblasti barevných teček. Metoda zapadá do procesů detekce hran. Funkce pro úpravu Laplaceovým operátorem je „Laplacian“ (*Obr. 24*). [9]



Obr. 24, Laplaceův operátor

9.9 Úprava funkcí Canny

Funkce „Canny“ je velice oblíbená pro zpracování obrazu za účelem detekce hran. Její Použití je jednoduché a efektivní (*Obr. 25*). [9]

Obr. 25, Canny

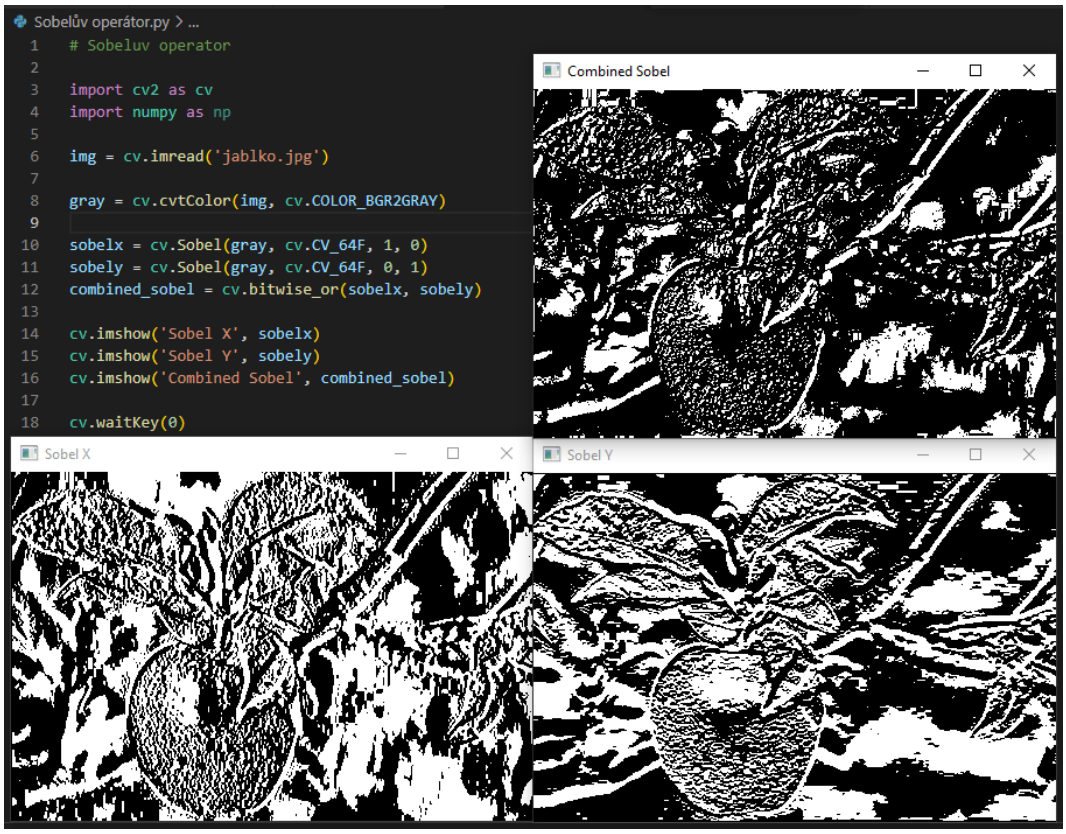
9.10 Dilatace a Eroze

Jedná se o kombinované metody, které často vycházejí z funkce „Canny“. Svým působením se snaží zefektivnit proces rozpoznání vzorů. Funkce dilatace je „dilate“ a funkce eroze je „erode“. Kód i výstupní obrazy jsou zaneseny na *Obr. 26*. [9]

Obr. 26, Dilatace a Eroze

9.11 Úprava pomocí Sobelova operátoru

Úprava pomocí Sobelova operátoru se jako další řadí mezi často využívané metody detekce hran (*Obr. 27*). Úprava jednotlivými maticemi je realizována funkcí „Sobel“, pro jejich kombinaci se dále používá „bitwise_or“. [9]



Obr. 27, Sobelův operátor

9.12 Prahování

Tuto metodu je možné rozdělit na dvě, a to jednoduché prahování (určeno funkcí „treshold“) a adaptivní prahování (určeno funkcí „adaptiveTreshold“) (*Obr. 28*). [9]

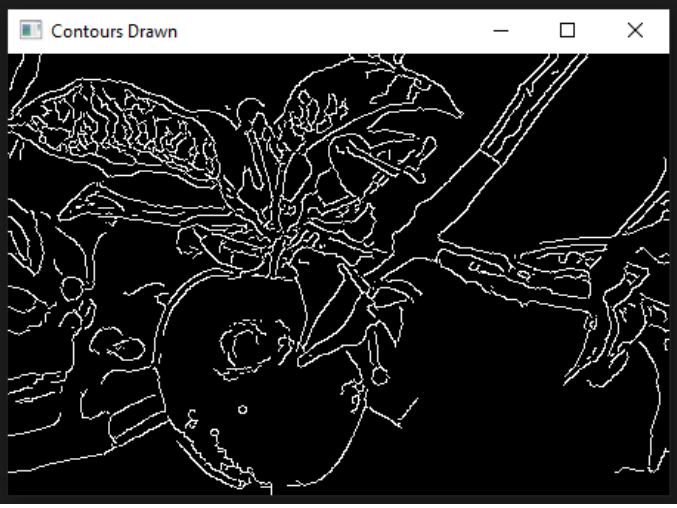
Obr. 28, Prahování

9.13 Metoda hledání kontur

V této kombinované metodě detekce hran byla použita velká část metod předchozích. Funkcí pro hledání kontur je „findContours“. Ta i vypisuje počet nalezených kontur (obrysů). Přímo pro tento případ bylo nalezeno 323 kontur. Níže je uveden algoritmus a jeho výstup (*Obr. 29*). [9]

Kontury.py > ...

```python
#Kontury

import cv2 as cv
from cv2 import contourArea
import numpy as np

img = cv.imread('jablko.jpg')

blank = np.zeros(img.shape, dtype='uint8')

gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

blur = cv.GaussianBlur(gray, (5,5), cv.BORDER_DEFAULT)

canny = cv.Canny(blur, 10, 80)

contours, hierarchies = cv.findContours(canny, cv.RETR_LIST, cv.CHAIN_APPROX_SIMPLE)
print(f'{len(contours)} contour(s) found!')

cv.drawContours(blank, contours, -1, (255,255,255), 1)
cv.imshow('Contours Drawn', blank)

cv.waitKey(0)
```



Obr. 29, Kontury

10 NÁVRH A TVORBA ZÁKLADNÍCH VZORŮ

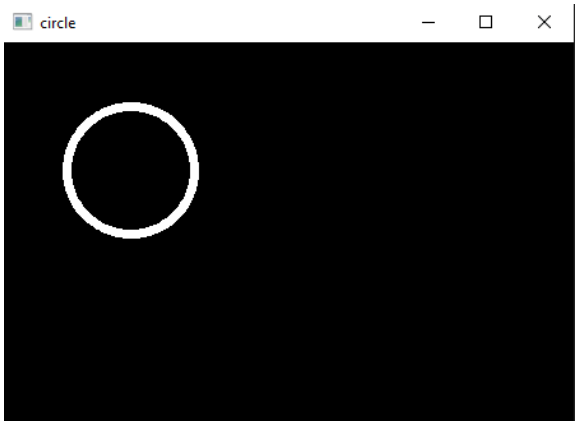
Mezi nejčastější tvary, které jsou potřebné při zpracování obrazu, patří dle mého názoru kruh a obdélník (popřípadě čtverec). Není to však nijak omezeno, vzor může být jakéhokoliv tvaru, pakliže zvyšuje efektivitu a přesnost výsledku algoritmu.

