
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# ROZPOZNÁVÁNÍ OBJEKTŮ GRAFICKÉHO UŽIVATELSKÉHO ROZHRANÍ PRO VIZUÁLNÍ TESTOVÁNÍ

**WIDGET RECOGNITION FOR VISUAL GUI TESTING**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** — **SEBASTIÁN KISELA**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. ALEŠ SMRČKA, Ph.D.**
**SUPERVISOR**



**BRNO 2018**

*Zadání bakalářské práce/20304/2017/xkisel02*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav inteligentních systémů Akademický rok 2017/2018

# Zadání bakalářské práce

Řešitel: **Kisela Sebastián**
Obor: Informační technologie
Téma: **Rozpoznávání objektů grafického uživatelského rozhraní pro vizuální testování**
**Widget Recognition for Visual GUI Testing**
Kategorie: Analýza a testování softwaru

Pokyny:

1. Seznamte se se vzdálenou správou desktopu VNC. Nastudujte současné metody rozpoznávání vzorů a rozpoznávání komponent grafického uživatelského rozhraní (GUI).
2. Analyzujte vlastnosti rozpoznávacích algoritmů. Na základě analýzy vyberte s vedoucím projektu typ ovládacích komponent GUI pro automatické rozpoznávání. Navrhněte algoritmus rozpoznání vybraných ovládacích komponent GUI.
3. Implementujte program pro automatické rozpoznávání ovládacích komponent GUI. Program bude zařezen jako komponenta platformy Testos.
4. Ověřte úspěšnost rozpoznávání ovládacích prvků na umělé testovací sadě zahrnující různé styly daných ovládacích prvků. Ověřte úspěšnost algoritmu na reálné aplikaci.

Literatura:

- Domovská stránka platformy Testos. http://testos.org/
- E. Alégroth, R. Feldt, L. Ryrholm. 2014. Visual GUI testing in practice: challenges, problems and limitations. In Empirical Software Engineering. Vol. 20, Issue 3. doi: 10.1007/s10664-013-9293-5

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
*vedoucí ústavu*

# Abstrakt

Bakalárska práca sa sústredí na analýzu súčasných metód používaných pri testovaní grafického používateľského rozhrania, ako aj nástrojmi na to použitými. Stanovuje špecifikácie požiadavkov a návrh metód rozpoznávania grafického používateľského rozhrania. Neskôr sú v práci podrobne popísané implementácia a obmedzenia implementovaného riešenia postavenom na vizuálnom rozpoznávaní na základe pixelov. V závere sú ukázané spôsoby, ktorými bola funkčnosť riešenia overovaná.



# Abstract

The bachelor thesis analyzes current methods and tools used for user interfaces testing. The thesis sets specification requirements and design of the implemented tool. Implementation and limitations of the implemented tool, which is based on visual object recognition, are explained in a detailed way. Verification process of the tool is described at the end.



# Kľúčové slová

Testovanie grafického užívateľského rozhrania, automatizácia testovania, počítačové videnie, spracovanie obrazu

# Keywords

Visual Graphical User Interface testing, automated testing, computer vision, image preprocessing


# Citácia

KISELA, Sebastián. *Rozpoznávání objektů grafického uživatelského rozhraní pro vizuální testování.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

# Rozpoznávání objektů grafického uživatelského rozhraní pro vizuální testování

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Smrčku Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Sebastián Kisela
16. mája 2018


## Poďakovanie

Chcem poďakovať pánovi Ing. Smrčkovi Ph.D. za odborné rady, osobný prístup a pútavý spôsob vedenia bakalárskej práce.

# Obsah


**1 Úvod** **3**

**2 GUI a aktuálne metódy rozpoznávania objektov** **5**
2.1 Knižnica OpenCV . . . . . 5
2.1.1 Vyhľadávanie vzoru . . . . . 6
2.1.2 Detekcia hrán . . . . . 7
2.1.3 Detekcia kružnice . . . . . 9
2.1.4 Dilatácia a Erózia . . . . . 10
2.1.5 Získavanie kontúr . . . . . 10
2.2 Vizuálne testovanie . . . . . 11
2.2.1 Prenositeľnosť . . . . . 12
2.2.2 Sikuli . . . . . 12
2.3 Nástroje so znalosťou objektov . . . . . 14
2.3.1 Prenositeľnosť . . . . . 14
2.3.2 Selenium . . . . . 14
2.4 Metoda zaznamenaj a prehraj . . . . . 16

**3 Špecifikácia požiadaviek a návrh metody rozpoznávania komponent GUI** **17**
3.1 Nefunkcionálne požiadavky . . . . . 17
3.2 Špecifikácia funkcionálnych požiadaviek . . . . . 18
3.3 Návrh metódy rozpoznávania komponent . . . . . 19
3.3.1 Typické vlastnosti komponent GUI . . . . . 20
3.3.2 Zásuvné moduly . . . . . 21

**4 Implementačné detaily rozpoznávania GUI** **23**
4.1 Konfigurácia . . . . . 23
4.1.1 Konfiguračný súbor . . . . . 23
4.2 Detekcia . . . . . 28
4.2.1 Definovanie vzťahov medzi detekovanými objektami . . . . . 30
4.2.2 Odstraňovanie prebytočných objektov . . . . . 31
4.2.3 Obmedzenia detekcie kontúr . . . . . 34
4.3 Získavanie informácií o detekovaných objektoch . . . . . 35
4.3.1 Zarovnanie objektu . . . . . 35
4.3.2 Farba pozadia objektu . . . . . 36
4.3.3 Plocha objektu oproti rodičovskému objektu . . . . . 36
4.4 Klasifikácia . . . . . 37
4.4.1 Hodnota a interval klasifikácie . . . . . 37
4.4.2 Postup klasifikácie dodaného zásuvného modulu . . . . . 39

4.4.3 Generovanie výstupu . . . 42
4.5 Komparácia . . . 42
4.5.1 Porovnanie JSON výstupu . . . 42
4.5.2 Porovnanie vstupných obrázkov . . . 42

**5 Overenie funkcionality rozpoznávania** **43**
5.1 Testy na aplikácií používajúcej Bootstrap . . . 43
5.1.1 Nepresnosti klasifikácie . . . 44
5.1.2 Zhodnotenie . . . 46
5.2 Testy na aplikácií GitHub . . . 46
5.2.1 Nepresnosti klasifikácie . . . 47
5.2.2 Zhodnotenie . . . 49
5.3 Testovacia zostava . . . 49

**6 Závěr** **50**

**Literatúra** **51**

**A Používateľská príručka a API** **53**
A.1 Manipulácia so zásuvnými modulmi . . . 53
A.1.1 Inštalácia zásuvného modulu . . . 53
A.1.2 Povinný obsah zásuvného modulu . . . 54
A.2 Rozhranie triedy Wirec . . . 55
A.3 Rozhranie pre Testos Bus . . . 57
A.3.1 org.testos.wirec.run . . . 57
A.3.2 org.testos.wirec.reclassify . . . 58
A.3.3 org.testos.vnc.run . . . 58

**B Demonštrácia použitia aplikácie** **59**
B.1 Závislosti . . . 59
B.1.1 Operačný systém Fedora/CentOS/RHEL . . . 59
B.1.2 Operačný systém Debian/Ubuntu . . . 59
B.1.3 Závislosti na Python balíkoch . . . 59
B.2 Možnosti inštalácie . . . 59
B.2.1 Virtuálne prostredie . . . 59
B.2.2 Manuálna inštalácia . . . 60
B.3 Použitie . . . 60
B.3.1 Príklad vygenerovaného JSON súboru . . . 61

**C Bootstrap snímky** **63**

# Kapitola 1

# Úvod

Cieľom bakalárskej práce je zoznámiť sa so súčasnými nástrojmi a metódami spracovania obrazu, tiež s metódami používanými na testovanie používateľských rozhraní aplikácií. Ďalším zo zameraní práce je špecifikácia požiadavkov a návrh implementácie nástroja schopného analyzovať zadaný obrazový vstup na účely získania maximálneho možného množstva informácií o obsiahnutých prvkoch. Tieto informácie sú použité na vlastnú klasifikáciu konfigurovateľnú koncovým používateľom nástroja, ako aj pre ďalšie nástroje, ktoré majú iné zameranie a sú schopné získané informácie interpretovať na svoje účely.

Práca v úvode jednoduchým spôsobom opisuje princípy existujúcich metód, ktoré sú používané na testovanie používateľských rozhraní a rozpoznávanie obsahu obrazu. Zameriava sa na metódy prístupu k interným datovým štruktúram uchovávajúcim stav a vlastnosti používateľského rozhrania a taktiež porovnáva ich prednosti a nedostatky. Týmito prístupmi sú napríklad Template Matching a Vizuálne Testovanie grafického používateľského rozhrania a nástroje ako Selenium či Sikuli. Nedostatky spomenutých metód vytvárajú priestor pre opisovaný vytvorený nástroj.

Autor sa neskôr podrobne zaoberá návrhom a špecifikáciou algoritmov potrebných na spracovanie obrazu, ktorý je vstupom pre vytvorený nástroj. Skúmanými vstupmi sú rôzne snímky celých používateľských rozhraní webového frameworku Bootstrap a tiež použitie na reálnej komerčnej aplikácií GitHub.

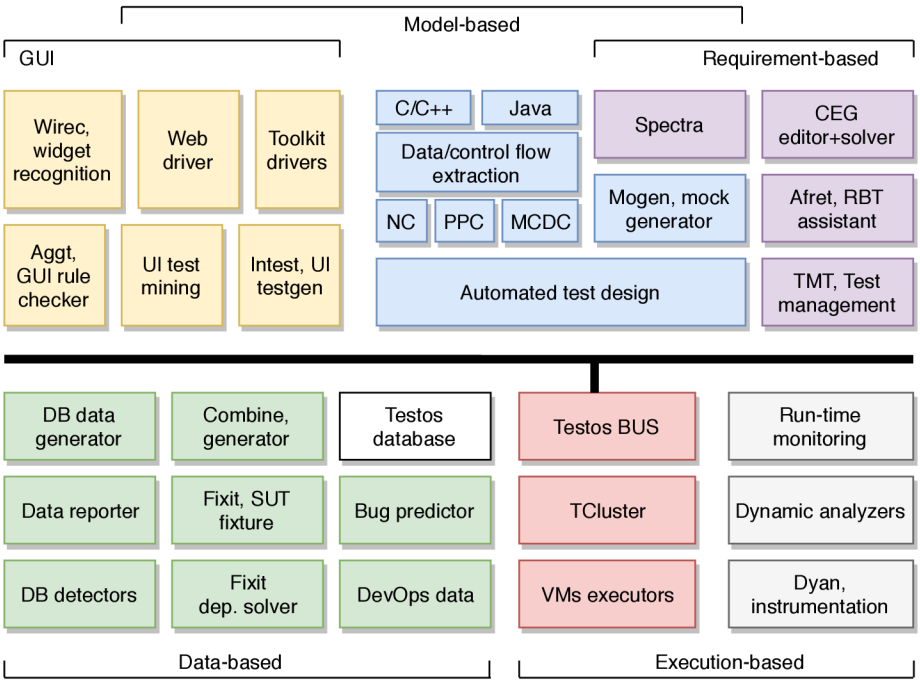
Podrobným spôsobom je vysvetlený postup zvolený pre analýzu obrazu s použitím Open Source knižnice Open Computer Vision (OpenCV), ktorá je vhodná pre preprocessing obrazu ako aj jeho analýzu. Funkcionalita je overená umelo vytvorenou testovaciou sadou , ako aj použitím na reálnych aplikáciách s následnou manuálnou analýzou získaných výsledkov.

Motiváciou k implementácií je vytvorenie prenositeľného riešenia testovania grafického používateľského rozhrania (v texte bude požívaná skratka GUI z anglického Graphical User Interface) jednoducho použiteľného pre účely iných projektov. Implementovaný nástroj bude zaradený do projektu Testos, ktorého architektúra je ukázaná na obrázku 1.1.

Testos (Test Tool Set) [6] je projekt, ktorého hlavným cieľom je vytvorenie sady nástrojov podporujúcich automatizované testovanie software. Nástroje v platforme Testos kombinujú rôzne úrovne testovania a je možné ich zaradiť do niekoľkých kategórií: testovanie založené na modeloch (Model-based), testovanie založené na požiadavkoch (Requirement-based), testovanie grafického používateľského rozhrania (GUI), testovanie založené na dátach (Data-based) a dynamická analýza (Execution-based).

V aktuálnom vývoji nástrojov pre testovanie GUI je nástroj pro tvorbu testovacej sady splňujúcej kritériá pokrytia grafických prvkov [14], nástroj pre rozpoznanie objektov GUI [8] a nástroj pre kontrolu pravidiel GUI [7].

Druhá kapitola vysvetľuje algoritmy použité na vizuálne rozpoznávanie obrazu pri použití knižnice OpenCV a tiež súčasné nástroje používané na vizuálne testovanie. Tretia kapitola popisuje návrh a požiadavky stanovené pre implementáciu nástroja. Štvrtá kapitola podrobne popisuje implementáciu a možnosti konfigurácie nástroja a v poslednej kapitole je ukázaný a komentovaný výstup kompletného implementovaného riešenia.




Obr. 1.1: Schéma Testos. Prevzaté z [6].

# Kapitola 2

# GUI a aktuálne metódy rozpoznávania objektov

V súčasnej dobe sa GUI objavujú v rôznych formách a zariadeniach. Zaručenie funkcionality GUI v každej chvíli a tiež ďalšie podmienky potrebné pre ideálny zážitok používateľa sú ešte stále náročnými úlohami pre programátora.

Donedávna jediným spôsobom testovania GUI a zaručovania ich funkčnosti bolo manuálne testovanie samotným programátorom, či iným človekom určeným na túto činnosť. Tento prístup však obsahuje príliš mnoho nedostatkov na to, aby bol dlhodobo udržateľný pri súčasných snahách o automatizáciu maximálneho možného množstva úkonov vykonávaných manuálne.

Hlavnými nedostatkami sú napríklad nemožnosť skutočne zamedziť regresiám v GUI, keďže pri robustných aplikáciách zabezpečiť všetky možné akcie vykonateľné používateľom a tým zaručiť, že nevznikli žiadne nové chyby, je dlhodobo nemožné. Existuje však viacero nástrojov schopných automatizácie testovania GUI do istej miery.

Druhy automatizovaného testovania GUI sa radia do troch kategórií:

- Nástroje so znalosťou objektov - nástroje, ktoré na automatizáciu používajú rozpoznávanie obrazu(Sikuli..).
- Vizuálne testovanie - nástroje, ktoré poznajú a majú prístup k vnútornej štruktúre GUI, poznajú názvy grafických prvkov ako aj ich funkcionalitu (Selenium..).
- Record and Replay - nástroje dokonale imitujúce akcie používateľa.

## 2.1 Knižnica OpenCV

OpenCV je používaný na jednu z metód automatizácie testovania GUI, ktorá je založená na vyhľadávaní zhody tzv. Template obrazu v tzv. Source obraze. Teda vyhľadávanie vzoru v prehľadávanom GUI, tzv. Template Matching.