10.1 Kružnicový vzor

Pro vytvoření kruhového vzoru je nejlepší začít vytvořením černého pozadí pomocí „np.zeros“. Do něj je možné zapsat kruh libovolných rozměrů a barvy použitím funkce „circle“. Kružnici je možné číselně definovat tloušťku a při zadání hodnoty „-1“ dostáváme plný kruh. Výstupem kódu (*Obr.* 31) je bílá kružnice na černém pozadí (*Obr.* 30). [33] [34]

```
1   import cv2 as cv
2   import numpy as np
3
4   img = cv.imread('jablko.jpg')
5
6   blank=np.zeros((img.shape[:2]),dtype='uint8')
7   #vytvoreni cerneho obrazku ve tvaru predlohy
8
9   circle=cv.circle(blank, (100,100),50,255,5)
10  #zapsani kruznice do cerneho obrazku
11
12  cv.imshow('circle', circle)
13  cv.waitKey(0)
```

Obr. 31, Kód kružnicového vzoru



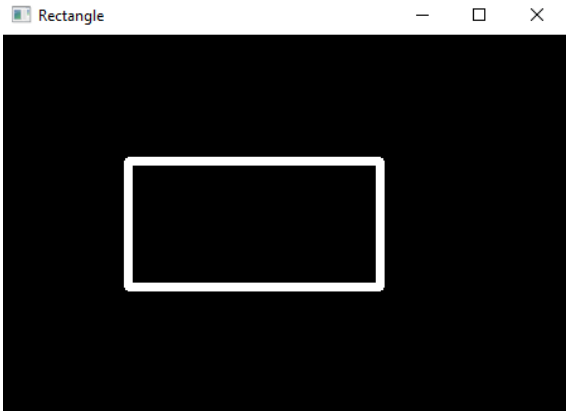
Obr. 30, Obrázek kružnicového vzoru

10.2 Obdélníkový vzor

Pro vytvoření obdélníkového vzoru je nejlepší začít vytvořením černého pozadí pomocí „np.zeros“. Do něj je možné zapsat obdélník libovolných rozměrů a barvy použitím funkce „rectangle“. Obdélníku je možné číselně definovat tloušťku a při zadání hodnoty „-1“ dostáváme plný obdélník. Výstupem kódu (Obr. 32) je bílý obdélník na černém pozadí (Obr. 33). [33] [34]

```
1   import cv2 as cv
2   import numpy as np
3
4   img = cv.imread('jablko.jpg')
5
6   blank=np.zeros((img.shape[:2]),dtype='uint8')
7
8   rectangle=cv.rectangle(blank.copy(), (100,100),(300,200),255,5,)
9
10  cv.imshow('Rectangle', rectangle)
11
12  cv.waitKey(0)
```

Obr. 32, Kód obdélníkového vzoru



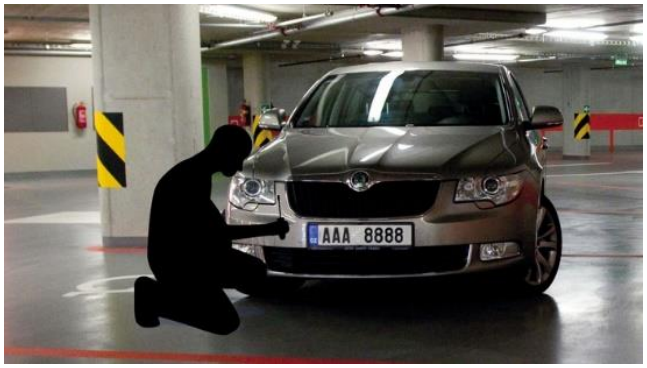
Obr. 33, Vizualizovaný proces čtvercového vzoru

11 DETEKCE SPZ

Tento program slouží k pochopení tvorby a optimalizace masky. Od reálného algoritmu má daleko, a to jak složitostí, tak účinností.

11.1 Vstup

Jako vstup jsem zvolil tři obrazy automobilu (*Obr. 36, Obr.* 34*, Obr.* 35), u kterých je SPZ dobře čitelná. Každý je focený z lehce jiného úhlu a je od jiného výrobce. Vzorek je takto malý kvůli časové náročnosti programu a výkonu mého zařízení.



Obr. 36, Automobil 1 [30]

Obr. 35, Automobil 2 [31]

Obr. 34, Automobil 3 [32]

11.2 Import knihoven

Do programu je nutné naimportovat všechny potřebné knihovny. V tomto případě je to: OpenCV, Matplotlib, Numpy, Imutils a Esyocr. Taky je nutné stáhnout knihovnu Tesseract-OCR a aktualizovat Python na verzi 4.5 nebo vyšší, pro vyhnutí se problému s neočekávanou C++ výjimkou.

11.3 Sběrné matice

Jelikož optimalizaci provádím použitím cyklů, je potřeba vytvoření matic, do kterých se hodnoty budou zapisovat. [34]

11.4 Úprava vstupu

Načtený obrázek je dobré převést do stupňů šedi, pracujeme pak jen s jedním kanálem. Následuje aplikace úprav obrázku. Pro tuto aplikaci jsem zvolil „bilateralFilter“ (*Obr. 37*) pro redukci šumu a „Canny“ (*Obr.* 38) pro zvýraznění hran. [33]

Obr. 37, Bilaterální filtr [33]

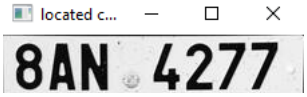
Obr. 38, Canny [33]

11.5 Cykly

Cykly jsou v aplikaci z důvodu optimalizace masky, tu proměnné mění.

11.6 Kontury

Díky konturám najdeme v obrazu chtěné uzavřené objekty. Určíme jejich lokaci a tuto lokaci dále izolujeme (*Obr. 39*). [33]



Obr. 39, SPZ

11.7 Rozpoznání textu

K rozpoznání textu jsem využil funkci „easyocr,Reader.readtext“. Ta nachází text bez mezer, proto jsem nalezené dva texty SPZ spojil. [35]

11.8 Třídění

Funkcí „if“ jsem vytřídil případy, kdy je správně čtená SPZ. Tím je možné získat optimální proměnné pro masku. [34]

11.9 Výstup

Výstupem je počet měření, správných výsledků a jejich procentuální zastoupení. Taktéž se generuje textový soubor s měřenými hodnotami pro jejich další analýzu.

11.10 Výsledek

Výsledný zápis hodnot je přidán ve formě přílohy. Jeho analýzou jsem došel k závěru, že proměnou „a“ pro bilaterální filtr je vhodné volit v rozpětí 50 a 100, proměnou „b“ pro funkci „Canny“ mezi hodnotami 20 a 400. Vhodnou proměnou „c“ jsme předchozími měřeními určil okolo hodnoty 10. Účinnosti (*Obr. 40 ,Obr. 41 ,Obr. 42*) jsou uvedeny pro zkouškový rozsah, pro optimalizovaný rozsah by se blížila 100 %.

```
Pocet spravne urcenych v = 27
Pocet pokusu n = 32
Ucinnost u = 84.375 %
```

Obr. 42, Výsledek automobilu 1

```
Pocet spravne urcenych v = 24
Pocet pokusu n = 32
Ucinnost u = 75.0 %
```

Obr. 41, Výsledek automobilu 2

```
Pocet spravne urcenych v = 24
Pocet pokusu n = 32
Ucinnost u = 75.0 %
```

Obr. 40, Výsledek automobilu 3

11.11 Kód programu

```
import cv2 as cv
from matplotlib import pyplot as plt
import numpy as np
import imutils
import easyocr
# import knihoven

img = cv.imread('cesta k obrazku') #nacteni obrazku
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY) #stupne sedi

bfilter_mat = []
edged_mat = []
contour_mat = []
spz_mat = []
#matice pro sber dat

n = 0

for a in range(0,199,50): #rozpeti pro bilateralni filtr

    for b in range(20,399,50): #rozpeti pro detekci hran

        bfilter = cv.bilateralFilter(gray, a, a, a) #bilateralni filtr
        edged = cv.Canny(bfilter, 30, b) #detekce hran

        keypoints = cv.findContours(edged.copy(), cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
        contours = imutils.grab_contours(keypoints)
        contours = sorted(contours, key=cv.contourArea, reverse=True)[:10]
        # kontury

        for c in range(10,11): #rozpeti pro kontury
            n = n + 1 #pocet cyklu
            try: #zabrana preruseni procesu chybou(nenalezenim textu)

                location = None
                for contour in contours:
                        approx = cv.approxPolyDP(contour, c, True)
                        if len(approx) == 4:
                            location = approx
                            break
                #lokace kontur

```

Obr. 43, Kód programu detekce SPZ (část 1)

```
42
43                  mask = np.zeros(gray.shape, np.uint8)
44                  new_image = cv.drawContours(mask, [location], 0,255, -1)
45                  new_image = cv.bitwise_and(img, img, mask=mask)
46                  #vytvoreni masky pro vyrez kontur
47
48                  (x,y) = np.where(mask==255)
49                  (x1, y1) = (np.min(x), np.min(y))
50                  (x2, y2) = (np.max(x), np.max(y))
51                  cropped_image = gray[x1:x2+1, y1:y2+1]
52                  #vyrez nalezenych kontur
53
54                  reader = easyocr.Reader(['en'], gpu=False)
55                  result = reader.readtext(cropped_image)
56                  #rozpoznani textu
57
58                  d = result[0][-2]
59                  e = result[1][-2]
60                  text = d+' '+e
61                  #uprava nalezeneho textu
62
63                  if text == "8AN 4277":
64                      spz = 1
65                  else:
66                      pass
67                  bfilter_mat.append(a)
68                  edged_mat.append(b)
69                  contour_mat.append(c)
70                  spz_mat.append(spz)
71                  #sortace spravnych vysledku
72
73              except:
74                  pass
75
76  test = np.matrix(np.vstack((bfilter_mat, edged_mat, contour_mat, spz_mat)))
77  #matice promenych pro spravne vysledky
78
79  np.savetxt('mereni.txt', test, fmt='%.2f', delimiter=',')
80  #ulozeni matice spravnych vysledku s promenymi do textoveho souboru
81
82  f = np.sum(spz_mat) #pocet spravnych vysledku
83
84  print("Pocet spravne urcenych v =",f) #vystup spravnych vysledku
85  print("Pocet pokusu n =",n) #vystup poctu pokusu
86  print("Ucinnost u =",100*f/n, "%") #vystup procentualniho zastoupeni spravnych hodnot v prcesu
87  print("Hodnoty mereni jsou ulozeny v mereni.txt") #oznameni ulozeni merenych hodnot
```

Obr. 44, Kód programu detekce SPZ (část 2)

12 ROZPOZNÁNÍ TEXTU SPZ

12.1 Vstup

Jako vstup slouží sada dvaceti zkouškových státních poznávacích značek (*Obr. 45*). Ty jsem se snažil vybrat tak, aby byl jejich text rozmanitý a zároveň, aby si byly podobny velikostí.