Používa sa primárne prístup založený na vyhľadávaní vzoru resp. oblasti tzv. Area/Template Based, na rozdiel prístupu založenom na vlastnostiach vzoru tzv. Feature Based. Tento prístup je inovatívny v oblasti počítačového videnia a pokročilé podoby tohoto algoritmu sú schopné rozpoznávať hľadaný vzor dokonca nezávisle na jeho orientácií a globálneho jasu.

Prístup založený na vyhľadávaní zhody s hľadaným vzorom v obraze je používaný nástrojmi ako Sikuli, Eyetomate a jeho funkcionalita je tiež vo veľkej miere používaná v ná-

stroji, ktorý je výsledkom tejto práce so skratkou - Wirec, preto je potrebné túto OpenCV funkcionalitu viac priblížiť.

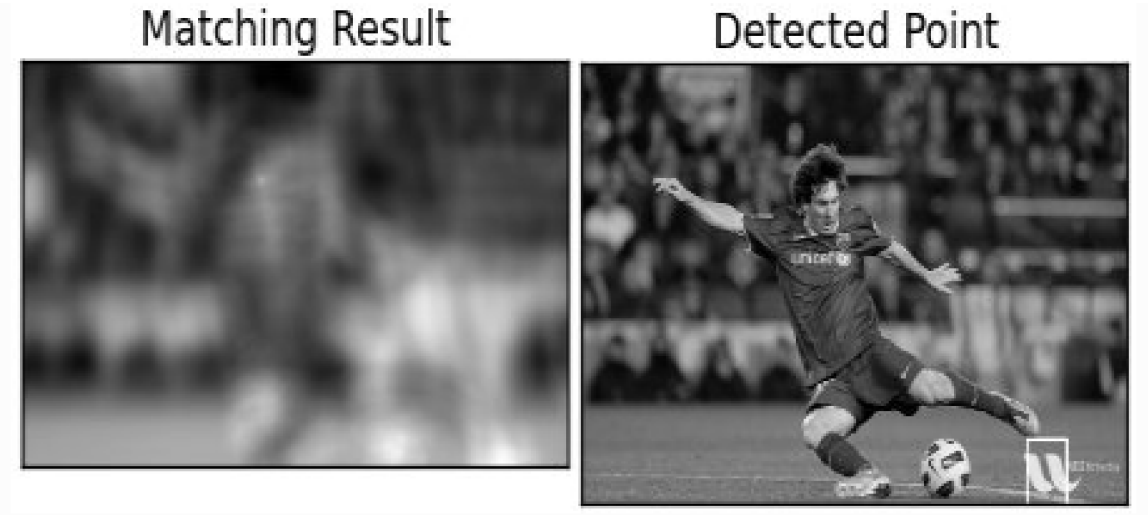
### 2.1.1 Vyhľadávanie vzoru

V tejto kapitole bude ukázaná iba malá časť používaných prístupov vyhľadávania vzoru, pomocou naivného a normalizovaného vyhľadávania na základe korelačného koeficientu.

#### Naivné vyhľadávanie vzoru

Metóda naivného vyhľadávania [11] prehľadáva zdrojový obraz v každej možnej pozící porovnaním dvojíc pixelov z daných oblastí v každom možnom mieste. Zhoda je vyhodnotená použitím jednoduchej korelácie medzi obrazmi. Naivné vyhľadávanie je veľmi náchylné na chyby, pretože aj vzorový obraz, ktorý je málo podobný zdrojovému, môže byť vyhodnotený ako veľmi korelovaný, keďže obraz s vysokým jasom môže byť kvôli vysokým hodnotám pixelov vyhodnotený ako veľmi podobný.

Obr. 2.1: Vzorový obraz. Prevzaté z [5].




Obr. 2.2: Výsledok naivného vyhľadávania. Prevzaté z [5].

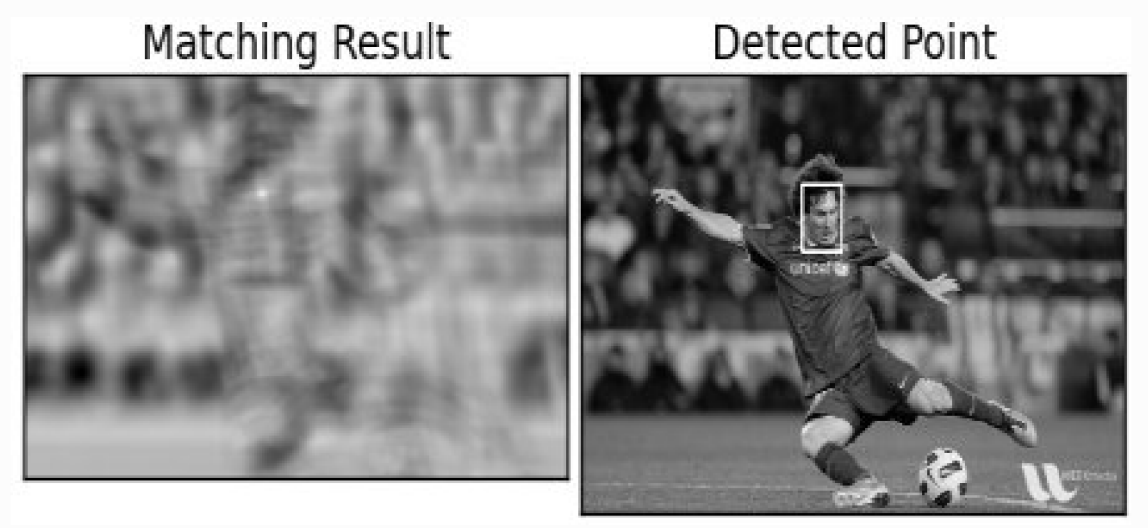
Na obrázku je vidieť, že oblasť loga v pravom dolnom rohu zdrojového obrazu, je najviac korelovaná, pretože je výrazne jasnejšia, ako ostatné potenciálne vhodné oblasti.

#### Normalizované vyhľadávanie vzoru

Zlešenie výsledkov naivného vyhľadávanie je možné dosiahnuť ich normalizovaním. Pri korelácií obrazov je od každého pixelu obrazu odčítaný priemerný jas obrazu. Takáto úprava spôsobí, že korelované obrazy sú nezávislé na globálnom jase obrazu. Výsledná hodnota

korelácie medzi obrazmi je udávaná v intervale <-1, 1>, kde zhodné obrazy sú korelované s hodnotou 1.0 a obraz s negáciou toho istého obrazu je korelovaný s hodnotou -1.0.

Obr. 2.3: Vzorový obraz. Prevzaté z [5].




Obr. 2.4: Výsledok normalizovaného vyhľadávania. Prevzaté z [5].

Na obrázku 2.4 je po normalizácií v tejto chvíli vidieť správne detekovaný vzorový obraz.

#### Identifikácia zhody

Korelácia vzorového a zdrojového obrazu môže byť na viacerých miestach zdrojového obrazu relatívne vysoká, preto je dôležité zvoliť prah prijateľnosti zhody, napr. zhoda vyššia ako hodnota 0.5. Ďalším spôsobom je zvolenie lokalne maximálne zhody, teda zhody najvyššej v súsediacich pixeloch.

#### Pokročilé metódy

Zložitosť výpočtu predošlích operácií vo väčších rozmeroch obrazov rastie a obrazy musia medzi sebou zachovávať rotáciu a mieru. Pokročilé metódy používajú rôzne heuristiky, ktoré sú schopné nájsť zhodu aj napriek rotácií.

### 2.1.2 Detekcia hrán

Ďalšou dôležitou metódou používanou pri manipulácií s obrazom na úrovni pixelov je detekcia hrán. Toto je uskutočnené analýzou obrazu v smere x a y maticovým násobením jadra a aktuálneho výrezu obrazu [9].

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

Gx

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Gy

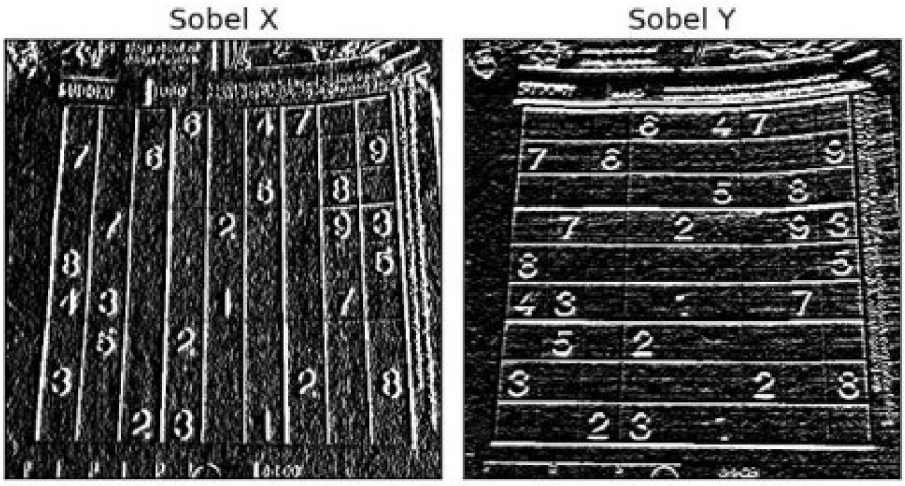
Obr. 2.5: Sobel Operator. Prevzaté z [9].

Výsledkom sobel operátoru by boli dva obrazy, prvý by mal vykreslené vertikálne hrany, druhý horizontálne. Ich spojením je vykreslený obraz obsahujúci iba hrany. Pre vyhnutie sa dvojitému násobeniu celeho obrazu maticami je používaný Laplace Operator, ktorý prechádza obraz len raz s nasledujúcim jadrom:

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

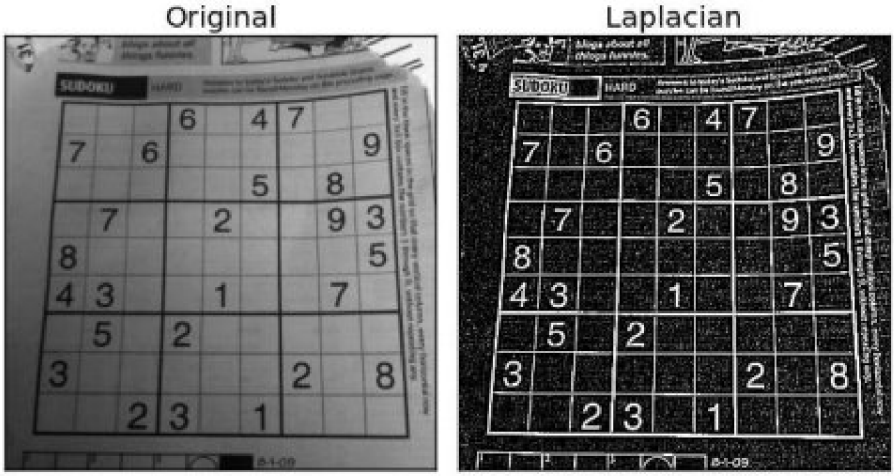
Obr. 2.6: Snímok jadra Laplacian operátora. Prevzaté z [4].

Príkladom Sobel detekcie je detekcia hrán na nasledujúcom obrázku:




Obr. 2.7: Snímok detekovaných hrán metodou Sobel. Prevzaté z [4].

Na nasledujúcom obrázku sú jasne viditeľné miesta, kde je kontrast vysoký a to sú miesta, v ktorých budú aj hrany vykreslené ako silné:




Obr. 2.8: Snímok detekovaných hrán Laplacian operátorom. Prevzaté z [4].

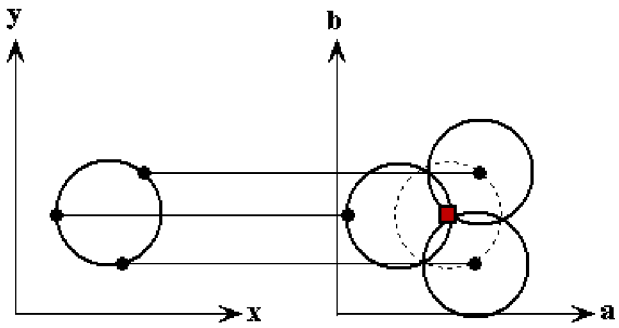
Technika detekcie hrán je používaná na rôzne účely. Pre nástroj Wirec je absolútne kritická. OpenCV túto funkcionalitu sprístupňuje cez použitie funkcií `Sobel()` a `Laplacian()`, ktoré prijímajú argumenty ako sú veľkosť jadra použitého na detekciu hrán, reprezentácia výstupného obrazu - farebný, grayscale, či čiernobiely.

### 2.1.3 Detekcia kružnice

Ďalšiou dôležitou metódou použitou pri tvorbe nástroja Wirec je detekcia kružnice. V pozadí OpenCV metódy `cvHoughCircles()` je prehľadávaný obrázok prevedený na bezfarebný a následne sú v ňom detekované hrany. Prvý postup je vykonávaný v prípade, že je známy polomer hľadanej kružnice. Na každej z detekovaných hrán je vykonaný nasledovný algoritmus.

- Vykreslenie kružnice s daným polomerom.
- Nájdenie priesečníkov kružníc.
- Priesečník zložený z najviac prienikov je považovaný za stred hľadanej kružnice.

Praktická ukážka algoritmu je na obrázku 2.9.

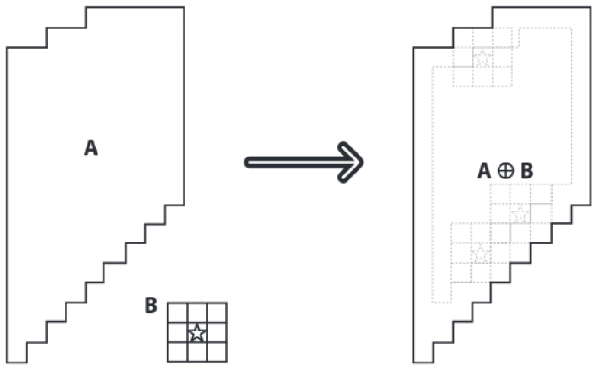



Obr. 2.9: Detekcia kružnice 1. Prevzaté z [16].

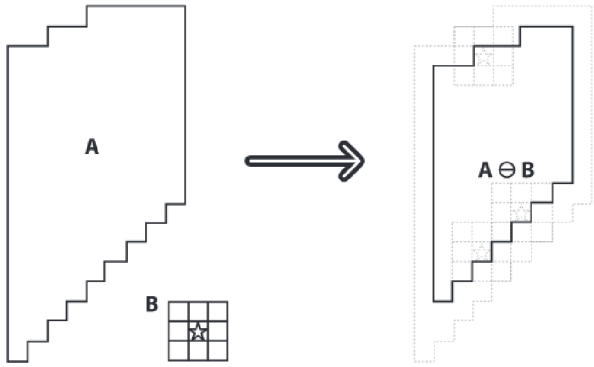
Druhým prípadom je hľadanie kružnice s neznámym polomerom. Tento postup je výpočetne náročnejší, keďže je potrebné hľadať možné kružnice s každým možným polomerom zo zadaného intervalu.

### 2.1.4 Dilatácia a Erózia

Dilatácia a Erózia sú používané metody pri spracovaní obrazu. Ide o spracovanie obrazu pomocou konvolúcie. Pri oboch postupoch je použitá aplikácia jadra na zdrojový obrázok. Jadro obsahuje centrálny pixel, a pri prechádzaní jadra zdrojovým obrazom pri dilatácií je počítané lokálne maximum. Pri Erózií je efekt opačný. Táto funkcionalita je ukázaná na nasledujúcom obrázku.




Obr. 2.10: Dilatácia. Prevzaté z [10].



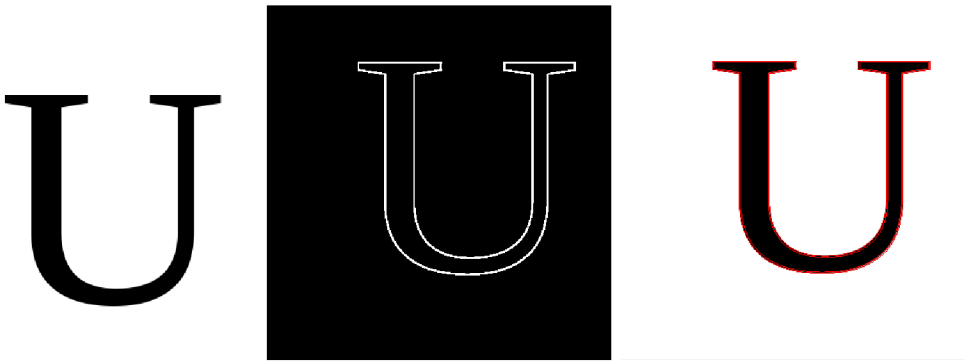

Obr. 2.11: Erózia. Prevzaté z [10].

### 2.1.5 Získavanie kontúr

Kontúry v kontexte OpenCV sú spojité krivky ohraničujúce objekt. Teda kontúrami štvorca sú krivky, ktoré ho obkresľujú. OpenCV poskytuje funkciu `findContours()`, ktorá zo vstupného obrazu analyzuje objekty a vráti jednotlivé zoznamy súradníc kontúr, ktoré dané objekty obkresľujú. Pre minimalizáciu redundantných súradníc vynecháva súradnice kontúr, ktoré sú obsiahnuté v dlhšej existujúcej krivke prechádzajúcej danými bodmi.

Funkcia `findContours()` vracia taktiež strom vzťahov medzi jednotlivými detekovanými kontúrami objektov. Tieto informácie Wirec nepoužíva, z neskôr vysvetlených dôvo-

dov. Príkladom detekcie hrán a následného získania kontúr sú nasledujúce obrázky písmena "U". Po detekcií hrán sú jeho obrysy obkreslené červenou farbou na znázornenie bodov, ktoré boli získané ako súradnice kontúr.



(a) Písmeno U.

(b) Detekované hrany písmena U.

(c) Vykreslené kontúry písmena U.

Obr. 2.12: Získavanie kontúr.

Detailnejším pohľadom na obrázky je ale vidieť dvojitú kontúru detekovanú na jednom objekte. Toto je očakávané správanie funkcie `findContours()`, ktorá štandardne hľadá všetky kontúry, teda aj vnútorné kontúry, nazývané "holes". Takéto zdvojenie je možné eliminovať použitím parametru `RETR_EXTERNAL`, ktorý spôsobí získanie len externých kontúr a ignorovanie všetkých vnútorných kontúr daného objektu.

(a) Detail dvojitej kontúry.

(b) Detail externej kontúry.

Obr. 2.13: Detaily kontúr.

Nástroj Wirec používa verziu `findContours()` s detekciami všetkých kontúr, pretože okrem iného závisí vo veľkej miere aj na hierarchických vzťahoch detekovaných objektov GUI.

## 2.2 Vizuálne testovanie

Vizuálne tesovacie nástroje pracujú na úrovni analýzy pixelov obrazu. Pre manipuláciu s obrazom rôzne nástroje so znalosťou objektov používajú spomenutú knižnicu OpenCV, primárne jej implementované metódy `cvtColor()` pre prevod medzi rôznymi formátmi obrazu, vyššie podrobné rozpísaný `matchTemplate()` a ďalšie.

### 2.2.1 Prenositeľnosť

Keďže nástroje vizuálneho testovania nijakým spôsobom nezasahujú do vnútornej štruktúry GUI, nie je problém použiť tieto nástroje ani po zásadných zmenách vnútornej štruktúry GUI.

### 2.2.2 Sikuli

Princíp nástroja Sikuli [13], implementovaného v jazyku Java, funguje na základe tzv. visual workflow. Ide o postupnosť akcií štandardne vykonávaných nad GUI. Príkladom môže byť nasledovná postupnosť príkazov:

- Spustenie GUI.
- Kliknutie na text input field.
- Stisknutie viacerých kláves - písanie textu.
- Kliknutie tlačítka vyhľadať.
- Prehľadanie výstupu programu pomocou GUI.

Sikuli poskytuje možnosť vytvárania testovacích prípadov pomocou pseudo-record and replay prístupu. Používateľ vykonáva akcie a Sikuli vytvára záznam jeho akcií pomocou ukladania vykonaných akcií, ukladaním častí GUI ako malých obrazov neskôr použitých na Template Matching pre zisťovanie oblastí, nad ktorými má daná akcia byť vykonána. Sikuli potrebuje na svoj priebeh bežiaci display, alebo display vo virtuálnej podobe.

#### Princíp Sikuli vyhľadávania vzoru

Porovnávanie obrazov v Sikuli je vykonávané pomocou knižnice OpenCV použitím funkcie matchTemplate, ktorej algoritmus bol popísaný vyššie. Výsledky zhody sú funkciou matchTemplate vyhodnocované v intervale <0, 1>. Štandardne Sikuli považuje hodnotu 0.8 za dostatočnú zhodu, no pre presnejšie zhody je odporúčané používať hodnoty vyššie ako 0.95.

#### Problematika časovania

Niektoré akcie GUI vyžadujú na vykonanie dlhší časový interval. Sikuli tento problém rieši pozastavením vykonávania testovacieho scenára na čas, kým daná oblasť GUI neobsahuje určitý vzor, ktorý je v danej oblasti periodicky vyhľadávaný.

Príkladom použitia môže byť nasledujúci úryvok Sikuli kódu [13]:

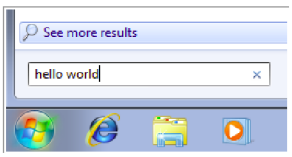
```
# some top left part of the screen
aRegion = Region(0, 0, 500, 500)
# a png image file on the file system
# this is the image we want to look for in the given Region
aImage = 'someImage.png'
# search and get the result
aMatch = aRegion.find(aImage)
```

Teda je možné po vykonaní akcie čakať na zmenu vzhľadu GUI podobnú zadanému vzoru. Ak je zhoda vzorového obrazu a zdrojového obrazu dostatočná, vykonávanie testovacieho scenáru pokračuje. V opačnom prípade je proces vnútorne uspaný na určitý čas, po

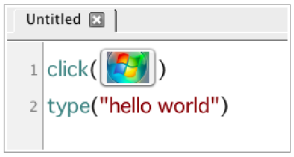
ktorom je zdrojový obraz aktualizovaný na aktuálny stav GUI. Po zadanom maximálnom počte opakovaní je testovací scenár ukončený s chybou.

### Metóda Nahraj&Prehraj

Sikuli disponuje možnosťou nahrať sled krokov používateľa a následné vygenerovanie pseudokódu [12], ktorý môže byť transformovaný do spominaného Python kódu . Ukážkou môže byť napríklad postupnosť krokov - kliknutie na tlačítko "Štart" a následné písanie textu.




(a) Príklad nahrania Sikuli testovacieho scenára. Prevzaté z [12].




(b) Príklad vygenerovaného Sikuli pseudokódu. Prevzaté z [12].

Obr. 2.14: Príklady použitia Sikuli.

### Problematika mierky

Testovacie scenáre Sikuli v jeho súčasnej verzií 1.1.0 vytvorené na zariadení s inými parametrami, ako je zariadenie vykonávajúce scenár, môžu byť použiteľné, pretože Sikuli používa verzie vyhľadávania vzoru neschopné nájsť zhodu pri rozlišných mierkach vzorového obrazu a zdrojového obrazu.

### Výhody nástroja Sikuli

- Nástroj testuje len to, čo skutočne vidí, podobne ako reálny používateľ.
- Nástroj je nezávislý na vnútornej štruktúre používateľského rozhrania.
- Nie su vyžadované vysoké technické zručnosti používateľa Sikuli.

### Nevýhody nástroja Sikuli

- Bezpečné použitie testovacej sady je možné len na prístroji s rovnakými parametrami, ako na prístroji, kde sada vznikla.
- Nástroj testuje GUI len staticky, pre testovanie GUI presahujúceho rozmery displeju je potrebné použiť rôzne akcie na premiestnenie sústredeného obrazu.
- Nástroj slepo nachádza požadované zhody obrazov, bez akéhokoľvek poznania kontextu GUI.
- Nástroj používa výpočtovo náročné operácie pre svoju funkcionalitu čím spomaľuje vykonávanie testovacích scénarov.

## 2.3 Nástroje so znalosťou objektov

Nástroje, ktoré majú prístup k vnútornej štruktúre programu (napr. znalosť XML[1]/HTML[2] stromu GUI) majú výhodu v tom, že znalosť stavu GUI je absolútne presná, no to nemusí zodpovedať reálnemu vykresleniu GUI, ktoré používateľ vidí.

### 2.3.1 Prenositeľnosť

Nevýhodou nástrojov so znalosťou objektov je, že podpora jednotlivých štruktúr GUI nie je jednoducho prenositeľná. Je nutné osobitne pristupovať ku rôznym nástrojom použitým v GUI. Nástroje založené na prácu s HTML DOM[3] nie sú použiteľné na desktopové aplikácie používajúce iné štruktúry na uchovanie vnútorného stavu atď.

### 2.3.2 Selenium

Selenium [15] je jedným z GUI Aware nástrojov. Používa sa na automatizáciu webových používateľských rozhraní. Pre prehľad a manipuláciu s GUI používa prístup popísaný nižšie.

#### Selenium WebDriver

Pre manipuláciu so stavom web stránky je používaný Selenium WebDriver, ktorý imituje skutočnú interakciu s webovým rozhraním. Spôsob vykonávania jednotlivých akcií závisí na konkrétnych prehliadačoch a ich vstavanej podpore pre automatizáciu.

Príklady prístupu k nasledujúcemu DOM pomocou Selenium WebDriver:

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" id="submit" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
</body>
<html>
```

Kompletný Selenium skript vyzerá následovne. Inicializácia Selenium WebDriver a získanie obsahu DOM stránky, ktorú je potrebné otestovať:

```
from selenium import webdriver
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.support.ui import WebDriverWait
# available since 2.4.0
from selenium.webdriver.support import expected_conditions as EC
# available since 2.26.0

driver = webdriver.Firefox()
driver.get("http://www.google.com")
```

[1] eXtensible Markup Language
[2] HyperText Markup Language
[3] Document Object Model

Ladiaci výpis titulku stránky do terminálu:

```
print driver.title
```

Získanie elementov a zápis textu do získaných elementov:

```
username = driver.find_element_by_name("username")
username.send_keys("user")
password = driver.find_element_by_name("password")
password.send_keys("password")
```

Pre získanie viacerých elementov rovnakého mena je možné použiť funkciu `find_elements_by_name()`, ktorá v Python Selenium scripte získa zoznam všetkých vyhovujúcich elementov.

Získanie elementu tlačidla pomocou jeho ID na odoslanie formuláru a spustenie tejto akcie:

```
button = driver.find_element_by_id("submit")
button.submit()
```

Následuje overenie očakávaného stavu GUI po vykonaných akciách:

```
try:
    WebDriverWait(driver, 10).until(EC.title_contains("success!"))
    print driver.title