Obr. 45, Vstupní SPZ [38]

12.2 Maska

Pro problematiku tohoto úkolu jsem vytvořil 3 sady masek. Všechny mají stejný původ, a to obrázek písma psaným fontem pro SPZ. První maska je výřez znaku i s částí okolí (*Obr. 48*). Druhá maska je taktéž výřez znaku, ale s minimálním podílem okolí znaku (*Obr. 47*). Třetí maska vychází z masky první, je ale razantně upravena funkcí „Canny“ (*Obr. 46*).

Obr. 48, SPZ maska 1 [39]

Obr. 47, SPZ maska 2 [39]

Obr. 46, SPZ maska 3 [39]

12.3 Princip

Tento program se skládá z několika kaskádových cyklů. Postupně si načítá jak vstupní obrazy značek, tak obrazy znaků, ze kterých se dále tvoří masky. O samotné rozpoznávání se stará funkce „matchTemplate“, a to pomocí šesti metod. Pro optimalizaci jsem vytvořil cyklus, ve kterém postupně narůstá hodnota procentuální shody. Díky tomuto cyklu je možné určit rozpětí, kdy maska začíná určovat právě jeden správný znak a kdy už žádný znak neurčí.

12.4 Výstup

Výstupů jsem vytvořil hned několik. Při spuštění programu se v reálném čase vypisuje označení a cesta k SPZ, kterou program načetl ke zpracování. Při rozpoznání SPZ se systematicky vypisuje: srovnávací metoda, procentuální shoda, souřadnice jednotlivých nalezených znaků a samotné určené znaky SPZ. Po ukončení programu se ukládá textový soubor pro další vyhodnocení.

12.5 Výsledek

12.5.1 Maska 1

			metoda	zastoupení metod	průměrná procentuální shoda pro metodu [%]
počet výsledků	79		TM_CCOEFF	7	92,86
počet správných výsledků	65		TM_CCOEFF_NORMED	16	96,00
počet chybných výsledků	14		TM_CCORR	0	-
počet SPZ	20		TM_CCORR_NORMED	8	96,27
určeno SPZ	13		TM_SQDIFF	15	94,11
neurčeno SPZ	7		TM_SQDIFF_NORMED	19	95,14

Tab. 2, SPZ, výsledek masky 1

12.5.2 Maska 2

			metoda	zastoupení metod	průměrná procentuální shoda pro metodu [%]
počet výsledků	68		TM_CCOEFF	7	92,29
počet správných výsledků	61		TM_CCOEFF_NORMED	9	95,78
počet chybných výsledků	7		TM_CCORR	0	-
počet SPZ	20		TM_CCORR_NORMED	16	94,75
určeno SPZ	12		TM_SQDIFF	4	95,50
neurčeno SPZ	8		TM_SQDIFF_NORMED	25	90,56

Tab. 3, SPZ, výsledek masky 2

12.5.3 Maska 3

Maska 3 je pro tento proces nevyhovující.

12.6 Vyhodnocení

U masky číslo 3 jsem měl velká očekávání, opak byl bohužel pravdou. Výsledky při jejím použití byly nesmyslné, nebudu ji tedy zahrnovat do výsledného srovnání.

Jak u masky 1, tak u masky 2 převládá ve správném určování poslední metoda „TM_SQDIFF_NORMED“, naopak metoda „TM_CCORR“ nemá ani jeden správný výsledek, vyhodnocuji ji tedy jako nevyhovující. I přesto, že má maska 1 jednou tak velké množství chybných výsledků, co maska 2, vítězí jak v celkovém množství výsledků, tak v množství správně určených SPZ. Nejefektivnější maskou pro tuto aplikaci je maska 1.

12.7 Kód programu

```
import cv2 as cv
import numpy as np
import glob


num = 8     # pocet cislic na znacce

# pomocne matice
out = []
nul = [0]
abcd = np.arange(0,35)
text = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
        'A','B','C','D','E','F','G',
        'H','I','J','K','L','M','N',
        'P','Q','R','S','T','U','V',
        'W','X','Y','Z']

# cesty ke vstupnim datum
img_path = glob.glob("D:/Programy/Python/Projekty/spz/*.jpg")
path = glob.glob("D:/Programy/Python/Projekty/spz_masky/*.jpg")

# metody matchtemplatingu
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
            'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']
o = -1
for img_ in img_path:   # cyklus nacteni spz
    print('.....................................................................')
    print(img_)
    print('.....................................................................')
    img = cv.imread(img_,0)     # nacteni obrazku
    m,n = img.shape     # rozmery obrazku
    o = o + 1
    for n in range(80,99,2):
        accc = [n, n ,n ,n ,n]   # pomocna matice presnosti
        k = -1
        for meth in methods:
            k = k+1
            j = -1
            acc = accc[k]
            # pomocne matice vysledku
            lokace = []
            znak = []
            for lnmask in path:     # cyklus nacitani masky
                mask = cv.imread(lnmask,0)      # nacteni masky

                # rozmery obrazku
                h, w = mask.shape
```

Obr. 49, Kód programu rozpoznání textu SPZ (část 1)

```
            h, w = mask.shape
            height = m-20
            width = round(height*w/h)
            cut = cv.resize(mask, (width, height))

            # barevna inverze
            inv = cv.bitwise_not(cut)

            j = j + 1

            # matchtemplating
            img2 = img.copy()
            method = eval(meth)
            res = cv.matchTemplate(img2,cut,method)
            high = cv.matchTemplate(cut,cut,method)
            low = cv.matchTemplate(cut,inv,method)

            # rozdeleni podle podminek metod
            if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
                # rozsah metod
                top = high.min()
                bot = low.min()

                threshold = bot*(1-acc/100)     # threshold

                # lokace
                loc = np.where(res <= threshold)
                for pt in zip(*loc[::-1]):
                    cv.rectangle(img2, pt, (pt[0] + width, pt[1] + height),(0,0,255), 1 )
                locmat = np.concatenate([nul,np.sort(loc[1])], axis=0)
                lenght = len(locmat)
                f = []
                try:
                    for i in range(0,lenght):
                        if locmat[i+1] - locmat[i] - 1 == 0 or locmat[i+1] == locmat[i]:
                            pass
                        else:
                            f.append(locmat[i+1])
                except:
                    pass

                # pomocne matice
                ones = np.ones((len(f),), dtype=int)
                letmat = ones*abcd[j]

                # cyklicky zapis znaku a jeho souradnice
                lokace.extend(f)
                znak.extend(letmat)

            else:
                # rozsah metod
                top = high.max()
                bot = low.max()
                diff = top-bot

                threshold = bot + diff*acc/100      # threshold

                # lokace
                loc = np.where(res >= threshold)
```

Obr. 50, Kód programu rozpoznání textu SPZ (část 2)

```
                    loc = np.where(res >= threshold)
                    for pt in zip(*loc[::-1]):
                        cv.rectangle(img2, pt, (pt[0] + width, pt[1] + height),(0,0,255), 1 )
                    locmat = np.concatenate([nul,np.sort(loc[1])], axis=0)
                    lenght = len(locmat)
                    f = []
                    try:
                        for i in range(0,lenght):
                            if locmat[i+1] - locmat[i] - 1 == 0 or locmat[i+1] == locmat[i]:
                                pass
                            else:
                                f.append(locmat[i+1])
                    except:
                        pass

                    # pomocne matice
                    ones = np.ones((len(f),), dtype=int)
                    letmat = ones*abcd[j]

                    # cyklicky zapis znaku a jeho souradnice
                    lokace.extend(f)
                    znak.extend(letmat)

            # sortovani matice se znaky a lokacemi
            nbv = np.vstack((lokace,znak))
            nbv= np.transpose(nbv)
            nbv= sorted(nbv,key=lambda x: x[0])
            nbv= np.transpose(nbv)

            spz = []    # matice pro spz3

            # cyklus pro vypis znacky
            try:
                for l in range(0,len(nbv[0])):
                    tr = text[nbv[1][l]]
                    spz.append(tr)
                # print('spz =',spz[0]+spz[1]+spz[2]+' '+spz[3]
                    # +spz[4]+spz[5]+spz[6]+spz[7])
                plen = len(nbv[0])

                out_ = []
                if plen == num:
                    print(meth)
                    print('shoda = ',acc)
                    print('souradnice',nbv[0])
                    print('spz',spz)
                    out_.append(o)
                    out_.append(acc)
                    out_.append(k)

                    cv.imshow('img', img)
                    cv.waitKey(0)
                if out_ != []:
                    out.append(out_)

            except:
                pass

np.savetxt('spz_original.txt', out, fmt= '%1.4e', delimiter=',')
```

Obr. 51, Kód programu rozpoznání textu SPZ (část 3)

13 DETEKCE OBJEKTŮ A JEJICH ZVÝRAZNĚNÍ

Tento program slouží k pochopení tvorby a optimalizace masky. Od reálného algoritmu má daleko, a to jak složitostí, tak účinností.