finally:
    driver.quit()
```

### Nedeterminizmus

Ďalším zo spôsobov prehľadávania DOM je použitie jazyku XPath[4]. Použitie je možné špecifikovaním absolútnej cesty XPath od koreňa DOM, pomocou n-tého výskytu prvku, prvku s daným ID alebo triedou. Problém môže nastať v prípade, ak cesta nie je deterministická. Druhým problémom je, ak sa obsah DOM vytvára dynamicky pomocou náhodných pomenovaní elementov HTML, preto je potrebné GUI vytvárať aj s cieľom vytvoriť ho testovateľné.

### Selenium IDE

Selenium poskytuje možnosť vygenerovať testovací scénar na základe Nahraj&Prehraj metódy pomocou rozšírení internetových prehliadačov, ktoré to podporujú. Po sledovaní akcií používateľa je vygenerovaný kód v zadanom programovacom jazyku a je možné ho upraviť podľa potrieb.

### Výhody nástroja Selenium

- Nástroj pozná kontext GUI, je schopný pristupovať k rôznym jeho vlastnostiam.
- Prístup ku prvkom GUI je relatívne rýchly.
- Veľká komunita vývojárov nástroja.
- Dynamický prístup ku GUI.

[4] XML Path Language

**Nevýhody nástroja Selenium**

- Nástroj vyžaduje vyššiu technickú zručnosť testera.
- Nástroj testuje GUI v inej forme, ako je reálne zobrazené používateľovi.

## 2.4 Metoda zaznamenaj a prehraj

Poslednou metódou sú nástroje, ktoré fungujú spolu s tzv. Zaznamenaj a Prehraj. Používateľ má možnosť naučiť testovací nástroj následnosť krokov rôznych akcií. Týmito akciami sú napríklad kliknutie tlačítka myši a stisk kláves klávesnice.

Testovací nástroj následne dokonale imituje správanie používateľa, a pri akcií kliknutí myšou simuluje kliknutie na rovnakých súradniciach, na ktorú bolo kliknuté pri vytváraní testovacieho scenára. Nevýhodou tohoto prístupu je skutočnosť, že aj malé zmeny v stave GUI oproti jeho očakávanej podobe môžu mať za následok neúspech celého testovacieho scenára.

# Kapitola 3

# Špecifikácia požiadaviek a návrh metody rozpoznávania komponent GUI

Cieľom návrhu nástroja Wirec je, aby bol implementovaný nástroj prienikom výhod existujúcich metód - reálny pohľad nástroja Sikuli na GUI, znalosť objektov GUI nástroja Selenium - a zároveň do istej miery niektoré slabé miesta eliminoval - neprenositeľnosť nástroja Selenium, neznalosť objektov GUI nástrojom Sikuli. Hlavné požiadavky je teda možné formulovať nasledovne.

## 3.1 Nefunkcionálne požiadavky

Nástroj Wirec je vytvorený použitím jazyka Python 3.6. Využíva knižnicu OpenCV na rozpoznávanie obrazu GUI pre získavanie informácií o objektoch GUI. Na detekciu textu je použitý Python module TesserOCR[1]. Nástroj Wirec je aplikácia spustiteľná z terminála akéhokoľvek počítača, ktorý disponuje závislosťami vyžadovanými nástrojom.

Závislosti nástroja Wirec sú špecifikované v súbore `requirements.txt` a v prílohe B. Pre beh nástroja Wirec je možné špecifikovať konfiguráciu použitím niektorých z argumentov terminálového rozhrania a je nutné dodať snímok GUI. Konfiguráciu je možné upraviť aj dodaním konfiguračného súboru.

Výstupom nástroja Wirec je adresár obsahujúci nasledovné:

- `out.png` – Obrázok, na ktorom sú farebne zvýraznené všetky detekované objekty.
- `in.png` – Vstupný obrázok.
- `objects.json` – Súbor formátu JSON[2] obsahujúci informácie o každom detekovanom objekte.
- `${id}.png` – Obrázok pre každý detekovaný objekt (`${id}` je zástupný reťazec reprezentujúci číselný identifikátor daného objektu).

Výstupný obrázok `out.png` používa rôzne farby na ukázanie triedy priradenej objektu. Objekt je obkreslený nasledovnými farbami, ak príslušnosť k danej triede bola vyhodnotená na viac ako 40%.

[1] Optical Character Recognition
[2] Javascript Object Notation

| Farba | Význam |
|---|---|
| Zelená | Tlačítko (ďalej len radio button). |
| Tmavo modrá | Tlačítko výberu jednej z možností (ďalej len radio button). |
| Tysová | Zaškrtávacie tlačítko (ďalej len checkbox). |
| Žltá | Textové pole na vstup (ďalej len text input field). |
| Červená | objekt klasifikovaný na menej ako 40% v každej z kategórií. |

Tabuľka 3.1: Tabuľka významu použitých farieb.

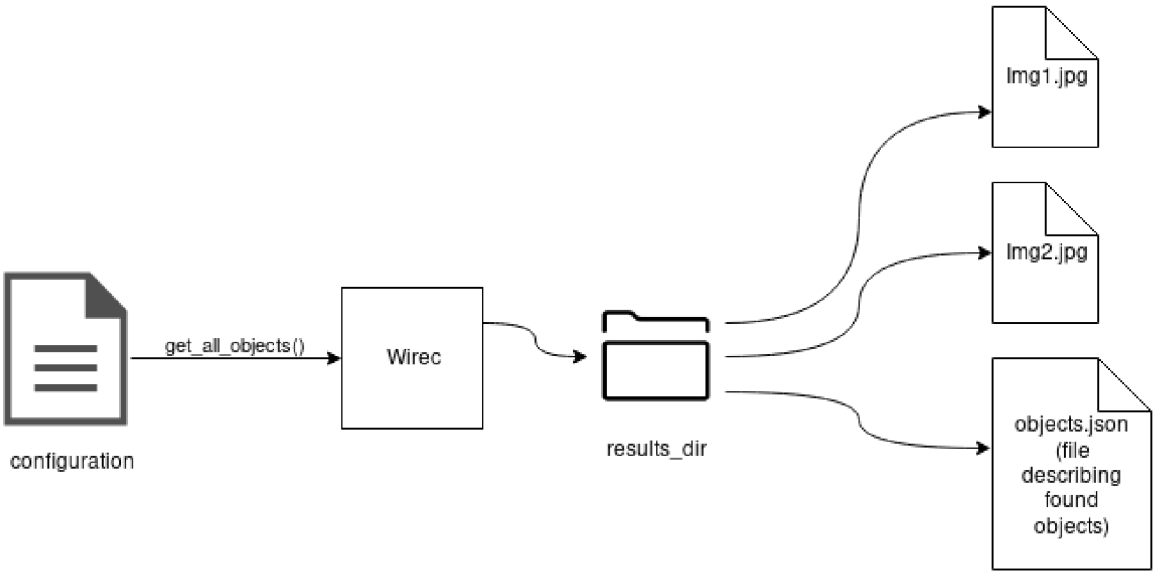
## 3.2 Špecifikácia funkcionálnych požiadaviek

| ID | Názov požiadavky | Popis požiadavky |
|---|---|---|
| 1 | Detekcia samostatných objektov GUI | Nástroj by mal detekovať všetky potrebné objekty, z ktorých neskôr na základe zadanej konfigurácie nepotrebné objekty ignoruje. |
| 2 | Definovanie hierarchických vzťahov | Nástroj priradí objektom informácie, kam v hierarchickom strome GUI patria. |
| 3 | Získavanie vlastností objektov | Nástroj získa a priradí množstvo informácií o daných objektoch. |
| 4 | Klasifikácia objektov | Nástroj na základe získaných informácií o objektoch a tiež na základe zadanej konfigurácie vyhodnotí pravdepodobnosť príslušnosti objektu k určitej triede objektov. Klasifikácia bude úplne konfigurovateľná. |
| 5 | Konfigurovateľnosť | Nástroj bude plne konfigurovateľný, aby bolo jeho funkcionalitu možné zásadne ovplyvňovať aj bez úprav zdrojového kódu. Nástroj bude distribuovaný spolu so vzorovým konfiguračným súborom. Nástroj bude schopný behu aj bez dodanej konfigurácie použitím základných vstavaných nastavení. |
| 6 | Rozšíriteľnosť | Detekovanie objektov, získavanie informácií a klasifikácia objektov budú implementované pomocou zásuvných modulov schopných bezproblémovo obohatiť informácie získané o detekovaných objektoch. Existujúce zásuvné moduly budú absolútne nahraditeľné. S týmto súvisí aj spôsob implementácie umožňujúci jednoducho pristupovať k programovému rozhraniu aplikácie. |

| | | |
|---|---|---|
| 7 | Výkonnosť | Nástroj bude možné konfigurovať tak, aby boli niektoré výpočtovo náročné operácie vynechané pre zrýchlenie behu. |
| 8 | Generovanie klasifikovaných objektov | Každý klasifikovaný objekt bude uložený ako samostatný obrázok v adresári s výsledkami. |
| 9 | Generovanie JSON výstupu | Podrobné informácie získané o jednotlivých objektoch budú vygenerované vo forme Javascript Object Notation súboru, pre jednoduchý prístup k informáciám iným nástrojom používajúcim výstup nástroja Wirec. |

Tabuľka 3.2: Tabuľka funkcionálnych požiadavkov.

Po stanovení požiadavok vyzerá použitie nástroja nasedovne. Nástroju je dodaný nepovinný konfiguračný súbor a jeho výstupom je adresár s JSON súborom popisujúcim klasifikované objekty a samotnými objektami vo forme PNG[3] obrázkov.




Obr. 3.1: Priebeh použitia nástroja Wirec.

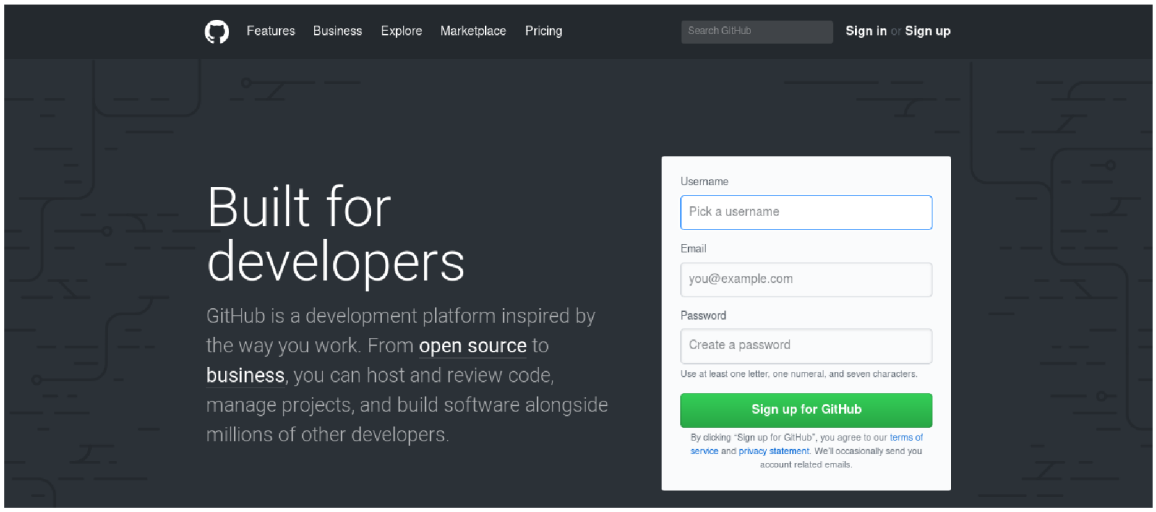
## 3.3 Návrh metódy rozpoznávania komponent

Nástroj je založený na predpoklade, že väčšina GUI je implementovaných konvenčne a teda jednotlivé komponenty, z ktorých sa používateľské GUI skladá, sú jednoznačne oddelené od okolitého prostredia farebne kontrastne. Tento predpoklad poskytuje priestor k použitiu algoritmov založených na detekcií hrán, detekcií kontúr objektov a analýze obsahu a okolia detekovaných objektov.

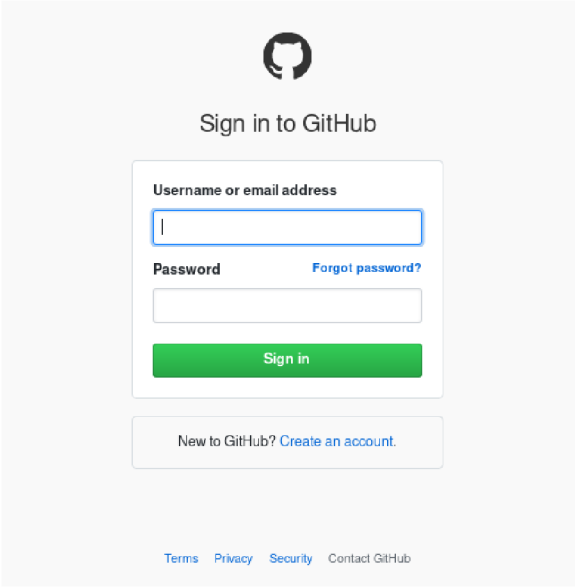
[3]Portable Network Graphics

### 3.3.1 Typické vlastnosti komponent GUI

Príklady konvenčne štýlovaných GUI, na ktoré je vhodné použiť nástroj Wirec.




Obr. 3.2: Príklad č. 1 štandardného GUI. Prevzaté z [2].




Obr. 3.3: Príklad č. 2 štandardného GUI. Prevzaté z [3].

Na obrázkoch je vidieť, že jednotlivé podstatné komponenty tohoto GUI sú jednoducho odlíšiteľné od svojho pozadia. Nástroj bude teda analyzovať vlastnosti ako tvar, farba pozadia, hierarchické vzťahy, veľkosť oproti rodičovskému objektu, tj. objektu, ktorý je vizuálne je mu priamo nadriadený a tiež umiestnenie oproti rodičovskému objektu, tj. vertikálne a horizontálne zarovnanie.

Tieto vlastnosti sami o sebe veľkú hodnotu nemajú, no s informáciami o typických vlastnostiach istého GUI je možné vytvoriť predpoklad, že prvky istej triedy majú rovnaké vlastnosti. To znamená, že je možné stanoviť kritéria pre klasifikáciu objektu napríklad

na základe toho, že objekty danej triedy obsahujú text, ktorý je vertikálne aj horizontálne zarovnaný na stred a daný objekt ma určitý tvar.

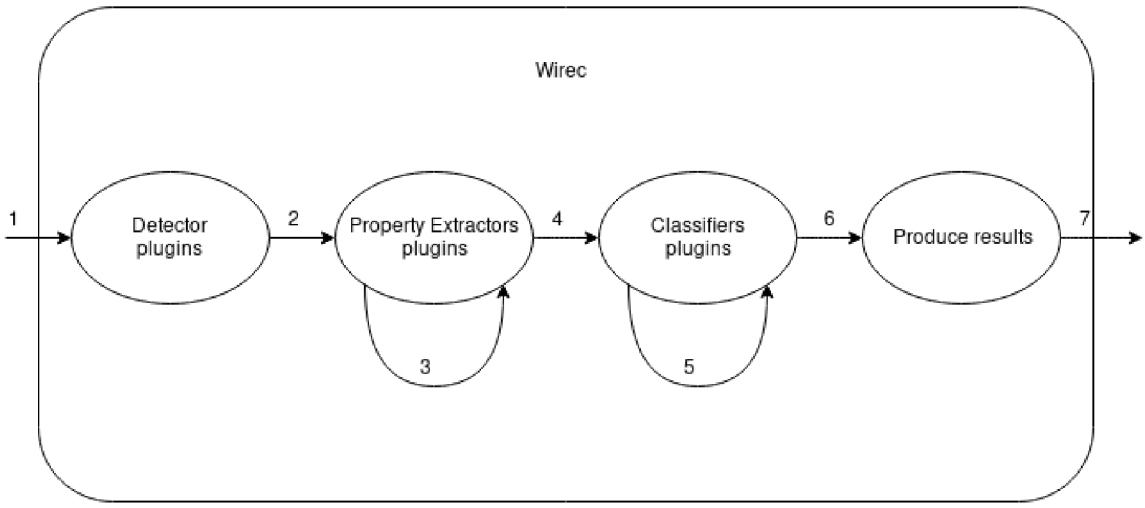
Tento predpoklad ale nie je možné úplne zovšeobecniť, preto bola stanovená požiadavka č. 5 a č. 6, ktoré umožnia jednoducho takéto kritéria špecifikovať pred spustením programu. Dodané zásuvné moduly sa budú zameriavať na podporu hodnotného rozpoznávania používaných webových aplikácií ako je napríklad Github, či množina webových stránok používajúcich webový framework Bootstrap.

### 3.3.2 Zásuvné moduly

Požiadavka rozšíriteľnosti je dôležitá z dôvodu odlišných vlastností rôznych GUI. Túto požiadavku nástroj Wirec rieši cez zásuvné moduly, ktoré byť rôzne pridávané prípadne rušené.

Súčasný koncept je taký, že v beh programu je rozdelený na viacero fáz. V každej fáze bude spúšťaná daná trieda modulov, kde každý modul danej triedy obohacuje dosiaľ získané výsledky danej triedy modulov. Tým sa dosiahne to, že nebude nutné vždy upravovať určitú časť kódu, ale bude možné len pridať nový modul, ktorý obohatí beh programu nezávisle na ostatných. Taktiež bude možné kedykoľvek určitý z modulov odstaviť, ak nebude vhodný na použitie z rôznych dôvodov.

Priebeh nástroja Wirec znázorneného na obrázku 3.4 bude vyzerať následovne.




Obr. 3.4: Diagram toku dát - Wirec zásuvné moduly.

| **Identifikátor hrany** | **Význam** |
|---|---|
| 1 | Vstupný obrázok pre nástroj Wirec. |
| 2 | Detekované objekty |
| 3 | Detekované objekty obohatené o každý jeden nezávislý beh všetkých zásuvných modulov typu Property Extractor. |
| 4 | Objekty obohatené o informácie. |

| | |
|---|---|
| 5 | Objekty obohatené o informácie o príslušnosti k triedam objektov. Tieto informácie sú získavané nezávislými behmi zásuvných modulov typu Klasifikátor. |
| 6 | Klasifikované objekty. |