13.1 Vstup

Jako vstup jsem zvolil jednoduchý obrázek pro dobré výsledky a optimalizaci (*Obr. 52*).

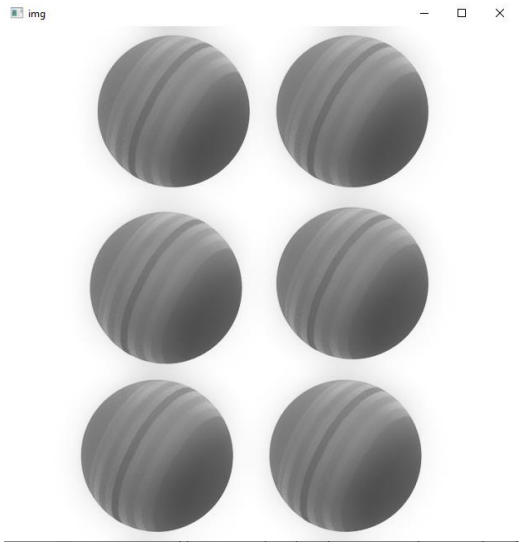
Obr. 52, Vzorek (golfové míčky) [36]

13.2 Import knihoven

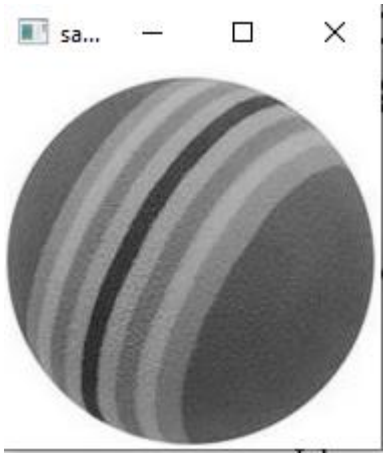
Pro tento případ jsem volil knihovny OpenCV a NumPy. Také bylo zapotřebí naimportovat funkce „maximum_filter“ a „minimum_filter“ z knihovny SciPy.

13.3 Vstupní obrázek a šablona

Vstupní obrázek (*Obr. 53*) jsem naimportoval ve stupních šedi a aplikoval jsem rozmazání pomocí bilaterálního filtru za pomoci parametru „l“. Pro správný chod je stěžejní tento parametr zvolit správně. Oba postupy velice ovlivňují výslednou přesnost. Dále jsem naimportoval šablonu (*Obr. 54*) ve stupních šedi, kterou jsem vytvořil výřezem jednoho objektu z obrázku. Pro obrazy, kde není statické pozadí je příhodné šablonu dále upravit. Příklad úpravy jsem uvedl v kódu programu. [33]



Obr. 53, Vstupní obrázek (ve stupních šedi a s bilaterálním filtrem)



Obr. 54, Šablona (ve stupních šedi)

13.4 Funkce „MatchTemplate“

Tato funkce postupně přesouvá šablonu po souřadnicích obrázku a srovnává je. Pro tuto funkci je v knihovně OpenCV přednastaveno 6 funkčních metod (TM_SQDIFF, TM_SQDIFF_NORMED, TM_CCORR, TM_CCORR_NORMED, TM_CCOEFF, TM_CCOEFF_NORMED). Ty jsem v cyklu zaměnil, abych pro každou dostal výstup. Tento cyklus jsem dále rozdělil do dvou větví z důvodu, že některé funkce mají jako výstup při shodě minimum a některé maximum. [33]

13.5 Počet objektů

Problematiku počtu objektů jsem řešil funkcemi „minimum_filter“, „maximum˙filter“ a „where“. Jejich kombinace segmentuje obraz na matice, v kterých nachází maxima nebo minima. Toto maximum nebo minimum lze označit jako střed objektu. [34]

13.6 Zvýraznění objektů

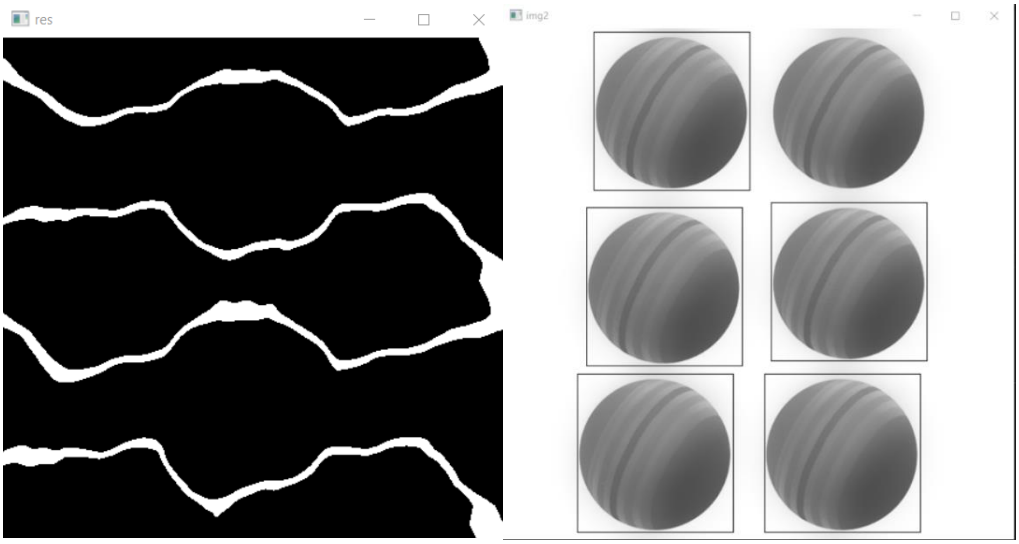
Díky prahování jsem získal souřadnice maxim a minim, do nich jsem následně vynesl středy obdélníků pro zvýraznění objektů. U tohoto procesu je důležité dobře nastavit parametry „a“ a „b“, pro správné vyhodnocení objektů. [33]

13.7 Výstup

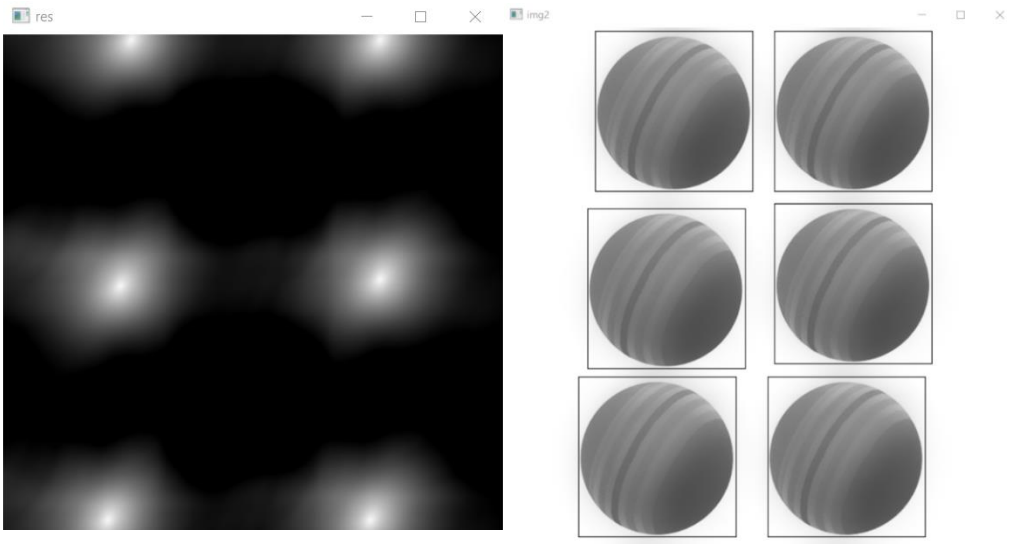
Výstupem je vizuální zobrazení funkce „matchTemplate“, obrázek se zvýrazněnými objekty a číslo v, které udává počet nalezených objektů v obrázku. [33]

13.8 Výsledek

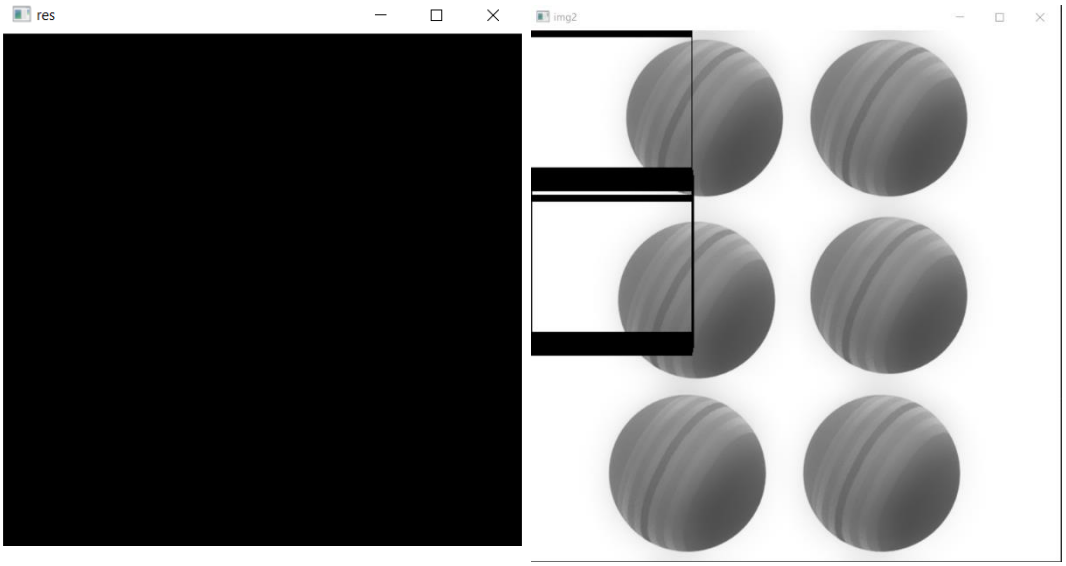
Pro vstupní obrázek, na který jsem parametry optimalizoval, je výsledek statický. U čtyř metod je všech šest objektů je zvýrazněno správně (*Obr. 55-58, 60, 61*). Počet objektů „v“ (*Obr. 59*) je spočte správně u všech metod korm druhé. Při aplikaci na jiný obrázek se stejnými parametry účinnost rapidně klesá.



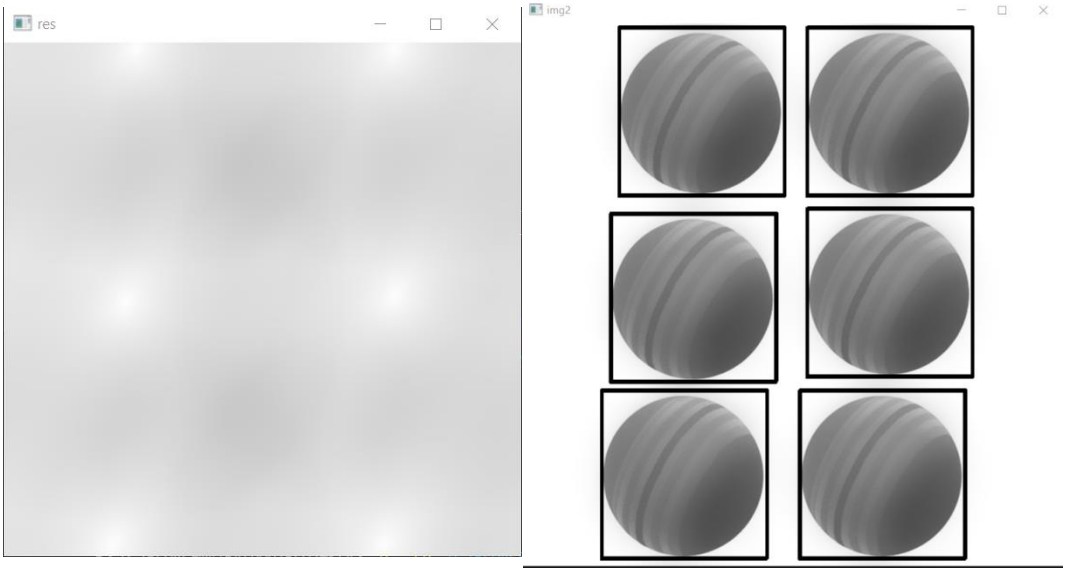
Obr. 55, Výsledek dle funkce „cv.TM_CCOEFF“ (metoda 1)



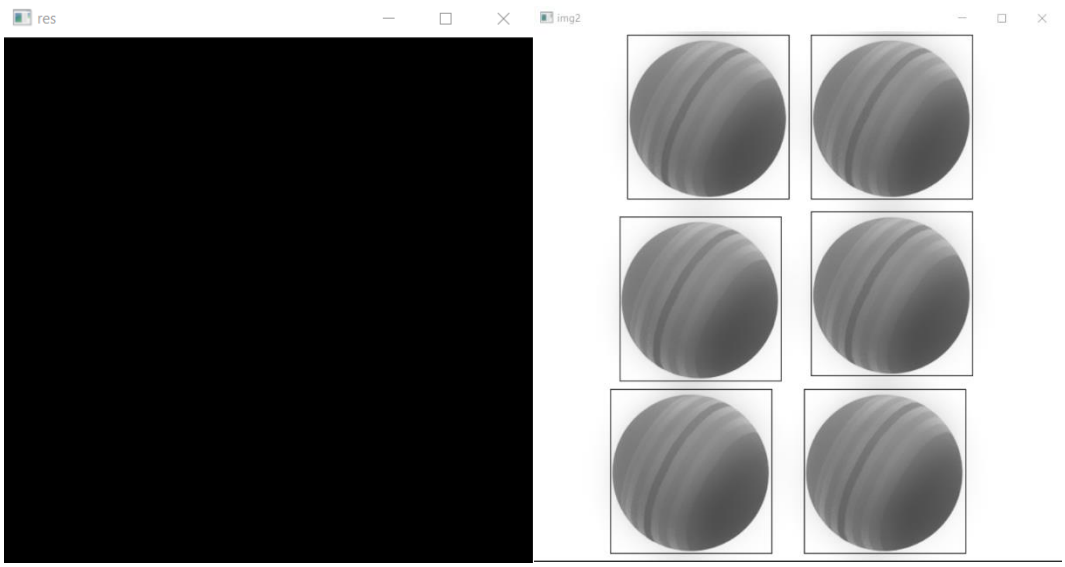
Obr. 56, Výsledek dle funkce „cv.TM_CCOEFF_NORMED“ (metoda 2)



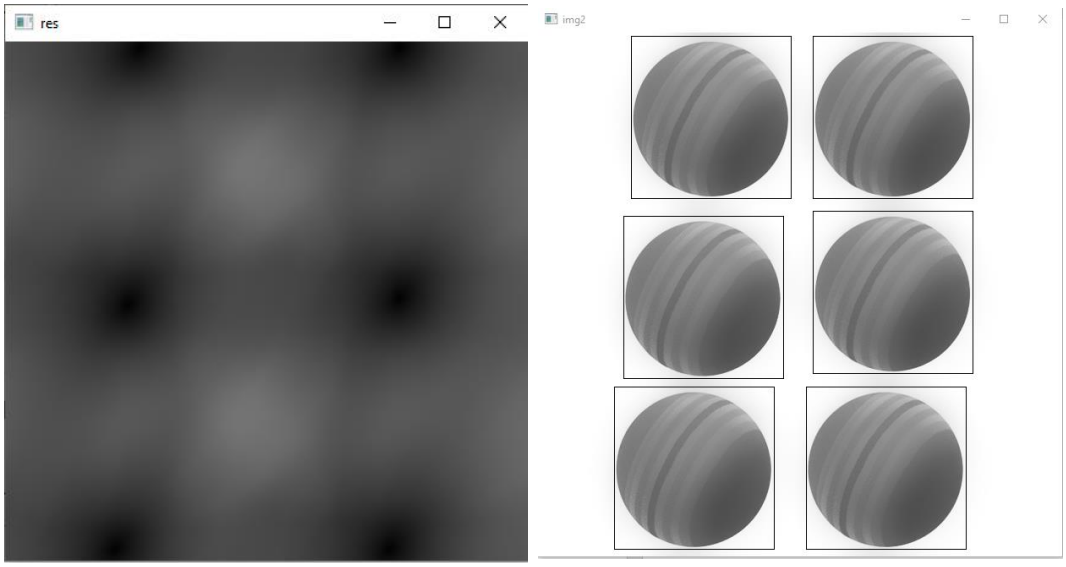
Obr. 58, Výsledek dle funkce „cv.TM_CCORR“ (metoda 3)



Obr. 57, Výsledek dle funkce „cv.TM_CCORR_NORMED“ (metoda 4)



Obr. 61, Výsledek dle funkce „cv.TM_SQDIFF“ (metoda 5)



Obr. 60, Výsledek dle funkce „cv.TM_SQDIFF_NORMED“ (metoda 6)

```
v = 6 pro metodu 4
v = 6 pro metodu 5
v = 8 pro metodu 2
v = 6 pro metodu 3
v = 6 pro metodu 0
v = 6 pro metodu 1
```