| 7 | Výstup vo forme zoznamu klasifikovaných objektov ľahko prevediteľný na JSON formát. |

Tabuľka 3.3: Tabuľka významu diagramu toku dat 3.4.

Na obrázku hrany idúce z jedneho stavu do toho istého znázorňujú iterácie získavania nových dat nezávisle na dátach získaných zásuvným modulom tej istej triedy, pre splňenie požiadavky 6 z tabuľky 3.2.

# Kapitola 4

# Implementačné detaily rozpoznávania GUI

Priebeh nástroja Wirec je rozdelený do 4. základných fáz. Prvou je konfigurácia, aj na základe konfigurácie sú v ďalšej fáze detekované objekty, ktorých vlastnosti sú neskôr analyzované a informácie o nich sú nakoniec použité na samotnú klasifikáciu. Niektoré zo spomenutých fáz môžu byť rozšíriteľné pomocou Python zásuvných modulov.

## 4.1 Konfigurácia

Nástroj Wirec je konfigurovateľný pomocou argumentov programu ako aj dodaním konfiguračného súboru. Týmto je splnená požiadavka 5 z tabuľky 3.2. V tabuľkách sú použité prednastavené hodnoty v tvare *prednastavená_hodnota (jednotka/interval platných hodnôt)*. Jednotka pixelu je skrátene zapísana ako px, jednotka kladného celého čísla ako uint a interval v tvare <0,1> znamená desatinné číslo v danom intervale. Jednotka "IK" je vysvetlená neskôr v kapitole 4.4.1. Posledným spôsobom platných hodnôt je vymenovanie všetkých možností oddelených lomítkom.

### 4.1.1 Konfiguračný súbor

Konfigurácia sa skladá z viacerých logicky zoskupených nastavení v konfiguračnom súbore. Každá skupina nastavení je v krátkosti vysvetlená vo vlastnej podkapitole.

#### Parametre detekcie objektov

Nasledujúca tabuľka obsahuje zoznam parametrov ovplyvňujúcich množstvo a charakter detekovaných objektov ako aj druh objektov, ktoré budú odstránené vo fáze detekcie objektov.

| **Názov a hodnota vlastnosti** | **Význam** |
|---|---|
| *button_min_horizontal_edge_length =0.6 (<0,1>)* | Minimálna dĺžka spojitej horizontálnej hrany potrebná na to, aby objekt mohol byť považovateľný za objekt triedy button. |

| | |
|---|---|
| *button_min_vertical_edge_length =0.4 (<0,1>)* | Minimálna dĺžka spojitej vertikálnej hrany potrebná na to, aby objekt mohol byť považovateľný za objekt triedy button. |
| *dilate_x=2 (uint)* | Šírka jadra použitého na dilatáciu objektov. Použitie: Rozmazanie textových objektov, keďže ich detaily jednotlivých znakov môžu byť nepotrebné. |
| *dilate_y=2 (uint)* | Výška jadra použitého na dilatáciu objektov. Použitie: Rozmazanie textových objektov, keďže ich detaily jednotlivých znakov môžu byť nepotrebné. |
| *em=12 (px)* | Parameter udávajúci štandardnú veľkosť písma. Násobky tohoto parametru rôznymi koeficientmi vnútri nástroja Wirec ovplyvňujú jeho správanie. Príkladom je zahodenie objektov, ktorých obsah je menší ako 3-násobok hodnoty em. |
| *min_dist_h_relation=2 (px)* | Minimálna možná vzdialenosť nasledujúceho horizontálneho objektu na to, aby bol považovaný so súčasným objektom v relácií. Použitie: definovanie vzťahu medzi radio button a textom nasledujúcim horizontálne za ním. |
| *max_dist_h_relation=50 (px)* | Maximálna možná vzdialenosť nasledujúceho horizontálneho objektu na to, aby bol považovaný so súčasným objektom v relácií. Použitie: definovanie vzťahu medzi radio button a textom nasledujúcim horizontálne za ním. |
| *max_dist_v_relation=15 (px)* | Maximálna možná vzdialenosť nasledujúceho vertikálneho objektu na to, aby bol považovaný so súčasným objektom v relácií. |
| *max_obj_size=-1 (px)* | Maximálny obsah objektu potrebný na to, aby bol objekt považovaný za zmysluplný prvok GUI. |
| *merge_based_on_rel_size= 0.8 (<0,1>)* | Parameter udáva relatívny obsah objektu oproti jeho rodičovskému prvku potrebný na to, aby bol objekt zjednotený s rodičovským prvkom. Použitie: Zjednotenie objektov s tenkými hranamy - eliminácia dvojitého detekovania rovnakého objektu typu button, kde je detekovaná aj vnútorná aj vonkajšia kontúra. |
| *min_obj_size=20 (px)* | Minimálny obsah objektu potrebný na to, aby bol objekt považovaný za zmysluplný prvok GUI. |

| | |
|---|---|
| *min_obj_height=-1 (px)* | Minimálna výška objektu potrebná na to, aby bol objekt považovaný za zmysluplný prvok GUI. |
| *out_of_contour_tolerance= 0.05 (<0,1>)* | Tolerancia pre vzdialenosť bodu od objektu, pri zisťovaní tvaru objektu. Použitie: Bounding Box obdlžníkových objektov musí brať do úvahy radius (zaoblenie hrán). |
| *threshold=10 (jednotka jasu <0,255>)* | Prah prijateľnosti detekovaných hrán použitím Laplacian detektoru. |

Tabuľka 4.1: Tabuľka parametrov detekcie objektov.

### Parametre klasifikácie

Tieto parametre stanovujú pravidlá pre klasifikáciu.

| **Názov a hodnota vlastnosti** | **Význam** |
|---|---|
| *button_centered=0.4 (<0,1>)* | Maximálna možná hodnota priraditeľná objektu v prípade, že nastane ideálna zhoda podľa zadaných parametrov pri zarovnaní potomkovských prvkov. |
| *button_centered_int=-0.3x0.3xm (IK)* | Interval hodnôt a oblasť intervalu najlepšej zhody. Parameter ohraničujúci zarovnanie potomkov. |
| *button_size=0.3 (<0,1>)* | Maximálna možná hodnota priraditeľná objektu v prípade, že potomkovia tvoria dané % obsahu daného objektu. |
| *button_size_int= 0.0x0.6xc (IK)* | Interval hodnôt a oblasť intervalu najlepšej zhody. Parameter stanovujúci prípustné % obsahu tvorené potomkami. |
| *button_text=0.3 (<0,1>)* | Maximálna možná hodnota priraditeľná objektu v prípade, že nastane ideálna zhoda podľa zadaných parametrov pri existencií textu v potomkoch. |
| *button_text_int=1x30xc (IK)* | Interval hodnôt a oblasť intervalu najlepšej zhody. Parameter stanovujúci prípustný počet znakov v texte potomkov. |
| *textarea_centered_left= 0.35 (<0,1>)* | Maximálna možná hodnota triedy textarea priraditeľná objektu v prípade, že potomkovia sú horizontálne zarovnaný podľa zadaných parametrov. |

| | |
|---|---|
| *textarea_centered_left_int= -1x0xl (IK)* | Interval hodnôt a oblasť intervalu najlepšej zhody. Parameter stanovujúci prípustné horizontálne zarovnanie potomkov. |
| *textarea_centered_top= 0.35 (<0,1>)* | Maximálna možná hodnota triedy textarea priraditeľná objektu v prípade, že potomkovia sú vertikálne zarovnaný podľa zadaných parametrov. |
| *textarea_centered_top_int= -1x0.2xm (IK)* | Interval hodnôt a oblasť intervalu najlepšej zhody. Parameter stanovujúci prípustné vertikálne zarovnanie potomkov. |
| *textarea_size=0.3 (<0,1>)* | Maximálna možná hodnota triedy textarea priraditeľná objektu v prípade, že potomkovia tvoria % obsahu daného objektu podľa zadaných parametrov. |
| *textarea_size_int= 0.0x0.2xm (IK)* | Interval hodnôt a oblasť intervalu najlepšej zhody. Parameter stanovujúci prípustné % obsahu tvoreného potomkami. |

Tabuľka 4.2: Tabuľka parametrov klasifikácie objektov.

### Ladiace nastavenia

Nasledujúce parametre sú primárne využívané pri ladení nástroja Wirec ako aj pri zisťovaní príčin nesprávnej detekcie a klasifikácie objektov.

| **Názov a hodnota vlastnosti** | **Význam** |
|---|---|
| *contours_optimized=True (bool)* | Použitie optimalizovaného počtu kontúr použitých na definovanie objektu. True - množstvo kontúr optimalizované, False - zachovanie skutočného počtu pixelov obsiahnutých v kontúre objektov. |
| *display_contours=False (bool)* | Možnosť zobraziť klasifikovane vykreslený obrázok out.png hneď po ukončení klasifikácie nástrojom Wirec. Použitie: Zákaz zobrazenia v CI a prostrediach neobsahujúcich display socket. True - zobrazenie obrázku out.png, False - Zákaz zobrazenia. |
| *display_contours_bounding_box= True (bool)* | Obkreslenie objektov všetkými bodmi nachádzajucími sa v kontúre. True - Nakreslený len Minimal Bounding Box na základe kontúr, False - Vykreslenie všetkých bodov kontúry. |

| | |
|---|---|
| *display_object_id=False (bool)* | Výpis ID objektu na výstupný obrázok out.png. True - Výpis ID objektu, False - zákaz výpisu. |
| *show_threshed_image=False (bool)* | Zobrazenie vstupného obrazu po detekovaní hrán. True - zobrazenie obrazu používateľovi, False - zákaz zobrazenia. |

Tabuľka 4.3: Tabuľka ladiacich parametrov.

### Nastavenia riadiace výber algoritmov

Týmito parametrami je možné ovplyvniť výber algoritmov, ktoré sú použité v rámci ktorejkoľvek fáze.

| **Názov a hodnota vlastnosti** | **Význam** |
|---|---|
| *edges=8U (8U/8S/16S/32S/32F/64F)* | Druh reprezentácie vstupného obrazu po detekovaní hrán. Dostupné počty kanálov pre farby - 1,2,3 a 4. Detaily v knihe [10] v kapitole "IplImage Data Structure" |
| *laplace=True (bool)* | Metóda použitá na detekciu hrán objektov. Hodnoty: True - Laplace, False - jednoduchý thresholding. |
| *speed=1 (int)* | Argument stanovujúci minimálnu potrebnú rýchlosť zásuvného modulu potrebnú na to, aby bol spustený. Použitie: Zrýchlenie behu nástroja Wirec vynechaním modulov náročných na výpočet. Týmto je splnená požiadavka 7 z tabuľky 3.2. |
| *use_tesseract=True (bool)* | Možnosť zakázať použitie tesserOCR Python modulu ako aj zákaz detekovania textu. Použitie: Zrýchlenie behu nástroja Wirec. Hodnoty: True - použitie tesserOCR a detekcia textu, False - Zákaz detekcie textu |

Tabuľka 4.4: Tabuľka parametrov ovplyvňujúcich výber algoritmov.

### Nastavenia ciest súborového systému

Poslednou kategóriou sú parametre nastavujúce rôzne cesty k súborom ako je napríklad cesta ku adresáru obsahujúcemu všetok výstup nástroja Wirec a ďalšie.

| **Názov a hodnota vlastnosti** | **Význam** |
|---|---|
| *old_img=None (cesta v súborovom systéme)* | Cesta ku vstupnému snímku obsahujúcemu stav GUI v predchádzajúcom behu. Použitie: Tento vstup je potrebný pre zásuvné moduly triedy komparátor. |
| *old_state=None (cesta v súborovom systéme)* | Cesta k JSON súboru obsahujúcemu stav GUI v predchádzajúcom behu. Použitie: Tento vstup je potrebný pre zásuvné moduly triedy komparátor. |