Obr. 59, Výsledky počtu objektů dle metod

13.9 Kód programu

```python
import cv2 as cv
from matplotlib import pyplot as plt
import numpy as np
import imutils
import easyocr
from scipy.ndimage import maximum_filter
from scipy.ndimage import minimum_filter
#inportovane knihovny a funkce

l =70 #promenna pro bilateralni filtr
img = cv.bilateralFilter(cv.imread('balls.jpg',0),l,l,l) #filtr pro rozmazani

cut = cv.imread('balls_template.jpg',0) #vyrez slouzici jako sablona
h, w = cut.shape #rozmery sablony


#uprava masky, pro zkousene vzorky snizuje vykonnost>
    # blank=np.zeros((cut.shape),dtype='uint8')
    # correction = 1
    # x = round(w/2 - correction)
    # y = round(h/2 - correction)
    # if h > w:
    #     r = x
    # else:
    #     r = y
    # circle=cv.circle(blank.copy(), (x,y), r, 255, -1)
    # template = cv.bitwise_and(cut, cut, mask=circle)


methods = ['cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED', 'cv.TM_CCORR',
           'cv.TM_CCORR_NORMED', 'cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED'] #metodty matchtemplate

for meth in methods: #cyklus pro zamenu hodnot
    img2 = img.copy() #kopie vychoziho obrazku kam se budou zapisovat objekty
    method = eval(meth)
    res = cv.matchTemplate(img2,cut,method) #funkce pro porovnani sablony s obrazkem

    cv.imshow('res',res) #zobrazeni obrazku pod vlivem funkce matchtemplate

    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]: #tyto metody hodnoti podle minima matice res
        filter = minimum_filter(res, size=h)
        fmat = np.where(filter == res, 1, 0)
        #filtrovani minim matice pro ziskani poctu objektu

        print('v =', np.sum(fmat), 'pro metodu', method) #vystup poctu objektu

        x = res.min() #minimalni hodnota matice res
        a = 1.1 #promena pro upravu thresholdu
        threshold = a*x
        loc = np.where(res <= threshold) #souradnice minim

        for pt in zip(*loc[::-1]):
            cv.rectangle(img, pt, (pt[0] + w, pt[1] + h),(0,0,255), 1 )
        cv.imshow('img', img)
        cv.waitKey(1000)
        # vukresleni obdelniku okolo identifikovanych objektu do obrazku a jeho zobrazeni

```

Obr. 62, Kód programu detekce objektů (část 1)

```
57
58      else: #tyto metody hodnoti podle maxima matice res
59          filter = maximum_filter(res, size=h)
60          fmat = np.where(filter == res, 1, 0)
61          #filtrovani maxim matice pro ziskani poctu objektu
62
63          print('v =', np.sum(fmat),'pro metodu', method) #vystup poctu objektu
64
65          x = res.max() #maximalni hodnota matice res
66          b = 0.955 #promena pro upravu thresholdu
67          threshold = b*x
68          loc = np.where(res >= threshold) #souradnice maxim
69
70
71          for pt in zip(*loc[::-1]):
72              cv.rectangle(img, pt, (pt[0] + w, pt[1] + h),(0,0,255), 1 )
73          cv.imshow('img', img)
74          cv.waitKey(1000)
75          # vukresleni obdelniku okolo identifikovanych objektu do obrazku a jeho zobrazeni
76
77      cv.destroyAllWindows
```

Obr. 63, Kód programu detekce objektů (část 2)

14 DETEKCE OBJEKTŮ POMOCÍ RŮZNÝCH MASEK

14.1 Vstup

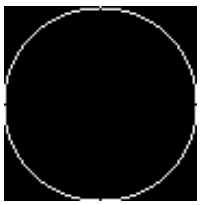
Jako vstup slouží pouze jeden obrázek s jablky (*Obr. 64*).

Obr. 64, Vstupní obrázek jablek [40]

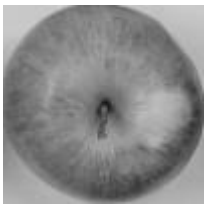
14.2 Maska

Pro tuto aplikaci jsem navrhl čtyři masky, první dvě jsem vytvořil přímo v programu jako černobílé obrazce (*Obr. 68, Obr. 67*). Třetí maska je čistý výřez průměrného nezdeformovaného jablka, kterou jsem pro přesnější výsledky otočil o 180 stupňů (*Obr. 66*). Čtvrtá maska je složená ze tří výřezů jablek. Tyto výřezy jsem volil záměrně co nejvíce odlišné, aby maska pokryla co největší množství dále srovnávaných jablek (*Obr. 65*).

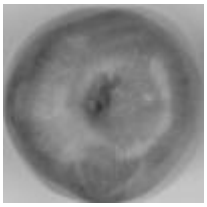
Obr. 68, Jablka, maska 1



Obr. 67, Jablka, maska 2



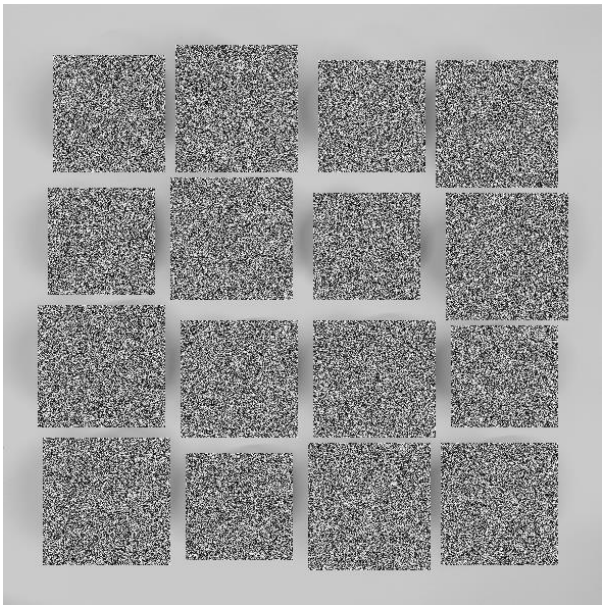
Obr. 66, Jablka, maska 3



Obr. 65, Jablka, maska 4

14.3 Princip

Princip tohoto procesu je na pochopení v celku jednoduchý. Prvním krokem bylo najít souřadnice všech jablek. Následuje cyklická separace jednotlivých segmentů obrazu. Ty se poté srovnávají s maskou pomocí funkce „matchTemplate“. Tomuto srovnání ještě předchází specifické úpravy jak masky, tak vstupního segmentu. Po srovnání masek s jednotlivými jablky zbývá ze vstupního obrazu pouze pozadí a „hluchá místa“ po jablkách. Tato místa jsem zaplnil bílou, černou nebo barevným šumem (*Obr. 69*). Jako poslední krok je srovnání masky se zbytkem ze vstupního obrazu. To z důvodu kontroly, zda program neurčí jablko i na pozadí, kde ale žádné není.



Obr. 69, Nahrazení šumem

14.4 Výstup

Při spuštění programu se do matic v reálním čase vypisují hodnoty procentuální shody masky a segmentu vstupního obrazu pro různé hodnoty rozmazání obrazu, metody srovnání a hodnot parametru funkce „Canny“. Pro tyto proměnné se vypisuje i maximální procentuální shoda masky s pozadím a také zprůměrovaná hodnota procentuální shody všech 16jablek dohromady.

14.5 Výsledek

Pro masku 1 a 2 je stěžejní funkce „Canny“, funkce „blur“ je zde zanedbatelná. Pro masku 3 a 4 je tomu naopak, „blur“ ovlivňuje výsledek nejvíce.

„1“ = vyhovuje, „0“ = nevyhovuje, „-“ = neprůkazné

maska 1	canny	0	5	10	15	20	25	30	35	40	45
TM_CCOEFF		0	0	0	0	0	1	1	1	1	1
TM_CCOEFF_NORMED		0	0	0	0	1	1	1	1	1	1
TM_CCORR		1	1	1	1	1	1	1	1	1	1
TM_CCORR_NORMED		1	1	1	1	1	1	1	1	1	1
TM_SQDIFF		1	1	1	1	1	1	1	1	1	1
TM_SQDIFF_NORMED		-	-	-	-	-	-	-	-	-	-

Tab. 4, Jablka, výsledek masky 1

maska 2	canny	0	5	10	15	20	25	30	35	40	45
TM_CCOEFF		0	0	0	0	0	0	0	0	0	0
TM_CCOEFF_NORMED		0	0	0	0	0	0	0	0	0	0
TM_CCORR		0	0	0	0	0	0	0	0	0	0
TM_CCORR_NORMED		1	1	1	1	1	1	1	1	1	1
TM_SQDIFF		1	1	0	0	0	0	0	0	0	0
TM_SQDIFF_NORMED		-	-	-	-	-	-	-	-	-	-

Tab. 5, Jablka, výsledek masky 2

Maska 3	blur	2	4	6	8	10	12	14	16	18
TM_CCOEFF		0	0	0	0	0	0	0	0	0
TM_CCOEFF_NORMED		1	1	1	1	0	0	0	0	0
TM_CCORR		-	-	-	-	-	-	-	-	-
TM_CCORR_NORMED		1	0	0	0	0	0	0	0	0
TM_SQDIFF		1	0	0	0	0	0	0	0	0
TM_SQDIFF_NORMED		1	0	0	0	0	0	0	0	0

Tab. 6, Jablka, výsledek masky 3

Maska 4	blur	2	4	6	8	10	12	14	16	18
TM_CCOEFF		1	1	1	1	1	1	1	1	1
TM_CCOEFF_NORMED		1	1	1	1	1	1	0	0	0
TM_CCORR		-	-	-	-	-	-	-	-	-
TM_CCORR_NORMED		1	0	0	0	0	0	0	0	0
TM_SQDIFF		1	0	0	0	0	0	0	0	0
TM_SQDIFF_NORMED		1	0	0	0	0	0	0	0	0

Tab. 7, Jablka, výsledek masky 4

14.6 Kód programu

```python
import cv2 as cv
import numpy as np
import glob

img_ = cv.imread("D:/Programy/Python/Projekty/ovoce/1.jpg",0)   #nacteni vstupniho obrazu

path = glob.glob("D:/Programy/Python/Projekty/ovoce_masky/*.jpg")   #cesta k vyrezum k sestaveni masky

#sestaveni masky
zero = np.zeros((100, 100))
for img in path:
    a = cv.imread(img,0)
    b = cv.resize(a,(100,100))
    c = b/3
    zero = zero + c
cut = np.array(np.round(zero),dtype='uint8')
cv.imshow('',cut)
cv.waitKey(0)

#metody matchtemplatingu
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
            'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']

#souradnice jablek v obraze a jejich zvirazneni
xy = np.array([ [50,  170, 310, 425,  45,  165, 305, 435,  35,  175, 305, 440,  40,  180, 300, 430],
                [50,  40,  55,  55,   180, 170, 185, 185,  295, 310, 310, 315,  425, 440, 430, 430],
                [160, 290, 415, 545,  150, 285, 410, 555,  160, 290, 425, 545,  165, 285, 420, 545],
                [165, 165, 165, 180,  285, 290, 290, 310,  415, 425, 425, 415,  550, 545, 555, 550]])
img2 = img_.copy()
for i in range(0,len(xy[0])):
    cv.rectangle(img2, (xy[0][i],xy[1][i]), (xy[2][i], xy[3][i]),(0,0,255), 1 )

cr = [2,20,2]   #rozsah pro cyklus rozmazani - blur

#pomocne matice
g = []
h = []
o = []
for meth in methods:    #cyklus metod
    print('-----------------------------------------------------------------------')
    print(meth)
    print(' ')
    b = []
    img3 = img_.copy()
    for j in range(0,len(xy[0])):   #cyklus pro vyrezy obrazu - segmenty
        cropped_ = img_[xy[1][j]:xy[3][j], xy[0][j]:xy[2][j]]
        # cv.imshow('crop',cropped_)
        # cv.waitKey(0)

        #zaplneni prazdnych mist v obraze sumem
```

Obr. 70, Kód programu detekce objektu a optimalizace masky (část 1)

```
        #zaplneni prazdnych mist v obraze sumem
        img3[xy[1][j]:xy[3][j], xy[0][j]:xy[2][j]] = np.random.randint(
            0,255,(xy[3][j]-xy[1][j],xy[2][j]-xy[0][j]),dtype='uint8')

        c = []
        a = []
        for k in range(cr[0], cr[1], cr[2]):    #cyklus rozmazani - blur
            method = eval(meth)

            #vizualni upravy
            cropped = cv.blur(cropped_,(k,k))
            circle = cv.blur(cut, (k, k))
            inv = cv.bitwise_not(circle)

            #matchtemplating
            res = cv.matchTemplate(cropped,circle,method)
            high = cv.matchTemplate(circle,circle,method)
            low = cv.matchTemplate(circle,inv,method)
            if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
                # rozsah metod
                top = high.min()
                bot = low.min()
                min = res.min()

                diff = bot - min
                summ = bot - top

                acc = round(100*diff/summ, 2)   #vypocet procentualni shody

                a.append(acc)
            else:
                # rozsah metod
                top = high.max()
                bot = low.max()
                max = res.max()

                diff = top - bot
                summ = max - bot

                acc = round(100*summ/diff, 2)   #vypocet procentualni shody

                #zapisy do vysledkovych matic
                a.append(acc)
            c.append(k)
        b.append(a)

    #cyklus pro zbytek obrazu (funkce stejne jako cykly vyse)
    e = []
    for k in range(cr[0], cr[1], cr[2]):
        rest= cv.blur(img3,(k,k))
        method = eval(meth)
        res = cv.matchTemplate(rest,circle,method)
        high = cv.matchTemplate(circle,circle,method)
        low = cv.matchTemplate(circle,inv,method)
        if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
            # rozsah metod
```

Obr. 71, Kód programu detekce objektu a optimalizace masky (část 2)

```
                top = high.min()
                bot = low.min()
                min = res.min()

                diff = bot - min
                summ = bot - top

                acc = round(100*diff/summ, 2)
                e.append(acc)
            else:
                top = high.max()
                bot = low.max()
                max = res.max()

                diff = top - bot
                summ = max - bot

                acc = round(100*summ/diff, 2)
                e.append(acc)
        b = np.array(b, dtype=int)

        avrmat = []
        for m in range(0, len(b[0])):   #cyklus pro pocitani vyslednych prumeru
            sum = 0
            for l in range(1,len(b)):
                sum = sum + b[l][m]
            avr = round(sum / (len(b)-1), 2)
            avrmat.append(avr)

        #zapisy do vysledkovych matic
        fin = np.array(np.vstack((c,avrmat, e)))
        f = b.max()
        g.append(f)
        avrmat = np.array(avrmat)
        avrmax = avrmat.max()
        h.append(avrmax)
        e = np.array(e)
        okoli = e.max()
        o.append(okoli)

        #vypisovani do terminalu
        print('max shoda funkce=', f, '%')
        print(' ')
        print(fin)
        print(' ')
        print(b)
        print(' ')

    print('-----------------------------------------------------------------'+
    '-----------------------------------------------------------------')
    print('prumerna maxima funkci', h)
    print('absolutni maxima okoli', o)
    print('absolutni maxima funkci', g)
    g = np.array(g)
    g = g.max()
    print('celkova max shoda=', g, '%')
    print('-----------------------------------------------------------------'+
    '-----------------------------------------------------------------')
```

Obr. 72, Kód programu detekce objektu a optimalizace masky (část 3)

15 ZÁVĚREČNÉ ZHODNOCENÍ

Pro svou bakalářskou práci jsem si vybral téma „Návrh, tvorba a optimalizace masky pro hledání vzorů ve strojovém vidění“. Hlavním cílem a účelem této práce je osvětlit strojové vidění a detekci objektů lidem, kteří se k němu nikdy nedostali. A to jak v psané formě programu, tak vizuální ve formě obrázků. Má sloužit jako jakýsi návod pro začátečníky a berlička ke vzniku jednoduchých aplikací zapálenými samouky.

Pro potřeby řešení problematiky této bakalářské práce je možné využít všechny tři srovnávané programovací jazyky. C++ jako jeden z nejběžnějších a nejpoužívanějších programovacích jazyků, Javu, populární programovací jazyk, který je velice stabilní nebo Python jako hybridní vysokoúrovňový open-source programovací jazyk, který podporuje rozmanitá programovací paradigmata. Z důvodů veřejné licence, jednoduché dostupnosti, množství knihoven, pochopitelnosti, rychlosti práce a pracnosti kódu jsem jako programovací jazyk této práce zvolil Python.

Jako programovací prostředí jsem zvolil Visual Studio Code, a to hlavně kvůli příjemnému designu.

Detekce státní poznávací značky dopadla svou funkčností nad má očekávání. Při vstupu kvalitního obrazu s dobře čitelnou značkou (nejlépe pro čtení statickým fotoaparátem) se správnost čtení po optimalizaci blíží stu procentům. Tuto aplikaci jsem dále samostatně rozšířil o program rozpoznávající jednotlivé znaky SPZ pomocí tří sad masek. Jedna sada se bohužel ukázala jako nevyhovující. Zbylé dvě sady masek už byly využitelnější, SPZ dokáží „přečíst“ s přesností přibližně 65 %.

Rozpoznání vícero podobných předmětů v jednom obrazu jako například ovoce na stromě nebo obsazená místa na pozemním parkovišti a jejich spočítání je dle mého názoru o úroveň těžší problematika. První část, která se touto problematikou zabývá, je spíše ukázková a má čtenáře seznámit s fungováním funkce „matchTemplate“. Druhá část má za úkol co nejvíce přiblížit postupy návrhu masky a její optimalizace.