| results_dir=*$(pwd) (cesta v súborovom systeme)* | Cesta k výstupnému adresáru obsahujúcemu všetok výstup. |

Tabuľka 4.5: Tabuľka parametrov ciest v súborovom systéme.

Každý z parametrov má prednastavenú hodnotu, tú je možné prepísať konfiguračným súborom s väčšou prioritou a najväčšiu prioritu majú argumenty získané z rozhrania príkazového riadku. Tých je ale menšia množina:

| **Názov a hodnota vlastnosti** | **Význam** |
|---|---|
| *config_path=None* | Cesta ku konfiguračnému súboru. |
| *display_contours* | Viď tabuľka 4.3 |
| *filepath=None* | Cesta ku vstupnému obrázku. |
| *old_img* | Viď tabuľka 4.5 |
| *old_state* | Viď tabuľka 4.5 |
| *results_dir* | Viď tabuľka 4.5 |
| *show_threshed_image* | Viď tabuľka 4.3 |
| *speed* | Viď tabuľka 4.4 |
| *threshold* | Viď tabuľka 4.1 |

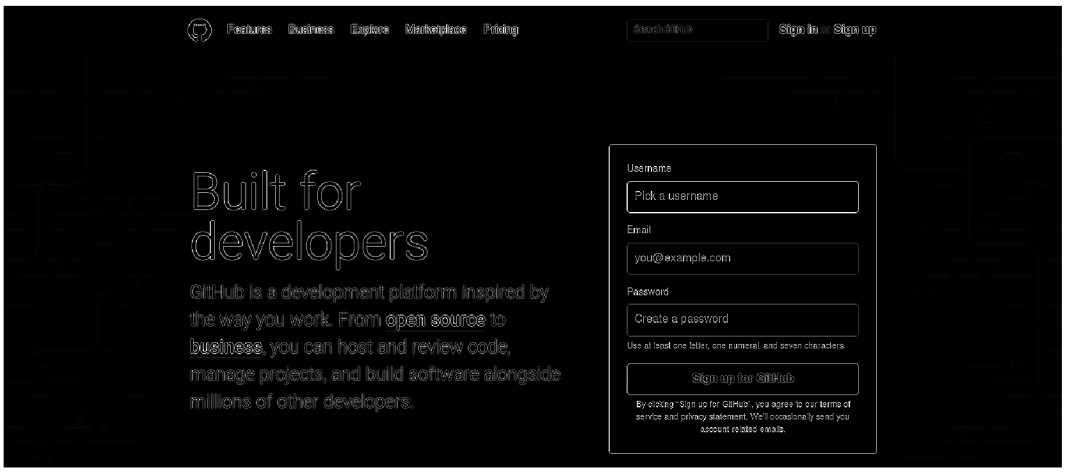
Tabuľka 4.6: Tabuľka parametrov terminálového rozhrania.

## 4.2 Detekcia

Druhou fázou nástroja Wirec je detekcia objektov. Cieľom tejto fázy je detekovať čo najväčšie množstvo objektov, ktoré majú prínos pre neskoršie fázy, a zároveň sa vyhnúť detekcií objektov, ktoré majú len negatívny vplyv na rozpoznávanie. Táto problematika je rozoberaná v podkapitole 4.2.2.

Táto fáza v súčasnosti nefunguje v iteráciách viacerého počtu zásuvných modulov typu detektor, pretože pri viacerých rôznych detektoroch by došlo ku duplicitne detekovaným objektom. Rôzne detektory je odporúčané používať vo viacerých nezávislých behoch nástroja Wirec.

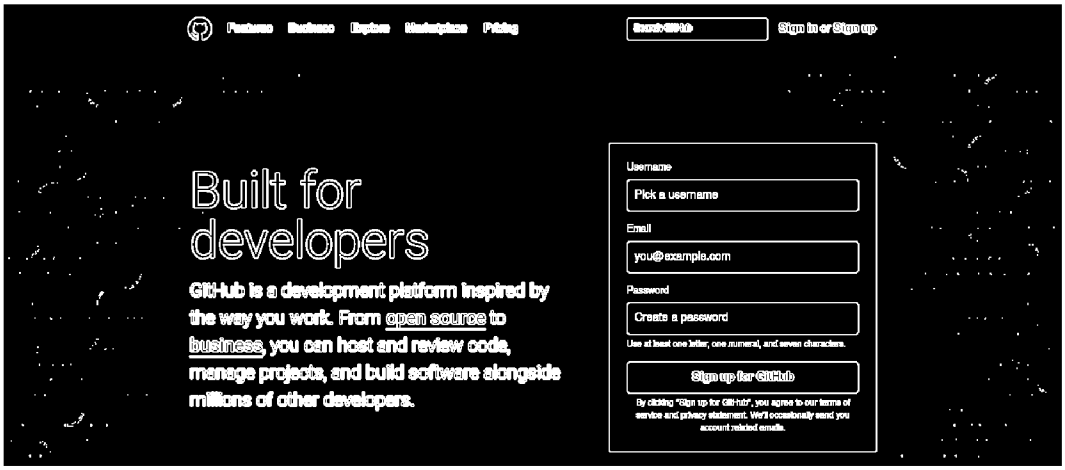
Na ukázanie priebehu fáze detekcie je použitý obrázok 3.2. Fáza detekcie je uskutočnená pomocou prevedenia vstupného obrazu na bezfarebný. Následne sú v bezfarebnom snímku detekované hrany pomocou Laplace operátora:




Obr. 4.1: Detekované hrany na vstupnom snímku.

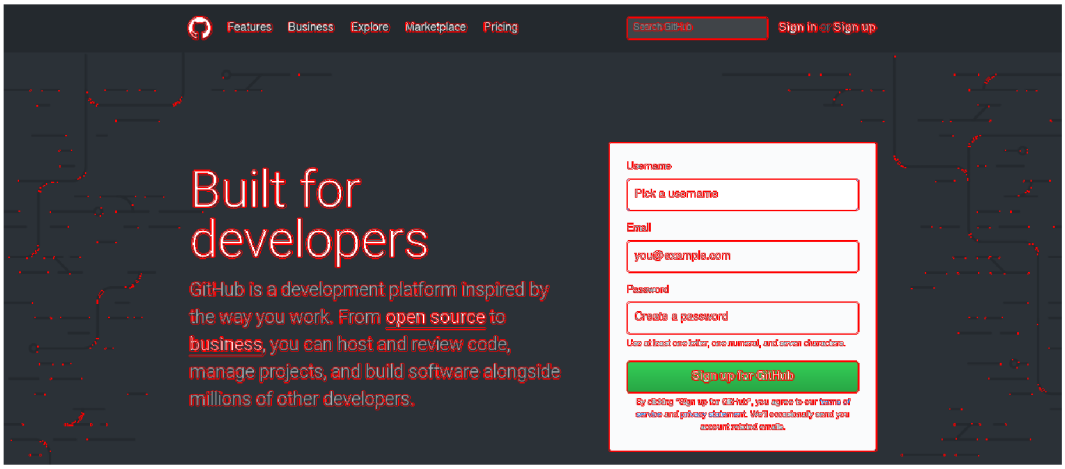
Keďže pri detekcií kontúr je každý biely objekt s uzavretými obrysmi považovaný za objekt, je užitočné eliminovať detaily pri objektoch ako napríklad text, keďže detaily pri takýchto objektoch dôležité v tejto fáze nie sú. Jedným zo spôsobov eliminácie detailov je dilatácia, pri ktorej je každý objekt jemne rozmazaný podľa použitého jadra na dilatáciu.

Ďalším dôvodom dilatácie sú prípady, keď istá časť uzavretého objektu má časť hrany slabo detekovanú a pri získavaní kontúr by bola zle detekovaná. Po tom ako bola aplikovaná dilatácia, je ešte potrebné odstrániť objekty, pri ktorých bola hrana detekovaná príliš slabo. Hrany s jasnosťou nižšiou ako je hodnota prahu nastaveného konfiguráciou sú odstránené.




Obr. 4.2: Snímka bez objektov s nízkym prahom.

V tejto chvíli je úprava vstupného snímku na konci a je možné pristúpiť k detekcií kontúr jednotlivých objektov. Kontúry získané zo snímku prevedeného prahovaním vyzerá po vykreslení všetkých bodov každej kontúry nasledovne.




Obr. 4.3: Snímok s vykreslenými kontúrami objektov.

### 4.2.1 Definovanie vzťahov medzi detekovanými objektami

Potom čo sú kontúry objektov získané, je potrebné definovať vzťahy medzi jednotlivými objektami, pre neskoršie použitie. Týmto je splnená požiadavka 1 a 2 z tabuľky 3.2.

Všetky detekované objekty sú iterované a pre každý objekt je hľadaný najbližší rodičovský potomok. Pre prípad, že GUI neobsahuje objekt, ktorý by bol najvyšším rodičovským prvkom v hierarchií, je vytvorený objekt, ktorý má rozmery vstupného snímku. Takto je vytvorený celý strom vzťahov pomocou algoritmu 1.

**Data:** Detekované objekty
**Result:** Objekty obsahujúce hierarchické vzťahy
**foreach** *detekované objekty as O* **do**
  **foreach** *detekované objekty as F* **do**
    **if** *F je v hierarchií nadradený a je bližšie v hierarchií ako O.parent* **then**
      O.parent = F;
  **end**
**end**
fake_parent = umelo vytvorený objekt zhodný so vstupným obrázkom
**foreach** *detekovaný objekt as O* **do**
  **if** *if O.parent == null* **then** O.parent = fake_parent;
**end**

**Algoritmus 1:** Pseudo-algoritmus zisťovania vzťahov medzi detekovanými objektami.

Pre zistenie či objekt patrí pod aktuálne iterovaný rodičovský objekt je využitá funkcia `belongs_under()`, ktorá kontroluje, či Minimal Bounding Box kontúr daného objektu patrí celý do Minimal Bounding Box kontúr rodičovského objektu. Na záver je zistené či sa ťažisko kontúr objektu nachádza v oblasti kontúr rodičovského objektu. Ďalšou z funkcionalít tejto funkcie je aj zistenie nasledujúceho horizontálneho a aj vertikálneho objektu.

Táto funkcia na základe zadanej konfigurácie nájde spomedzi svojich súrodeneckých objektov ten, ktorý je najbližšie a ak spĺňa konfigurované parametre, je aj označený ako horizontálne vo vzťahu s daným prvkom.

**Data:** Objekt, ktorého nasledovník má byť nájdený
**Result:** Objekt s informáciou o vzdialenosti horizontálneho následníka
**foreach** *súrodenecké objekty* **do**
| Nájdi súrodenecky najbližší horizontálny objekt vyhovujúci aj vertikálne
**end**
**if** *Najbližší objekt vyhovuje kritériám (pomer výšok a vzdialenosť objektov)* **then**
| Ulož informáciu o nasledujúcom prvku
**else**
| Nastav informáciu o nasledujúcom prvku na nedefinovanú
**end**

**Algoritmus 2:** Zistenie horizontálne nasledujúceho objektu.

### 4.2.2 Odstraňovanie prebytočných objektov

Dodaný zásuvný modul Laplace po získaní kontúr všetkých objektov najprv tieto objekty prefiltruje na základe konfigurovanej minimálnej a maximálnej širky, výšky či obsahu objektu, respektíve na základe pomeru obsahu ku konfigurovanej štandardnej výške textu. To však odstráni len malé množstvo objektov a preto sú potrebné nasledujúce kroky.

#### Spájanie objektov s tvarom podobným textu

Pri aplikáciách kde sa nachádza množstvo textov, je potrebné detekované textové objekty upraviť pred ďalším spracovaním. Toto je uskutočňované spájaním za sebou horizontálne idúcich objektov, ktorých tvar pripomína text. Kritériami pre spojenie dvoch za sebou idúcich textových objektov sú rovnaké pre oba spájané objekt. Spojenie nie je možné v nasledujúcich prípadoch:

- Objekt je obĺžnikového tvaru a obsahuje 1 a viac potomkov.
- Objekt je kruhového tvaru.
- Potomkovia objektu obsahujú kruhový alebo obdĺžníkový tvar a potomkovia sú zarovnaní na stred.
- Rozdiel vo výške objektov je väčší ako dovoľujú konfigurované parametre.
- Vzdialenosť medzi objektami je väčšia ako dovoľujú konfigurované parametre.

Prístup je popísaný algoritmom 3.

Táto funkcia bola experimentálne používaná aj na spájanie vertikálne nasledujúcich objektov, napríklad na účel spájania riadkov textov, do textových blokov, v prípade, že spolu súviseli, ale zatiaľ neboli stanovené dostačujúce kritéria na to, aby toto bolo vykonávané správne.

#### Spájanie objektov s rodičovským objektom

Po tom ako sa skončí horizontálne spájanie objektov je ešte možné odstrániť objekty detekované na základe vnútorných kontúr znakov textu. Príklad detekovanej vnútornej kontúry je na obrázku 4.4.

**Data:** Objekt, ktorého nasledovník má byť spojený
**Result:** Pravdivostná hodnota spojiteľnosti objektov
Zisti následníka volaním funkcie next_horizontally()
**foreach** *potomkovské objekty* **do**
| Vytvor zoznam vycentrovaných objektov, ktoré sú kružnicou alebo obdlžníkom
**end**
**if** *Aspoň jeden (kruhový/obdĺžnikový)objekt tvorí 80 a viac percent obsahu rodičovského objektu* **then**
| return False
**else**
| **if** *Existuje vzťah medzi horizontálnym následníkom* **then**
| | return True
| **else**
| | return False
| **end**
**end**

**Algoritmus 3:** Zistenie spojiteľnosti objektu s jeho horizontálnym následníkom.


Obr. 4.4: Detekované vnútorné kontúry v texte.

Takýchto vnútorných kontúr môže byť detekované veľké množstvo pri početnom výskyte textu na snímke obrazovky. Takéto detekované objekty nemajú žiadny prínos pre ďalšie fázy nástroja Wirec, preto je nutné ich eliminovať. To ale nie je možné úplne dokonale.

Použitý postup v dodanom detektore, je založený na predpoklade, že kontúry textu nemajú tvar obdlžníku ani kruhu. Tento predpoklad je dôležitý z dôvodu, že túto vlastnosť majú spravidla rôzne rámce ohraničujúce väčšie oblasti GUI, a tiež rôzne komponenty ako napríklad tlačítka, radio button, alebo checkbox. Zo spomenutých preto vyplýva priestor na algoritmus 4.

**Data:** Pole objektov
**Result:** Pole objektov bez prebytočných objektov
**foreach** *objekty as O* **do**
| **if** *!O.circle and !O.rectangular and !O.parent.circle and !O.parent.rectangular* **then** objekty.remove(O) ;
**end**

**Algoritmus 4:** Odstránenie prebytočných objektov nachádzajúcich sa v objektoch textu.

Z algoritmu je zrejmé, že celé odstraňovanie prebytočných objektov je závislé na detekcií tvaru.

### Detekcia obdĺžnikovitého tvaru

Na zistenie vlastnosti obdĺžnikovitého tvaru je použitý algoritmus založený na predpoklade, že obdĺžnik má všetky body svojich hrán blízko minimálneho opisujúceho obdĺžnika s určitou toleranciou.

**Data:** Objekt, ktorého tvar je zisťovaný
**Result:** Pravdivostná hodnota
Získaj súradnice rohov obdĺžníka opisujúceho kontúry objektu
**if** *Ak smer niektorej z horizontálnych hrán sa vertikálne mení* **then** return False ;
**if** *Ak smer niektorej z vertikálnych hrán sa horizontálne mení* **then** return False ;
return True

**Algoritmus 5:** Zistenie hranatosti objektu.

Na zistenie vertikálneho resp. horizontálneho odklonu hrany sa používa funkcia `is_continuous_edge()`, ktorá testuje hrany objektu na vzdialenosť od kontúr s konfigurovateľnou vzdialenosťou. Využíva na to funkciu OpenCV `pointPolygonTest()`. Táto funkcia získa len približný tvar objektu, pre zistenie dokonalej hranatosti objektu je použitá funkcia `is_rectangle()`, ktorá testuje úplnú zhodu bodov opisujúceho obdĺžniku s hranami objektu, v ktorej aj odchylka o 1 pixel môže spôsobiť nesprávnu detekciu.

### Detekcia kruhového tvaru

Pre zistenie kruhového tvaru objektu je použitá funkcia `is_circle()`, ktorá využíva algoritmus popisaný v kapitole 2.1.3. Je potrebné stanoviť minimálny a maximálny možný polomer detekovateľného kruhu. Predpoklad objekt považovaného za kruh musí mať podiel výšky a šírky blízky 1 a stred kružnice v strede obdĺžniku opisujúceho tento objekt. V opačnom prípade by mohlo dôjsť ku detekovaniu kruhu napríklad v čísle "8", preto je potrebné aby kruh mal stred v strede detekovaného objektu.

**Data:** Objekt O, ktorého tvar je zisťovaný
**Result:** Pravdivostná hodnota
Konvertuj obrázok vstupného objektu na bezfarebný.
Nastav minimálny polomer na tretinu šírky objektu a maximálny na polovicu objektu.
**if** *! O.width* $\doteq$ *O.height* **then** return False ;
Detekuj kruhy na obrázku objektu.
**foreach** *Detekované kruhy* **do**
  **if** *Ak detekovaný kruh nemá stred kružnice blízko stredu objektu* **then** return False ;
**end**
return True

**Algoritmus 6:** Zistenie kruhovosti objektu.

Tento algoritmus môže detekovať aj kružnice, ktoré nezapadajú do kontextu chápania kružnice nástroja Wirec. Dôvodom detekovania stredu kružnice blízko stredu objektu je snaha vyhnúť sa detekcií kružníc, ktoré sú v skutočnosti "False Positives", keďže parametre detekcie kružnice sú nastavené tak, aby detekovali kružnice s menšou toleranciou.

#### False positives nachádzajúce sa v textových objektoch

Kritéria spomenuté v predchádzajúcich algoritmoch sú stanovené z dôvodu podobnosti niektorých znakov abecedy podobným tvarom objektov, ktoré nie je vhodné spájať. Dôvodom je, že väčšina objektov, ktoré majú kruhový alebo obdĺžnikový tvar a sú samostatné, reprezentujú tlačítka, veľké samostatne bloky GUI, radio button, checkbox button, či iné.

Tomuto sa ale nie je možné úplne vyhnúť, pretože určité typy písem kombinované s určitými veľkosťami môžu spôsobiť, že sú v nich detekované objekty pripomínajúce spomínané tvary. Napríklad písmená I, l, O a C sú zvlášť náchylné na nesprávne vyhodnotenie a preto vedú k nesprávnemu horizontálnemu spájaniu textových objektov.

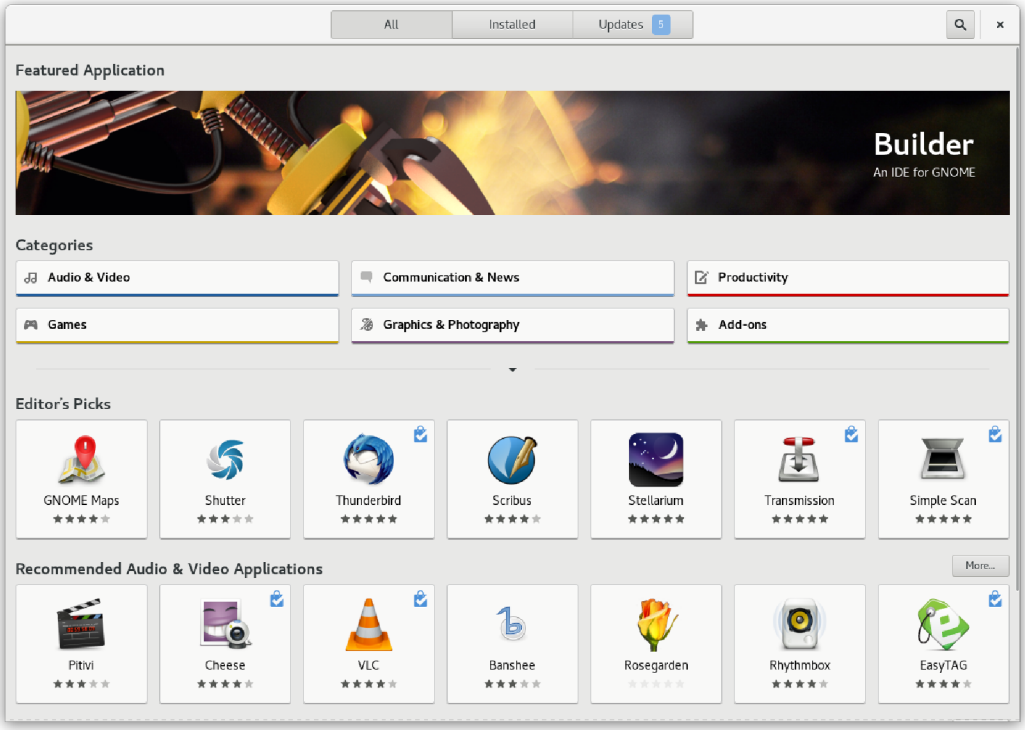
### 4.2.3 Obmedzenia detekcie kontúr

Niektoré GUI, ktoré pozostávajú z veľmi kontrastných spolu súvisiacich oblastí sú problémom pri detekcií objektov. To z toho dôvodu, že je detekované množstvo objektov, ktoré nemajú absolútne žiadny prínos pre ďalšie fázy a nie je možné ich jednoducho odstrániť na základe žiadnych dosiaľ zistených heuristík.

#### Nemožnosť odstránenia objektov

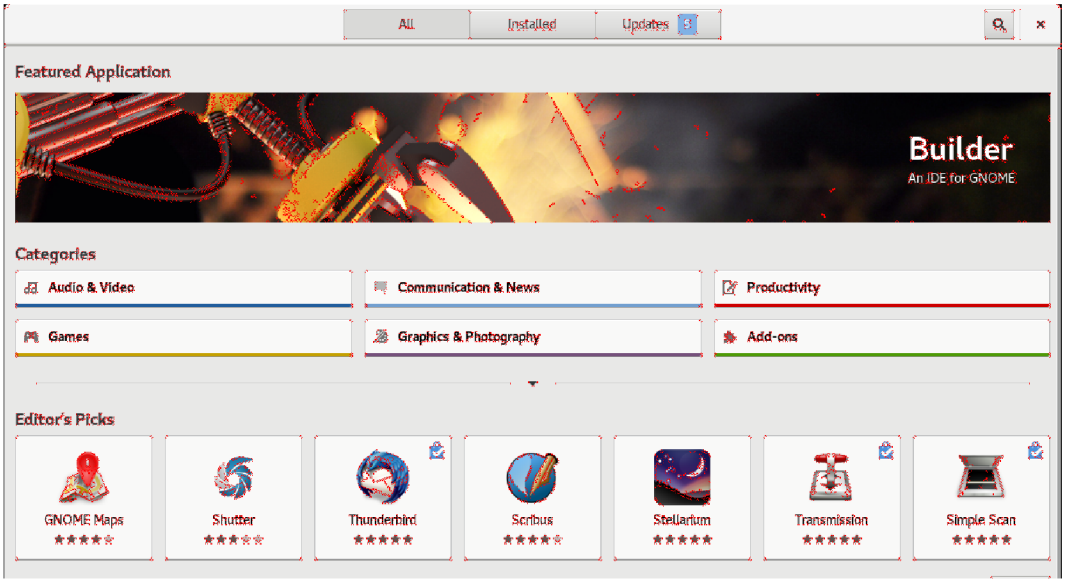
Na obrázku 4.3 je vidieť, že niektoré veľmi malé a pravdepodobne nepotrebné objekty boli vynechané na základe konfigurovanej minimalnej výšky, šírky, či obsahu. Takýchto objektov je mnoho na snímkoch GUI, ktorých pozadia resp. obrázky obsahujú veľa prechodov a kontrastných oblastí.

Nasledovný snímok Gnome 3 GUI je ukážkou mierne problematického GUI.




Obr. 4.5: Snímok GUI obsahujúceho výrazný obrázok.

Na GUI so spomenutými príliš významnými grafickými prvkami nástroj Wirec veľmi vhodný nie je. Vykreslené kontúry na vstupnom snímku je vidieť na nasledujúcom obrázku.




Obr. 4.6: Snímok GUI obsahujúceho výrazný obrázok s vykreslenými kontúrami.

## 4.3 Získavanie informácií o detekovaných objektoch

V ideálnom prípade sú v tejto fáze všetky nepodstatné objekty odstránené a je možné začať so získavaním informácií o detekovaných objektoch. Okrem už získaných informácií o vzťahoch medzi objektami je vhodné získať informácie ako pozadie objektu, zarovnanie oproti rodičovskému objekty, percento plochy rodičovského objektu zabraného objektom a ďalšie.

Informácie o objektoch získané z GUI by mali byť v atomických neinterpretovaných hodnotách, aby sa klasifikátory mohli dostať ku všetkým detailom o objektoch, ktoré by mohli využiť na klasifikáciu. Postupmi ukázanými v tejto kapitole je splnená požiadavka 3 z tabuľky 3.2. Keďže postupy získavania informácie o tvare objektu boli predstavené v predchádzajúcej kapitole, napriek dôležitosti týchto informácií už rozoberané dalej nebudú.

### 4.3.1 Zarovnanie objektu

Vlastnosť zarovnania objektu je získavaná na základe pomeru rozdielov vzdialeností od stien objektu, v ktorom sa analyzovaný objekt nachádza.


Obr. 4.7: Snímok textu zarovnaného na stred rodičovského objektu.

Zarovnanie objektu je udávané v intervale <-1,1>, kde hodnoty blízke k -1 znamenajú zarovnanie do ľava, hodnoty blízke k 1 zarovnanie vpravo a hodnoty v strede intervalu zarovnanie do stredu, podľa vzorca:

$$centered = \frac{x1 - x2}{x1 + x2}$$

Kde:

$$x1 = x - parentX$$

$$x2 = (parentX + parentWidth) - (x + width)$$

Rovnakým spôsobom je získavaná aj hodnota vertikálneho zarovnania.

### 4.3.2 Farba pozadia objektu

V GUI sa môžu vyskytovať viaceré objekty s rovnakými základnymi vlastnosťami ako farba písma, farba pozadia a tvar. Získanie farby pozadia takýchto objektov na základe pixelov nie je jednoduché z dôvodu, že text nachádzajúci sa v spomenutých objektoch môže mať rôznu dĺžku a relatívne významne ovplyvniť celkovú farbu objektu. Preto v rámci nástroja Wirec sú použité dva prístupy k získavaniu farby objektu.

#### Získavanie farby objektu na základe priemeru

Jednoduché získavanie farby objektu je založené na získaní všetkých zložiek farby a následným podielom množstvom pixelov tvoriacich objekt. Ako už bolo spomenuté toto je veľmi náchylné na chyby a preto sa v rámci nástroja Wirec nepoužíva.

#### Získavanie farby objektu na základe váženého priemeru

Získavanie farby objektu na základe pridelenia rôznych váh rôznym oblastiam objektu je presnejšie, no stále nie je dokonale spoľahlivé. Pracuje na predpoklade, že pokiaľ sa získava napríklad pozadie tlačítka, najsmerodajnejšie získavanie farby bude v oblasti približne v polovičnej vzdialenosti stredu vertikálne od kraju objektu, keďže v danej oblasti sa väčšinou kľúčové črty objektu nenachádzajú.


Obr. 4.8: Snímok objektu so zvýraznenými oblasťami najväčšej váhy.

Výpočet vážených hodnôt jednotlivých pixelov, z ktorých je následne zistení vážený priemer je popísaný v algoritme 7.

### 4.3.3 Plocha objektu oproti rodičovskému objektu

Ďalšou informáciou užitočnou pre klasifikáciu je plocha objektu oproti rodičovskému objektu. Táto hodnota je jednoducho získavaná na základe obsahu obdĺžnika opisujúceho kontúry objektu. Avšak objekty v tejto fáze sú horizontálne pospájané a teda informácia o kontúrach spojených objektov nie je správna. Keďže pri spájaní objektov bola zachovaná informácia o súradniciach, šírke a výške spojeného objektu, je jednoducho možné túto informáciu správne využiť.

**Data:** Obrázok na zistenie farby pozadia
**Result:** Trojica hodnôt červená, zelená a modrá
Získaj vzdialenosť stredu najviac ohodnotenej oblasti od stredu
**foreach** *Riadok obrázku objektu* **do**
  **foreach** *Pixel riadku* **do**
    Získaj hodnotu pixelu
    Získaj hodnotu koeficientu na základe vzdialenosti od stredu
    Získaj hodnoty RGB násobením koeficientom
  **end**
**end**
return Priemerná farbu objektu

**Algoritmus 7:** Zistenie priemernej farby objektu.

## 4.4 Klasifikácia

Záverečnou fázou behu nástroja Wirec je klasifikácia, ktorá je konfigurovateľná, no v rámci bakalárskej práce je dodaný iba jeden zásuvný modul klasifikujúci objekty triedy button, radio button, checkbox a text input field. Je možné dodať viaceré zásuvné moduly doplňujúce ďalšie informácie o pravdepodobnej príslušnosti k istej triede objektov. Je treba brať do úvahy získané hodnoty predchádzajúcimi zásuvnými modulami, aby nedošlo k strate získaných klasifikovaných hodnôt. Výsledkom tejto kapitoly je splnenie požiadavky 4 z tabuľky 3.2.

### 4.4.1 Hodnota a interval klasifikácie

Vo fáze klasifikácie je možné konfigurovať maximálnu hodnotu priraditeľnú na základe atribútu a aj interval, na ktorom túto zhodu je možné očakávať.

#### Hodnota klasifikácie

Objekt môže získať maximálnu pravdepodobnosť príslušnosti ku určitej triede objektov v hodnote 1. Preto ak je viacero kritérií stanovených pre príslušnosť ku istej triede objektov, je potrebné túto hodnotu distribuovať medzi všetky kritéria podľa uváženia používateľa.

Vhodným príkladom vyššie uvedeného je klasifikácia triedy button na základe:

- Existencie textu.
- Zarovnaných potomkov na stred.
- Množstvo plochy objektu zabranej potomkami.

Preto je možné hodnotu 1 rozdeliť medzi dané parametre v pomere dávajúcom v súčte 1, napríklad: 0.3:0.3:0.4.

#### Interval klasifikácie

Pre jednoduchý spôsob konfigurácie bolo vhodné použiť stručný spôsob.

Formát pre stanovenie hodnôt a typu intervalu je nasledovný:

$$MINxMAXxTYP$$

Minimálna a maximálna hodnota symbolizujú hraničné hodnoty prípustné pre priradenie určitej hodnoty pravdepodobnosti príslušnosti k určitej triede objektov.

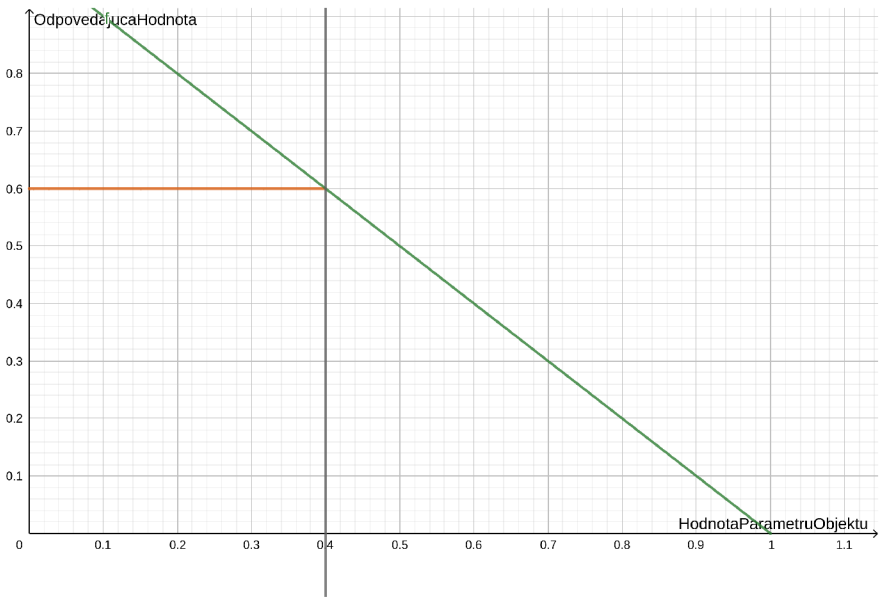
Typ intervalu symbolizuje správanie koeficientu použitého pre priradenie hodnoty objektu. V súčasnosti sú prípustné nasledovné hodnoty parametru “TYP“:

- m - Najlepšia zhoda v strede intervalu stanovenom min. a max. hodnotou. Zhoda rastie lineárne z oboch strán do stredu intervalu.
- l - Najlepšia zhoda pri minimálnej hodnote stanoveného intervalu. Zhoda rastie lineárne k minimálnej hodnote.
- r - Najlepšia zhoda pri maximálnej hodnote stanoveného intervalu. Zhoda rastie lineárne k maximálnej hodnote.
- c - Konštantná zhoda na celom stanovenom intervale.

Príklady:

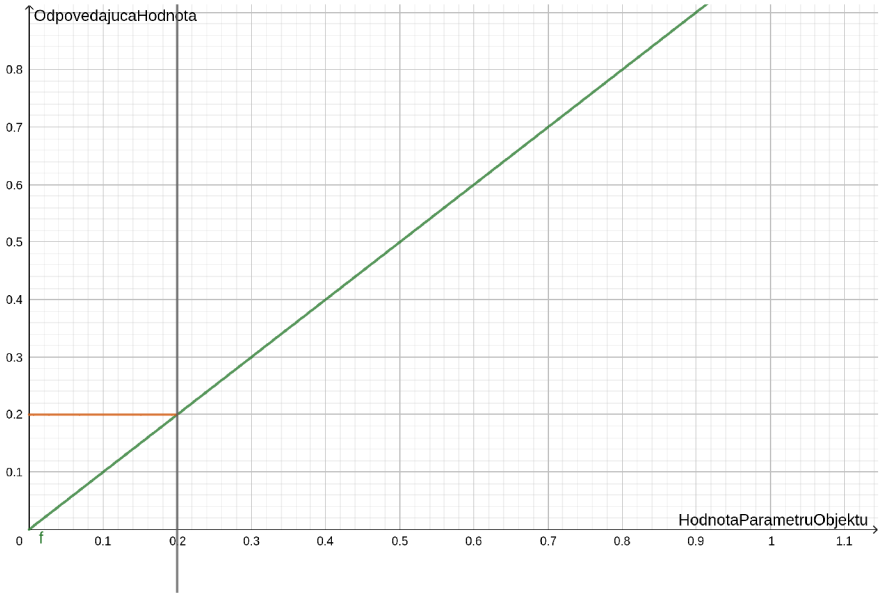
- -1x1xl - Ideálna zhoda atribútu objektu blížiaceho sa k hodnote -1. 0 zhoda pri hodnote atribútu blízkej 1.