Návrh a tvorba masky není jednoduchý proces, každá aplikace si ho žádá lehce jiný. Při tomto procesu je stěžejní, zaměřit se na charakteristickou vlastnost. Například u rozpoznání znaků SPZ je charakteristický kontrast barev značky a generické znaky. Optimální maska je tedy taková, která bude mít vysokou procentuální shodu. Optimalizace a tvorba takovéto aplikace si žádá čas a zkušenosti. Doufám, že principy těchto pojmů jsou z této práce pochopitelné a užitečné pro všechny čtenáře této práce.

16 POUŽITÁ LITERATURA

[1] ŽÁRA, Jiří, Beneš Bedřich, Sochor Jiří, Felkel Petr. Moderní počítačová grafika. 2. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0

[2] BURIAN, Peter K. a Robert CAPUTO. Škola fotografování: National Geographic: příručka do kapsy. Praha: Sanoma Magazines Praha, [2003]. ISBN 80-702-6253-2.

[3] LIPPMAN, Stanley B., Josée LAJOIE a Barbara E. MOO. C primer. 5th ed. Upper Saddle River: Addison Wesley, 2013. ISBN 978-0-321-71411-4.

[4] NOVOTNÝ, Luděk. Historie a vývoj jazyka Java [online]. Brno, 2003 [cit. 2022-05-31]. Dostupné z: https://www.fi.muni.cz/usr/jkucera/pv109/2003p/xnovotn8.htm. Článek. MUNI.

[5] General Python FAQ. Python [online]. Python Software Foundation, 2022 [cit. 2022-05-31]. Dostupné z: https://docs.python.org/3/faq/general.html#general-information

[6] C++ vs Java vs Python?. Tutorialspoint [online]. Hyderabad: Tutorials Point (India), 2022 [cit. 2022-05-31]. Dostupné z: https://www.tutorialspoint.com/cplusplus-vs-java-vs-python

[7] Nejoblíbenější programovací jazyky současnosti a jejich budoucí trendy. CoolClub [online]. Praha: CoolPeople, 2020 [cit. 2022-05-31]. Dostupné z: https://club.coolpeople.cz/nejoblibenejsi-programovaci-jazyky-soucasnosti-a-jejich-budouci-trendy/1372.html

[8] Screen resolution. PCMag [online]. New York: PCMag, 2022 [cit. 2022-06-01]. Dostupné z: https://www.pcmag.com/encyclopedia/term/screen-resolution

[9] OpenCV 3.x with Python By Example: Make the most of OpenCV and Python to build applications for object recognition and augmented reality. 2nd Edition. Birmingham: Packt Publishing, 2018. ISBN 1788396901.

[10] Základní formáty obrázků. Soom.cz [online]. SOOM.cz, 2022 [cit. 2022-06-01]. Dostupné z: https://www.soom.cz/clanky/988--Zakladni-formaty-obrazku

[11] The Best Python IDEs and Code Editors (According to Our Developers and the Python Community). STXNEXT [online]. Poznań: STX Next S.A., 2022 [cit. 2022-06-01]. Dostupné z: https://www.stxnext.com/blog/best-python-ides-code-editors/

[12] Top 8 Image-Processing Python Libraries Used in Machine Learning. Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python-libraries-for-machine-learning

[13] Image Processing in Python: Algorithms, Tools, and Methods You Should Know. Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[14] Rozdíl v zobrazení při přiblížení vektorového a rastrového obrázku. In: STARGEN [online]. Praha: STARGEN GROUP, 2018 [cit. 2022-06-01]. Dostupné z: https://www.stargen.cz/slovnik/vektorova-grafika/

[15] Compare-resolution. In: Smarty [online]. Praha: Smarty CZ, 2022 [cit. 2022-06-01]. Dostupné z: https://www.smarty.cz/Slovnik/rozliseni-druheho-displeje-a906

[16] Java-companies-1. In: CODINGNOMADS [online]. Brno: ENGETO, 2022 [cit. 2022-06-01]. Dostupné z: https://engeto.cz/blog/kariera/proc-se-stat-java-developerem/

[17] Features-of-python. In: All About Python [online]. 2020 [cit. 2022-06-01]. Dostupné z: https://www.allaboutpython.com/2020/12/introduction-to-python-language.html

[18] Popularity-of-environments-and-tools-among-Python-developers-chart. In: STXNEXT [online]. Poznań: STX Next S.A., 2022 [cit. 2022-06-01]. Dostupné z: https://www.stxnext.com/blog/best-python-ides-code-editors/

[19] Opengraph-home. In: Visual Studio Code [online]. Redmond: Microsoft Corporation, 2022 [cit. 2022-06-01]. Dostupné z: https://code.visualstudio.com/

[20] Colors-extracted-effect-1024x271. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python-libraries-for-machine-learning

[21] Square-structuring-2. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[22] Dilation. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[23] Erosion. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[24] Gaussain. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[25] 1*lmqjFx1R0EeWVJ3tks4rPg. In: Towards Data Science [online]. Tonichi Edeza, 2021 [cit. 2022-06-01]. Dostupné z: https://towardsdatascience.com/image-processing-with-python-application-of-fourier-transformation-5a8584dc175b

[26] Edge-Detection. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[27] Wavelet-Image-Processing-1-1. In: Neptuneblog [online]. Neptune Labs, 2022 [cit. 2022-06-01]. Dostupné z: https://neptune.ai/blog/image-processing-python

[28] Jablko-vis%C3%AD-na-strome-300x200. In: PEKNETELO.EU [online]. Zdravi.peknetelo.eu, 2018 [cit. 2022-06-01]. Dostupné z: https://zdravi.peknetelo.eu/ovoce-pro-zdravi/jablko/

[29] Apple.com [online]. Cupertino: Apple, 2022 [cit. 2022-06-01]. Dostupné z: https://support.apple.com/cs-cz/HT208108

[30] R696x392/62a4f704-c5f4-4499-9472-54eada2fd744. In: TN.cz [online]. Praha: TN.cz, 2012 [cit. 2022-06-03]. Dostupné z: https://tn.nova.cz/zpravodajstvi/clanek/262227-zlodeji-tankuji-zadarmo-na-vasi-spz-jak-se-branit-problemum

[31] Jak-najit-majitele-auta-podle-registracni-znacky-spz. In: Portál řidiče [online]. Pardubice: DF SOLUTIONS, 2021 [cit. 2022-06-03]. Dostupné z: https://www.portalridice.cz/clanek/jak-najit-majitele-auta-podle-spz

[32] Vyhledavac-spz. In: Portál řidiče [online]. Pardubice: DF SOLUTIONS, 2021 [cit. 2022-06-03]. Dostupné z: https://www.portalridice.cz/clanek/vyhledavac-spz-aneb-vyhledavani-podle-registracni-znacky-vozu

[33] OpenCV Tutorials. OpenCV [online]. Opencv.org, 2022 [cit. 2022-06-03]. Dostupné z: https://docs.opencv.org/4.x/d9/df8/tutorial_root.html

[34] NumPy user guide. NumPy [online]. NumPy Developers, 2022 [cit. 2022-06-03]. Dostupné z: https://numpy.org/doc/stable/user/index.html

[35] EasyOCR. Python Package Index [online]. Python Software Foundation, 2022 [cit. 2022-06-03]. Dostupné z: https://pypi.org/project/easyocr/

[36] 87033990_l. In: SPORTSDIRECT [online]. praha: Frasers group, 2022 [cit. 2022-06-03]. Dostupné z: https://cs.sportsdirect.com/slazenger-6pk-foam-balls-870339

[37] Kamery pro rozpoznávání obličejů zažívají celosvětový rozmach, Čína už jich nasadila 200 milionů. Česká televize [online]. Praha: Česká televize, 2019 [cit. 2022-06-03]. Dostupné z: https://ct24.ceskatelevize.cz/veda/2991737-kamery-pro-rozpoznavani-obliceju-zazivaji-celosvetovy-rozmach-cina-uz-jich-nasadila-200

[38] SPZ na přání. In: Peníze.cz [online]. Praha: NextPage Media, 2022 [cit. 2022-08-10]. Dostupné z: https://spz.penize.cz/spz-na-prani

[39] 29344-font-pro-spz-cr. In: SLUNEČNICE.cz [online]. Praha: Internet Info, 2009 [cit. 2022-08-10]. Dostupné z: https://www.slunecnice.cz/sw/font-pro-spz-cr/

[40] Trinact-druhu-jablek. In: Globus [online]. Praha: Globus, 2020 [cit. 2022-08-11]. Dostupné z: https://www.globus.cz/blog/clanky.html/355_18661-trinact-druhu-jablek-ktere-byste-meli-znat

17 SEZNAMY

17.1 Seznam obrázků

17.2 Seznam rovnic

17.3 Seznam tabulek

18 PŘÍLOHY

- Instalační aplikace Visual Studio Code
- Instalační aplikace Tesseract OCR
- Veškeré programované soubory.py
- Veškeré testované obrázky