- 1x2xr - Ideálna zhoda atribútu objektu blížiaceho sa k hodnote 2. 0 zhoda pri hodnote atribútu blízkej 1.
- 1x2xm - Ideálna zhoda atribútu objektu blížiaceho sa k hodnote 1.5. 0 zhoda pri hodnote atribútu blízkej 1 a 2.
- 1x2xc - Ideálna zhoda na celom intervale, pokiaľ sa atribút objektu nachádza kdekoľvek na danom intervale.

Nasledujú praktické ukážky grafov zobrazenia jednotlivých intervalov. Osa X má význam reálnej hodnoty parametru objektu, ktorý je vyhodnocovaný a osa Y ukazuje odpovedajúcu hodnotu ku danej hodnote parametru. Osa Y sa odkazuje na hodnotu intervalu 4.4.1.

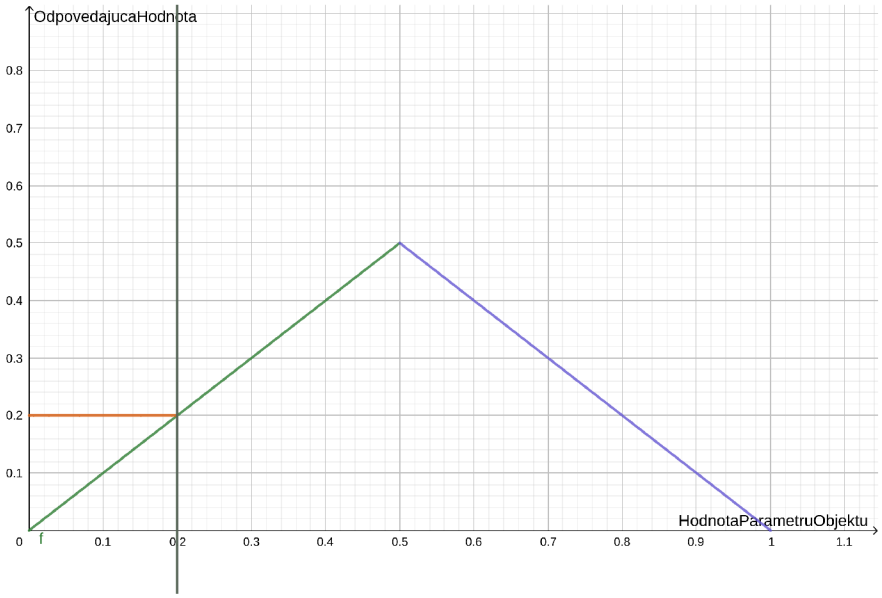



Obr. 4.9: Snímok intervalu 0x1xl - najlepšia zhoda blízko 0.

Na obrázkoch sú názorné ukážky vyhodnocovania intervalu klasifikácie. Napríklad na obrázku 4.9 je zobrazený interval klasifikácie 0x1xl. Je vidieť, že hodnotený parameter objektu (napríklad zarovnanie obsahu na stred malo hodnotu 0.4), bol vyhodnotený na 0.6. Táto hodnota je vynásobená koeficientom, hodnotou klasifikácie, ktorý je maximálne priraditeľný zarovnaniu obsahu na stred a tým je zistený výsledok.




Obr. 4.10: Snímok intervalu 0x1xr - najlepšia zhoda blízko 1.




Obr. 4.11: Snímok intervalu 0x1xm - najlepšia zhoda blízko 0.5.

### 4.4.2 Postup klasifikácie dodaného zásuvného modulu

Prvým krokom klasifikátora je získanie hodnôt prideliteľných objektom na základe výskytu hodnôt vlastností klasifikovaného objektu v zadanom intervale, ako bolo popísane v kapi-

tole 4.4.1. Hodnoty takto získané budú využívané pri vyhodnocovaní každej konfigurovanej oblasti.

Algoritmus klasifikácie spomenutých komponent GUI overuje rôzne informácie získané z predošlých fázy a pre každú takú informáciu vyhodnocuje zhodu volaním funkcie `evaluate_match()`, ktorá zisťuje miesto výskytu v intervale a odpovedajúcu hodnotu ku danej zhode.

**Data:** Informácie zistené o objekte, interval hodnôt, maximálna hodnota priraditeľná objektu, koeficient
**Result:** Priradená hodnota k danej triede objektov
Na základe typu intervalu nájdi hodnotu zhody z intervalu
**if** *Interval je typu priradenia konštantnej hodnoty* **then**
| return koeficient
**else**
| return koeficient * hodnota-zhody-z-intervalu
**end**

**Algoritmus 8:** Vyhodnotenie zhody na základe informácie o objekte.

Keďže klasifikácia objektov, ktoré závisia na interakcií - napríklad radio a checkbox - nie je bez interakcie jednoznačná, nie je odporúčané pri klasifikácií priradzovať veľkú mieru pravdepodobnosti príslušnosti k daným triedam objektov.

### Detekcia triedy button

Jedným z kritérií príslušnosti objektu do triedy button je zarovnanie jeho potomkov na stred podľa konfigurovaného intervalu. Problémom je, že zhoda pre interval je počítaná z priemerného zarovnania všetkých jeho potomkov. To z dôvodu, že je nutné nejak priemerné zarovnanie vypočítať, keďže ide o zarovnanie celého obsahu tlačítka. Problémovým môžu byť napríklad 2 detekované objekty na protiľahlých krajoch plochy tlačítka, čo by bolo vyhodnotené na maximálnu zhodu. Príklad je uvedený na nasledujúcom obrázku:

Obr. 4.12: Snímok objektu s protiľahle umiestnenými objektami v rámci text input field.

Je zrejmé, že ak by obsah obrázku 4.12 bol vyhodnotený ako takmer dokonale zarovnaný na stred, došlo by ku nesprávne priradenej hodnote pravdepodobnosti príslušnosti tohoto objektu ku triede button. Z tohoto dôvodu sú použité viaceré podmienky stanovujúce minimálnu príslušnosť do intervalu, inak nie je možné podľa daného parametra priradiť žiadnu hodnotu pridelenú danému typu parametru.

### Detekcia triedy checkbox

Klasifikácia checkbox vychádza z predpokladu, že jeho potomkovské objekty neobsahujú textové objekty a zároveň obsahujú horizontálne následný objekt vo vzdialenosti zadanej intervalom v konfigurácií. Keďže pravdepodobnosť príslušnosti objektu do triedy objektov checkbox nie je momentálne možné dobre rozpoznať, je použitá konštanta 0.5 pri splnení základných podmienok.

**Data:** Klasifikovaný objekt O
**Result:** Objekt obsahujúci informáciu o pravdepodobnosti príslušnosti k triede button
**if** *O.children == null* **then** return 0 ;
**if** *Objekt obsahuje potomkov obsahujúcich text* **then** pravdepodobnost-tlacitka += evaluate_match(súčet-dĺžok-potomkov, konfiguracia) ;
**if** *Každý potomok patrí zarovnaním do konfigurovaného intervalu* **then** pravdepodobnost-tlacitka += evaluate_match(priemerné-zarovnanie-potomkov, konfiguracia) ;
pravdepodobnost-tlacitka += evaluate_match(súčet-obsahov-potomkov, konfiguracia)

**Algoritmus 9:** Klasifikácia triedy button.

**Data:** Klasifikovaný objekt O
**Result:** Objekt obsahujúci informáciu o pravdepodobnosti príslušnosti k triede checkbox
Na základe typu intervalu nájdi výskyt zhody
**if** *!O.rectangular* **then** return 0 ;
**if** *Potomkovia objektu obsahujú text* **then** return 0 ;
**if** *O.width ≅ O.height and O.horizontally_next_related and O.horizontally_next_dist <= conf.min_dist_h_relation* **then** pravdepodobnost-checkbox-tlacitka += 0.5 ;

**Algoritmus 10:** Klasifikácia triedy checkbox.

### Detekcia triedy radio button

Klasifikácia triedy radio button je náročná v tom, že množstvo objektov môže mať základ podobný trieda radio button. Dodaný klasifikátor vychádza z predpokladu, že objekt triedy radio button musí byť kruhového tvaru a patrí k horizontálne nasledujúcemu objektu vo vzdialenosti zadanej intervalom v konfigurácií, ktorým býva väčšinou text. Keďže pravdepodobnosť príslušnosti objektu do triedy objektov radio button nie je momentálne možné dobre rozpoznať, je použitá konštanta 0.5 pri splnení základných podmienok. Ukážka tohoto prístupu je ukázaná v algoritme 11.

**Data:** Klasifikovaný objekt O
**Result:** Objekt obsahujúci informáciu o pravdepodobnosti príslušnosti k triede radio button
Na základe typu intervalu nájdi výskyt zhody
**if** *O.circle and O.horizontally_next_related and O.horizontally_next_dist <= conf.min_dist_h_relation* **then** pravdepodobnost-radio-tlacitka += 0.5 ;

**Algoritmus 11:** Klasifikácia triedy radio button.

### Detekcia triedy text input field

Klasifikácia text input field je najriskantnejšou zo všetkých spomenutých, pretože predpokladom pre jej funkčnosť je len zarovnanie potomkov a súčet obsahov potomkov. Text input field často obsahuje málo výrazný text a kliknutím na oblasť text input field sa jeho stav úplne zmení. Spoliehať sa na detekovanie textu pravdepodobne nie je správna cesta, pretože

také texty bývajú často málo kontrastné a ťažko rozpoznateľné. Preto text input field je jedna z komponent, ktorej klasifikácia zaručene musí byť dodatočne interakciou overená.

**Data:** Klasifikovaný objekt
**Result:** Objekt obsahujúci informáciu o pravdepodobnosti príslušnosti k triede text input field
**if** *Niektorý z potomkov nepatrí zarovnaním do konfigurovaného intervalu* **then**
return 0 ;
pravdepodobnost-input-field +=
evaluate_match(priemerne-zarovnanie-potomkov-vlavo, konfiguracia)
pravdepodobnost-input-field +=
evaluate_match(priemerne-zarovnanie-potomkov-dohora, konfiguracia)
pravdepodobnost-input-field += evaluate_match(sucet-obsahov-potomkov, konfiguracia)

**Algoritmus 12:** Klasifikácia triedy text input field.

### 4.4.3 Generovanie výstupu

Výstupom je vygenerovaný JSON súbor obsahujúci zoznam detekovaných objektov, kde každý objekt má svoje ID, a väčšinu informácií získaných zo všetkých fázy behu nástroja Wirec. Tento výstup môže byť použitý napríklad na upravenie hodnôt a spustenie nástroja len na fázu klasifikácie, prípadne na porovnanie dvoch rôznych stavov detekovaných nástrojom Wirec. Týmto bude splnená požiadavka 8 a 9 z tabuľky 3.2.

## 4.5 Komparácia

Voliteľnou fázou nástroja Wirec, je komparácia, v ktorej je možné porovnať výstup práve ukončeného behu s výstupom iného behu nástroja. Táto trieda zásuvných modulov môže nájsť využitie napríklad pri overovaní klasifikácie pomocou interakcie s GUI, vykonaním akcie a vytvorením snímku a jeho následným porovnaním. V nasledujúcich podkapitolách sú ukázané 2 možné prístupy.

### 4.5.1 Porovnanie JSON výstupu

Prvý prístup je založený na porovnávaní vygenerovaných JSON výstupov, kde je možné porovnávať textové reprezentácie jednotlivých oblastí (hash), pomocou porovnávania objektov jedného behu s objektami druhého behu s rovnakou pozíciou.

### 4.5.2 Porovnanie vstupných obrázkov

Príkladom môže byť porovnanie vstupných obrázkov rôznych dvoch behov pomocou odčítania, ktorým je možné ľahko zistiť všetky nerovnaké oblasti. Tieto jednotlivé detekované rozdiely sú znova detekované ako samostatné objekty a pre každý z nich je vygenerovaný výstupný obrázok vo výstupnom adresári nástroja Wirec.

# Kapitola 5

# Overenie funkcionality rozpoznávania

Funkcionalita rozpoznávania GUI nástroja Wirec bola overovaná na umelej testovacej sade v podobe snímkov všetkých štýlov frameworku Bootstrap na jednej stránke a tiež na reálnej komerčnej aplikácií GitHub pomocou testovacieho nástroja Behave v jazyku Python.

Testovanie zahŕňa jednotkové testy v podobe testovania funkcionality detekcie tvaru a klasifikácie a integračné testy v podobe testovania množstva správne detekovaných objektov.

## 5.1 Testy na aplikácií používajúcej Bootstrap

V prílohe C sú snímky GUI používajúce Bootstrap kaskádové štýly [1]. Tieto snímky sú vhodné na umelé testovanie, keďže obsahujú široké spektrum rôznych štýlov komponent, na ktorých rozpoznávanie sa nástroj Wirec zameriava. Nástroj Wirec použil na vstupe snímok a jeho výstupom bol snímok obohatený o komponenty so zafarbenými obrysmy znázorňujúce príslušnosť daných detekovaných objektov ku triedam.

Časť konfigurácie týkajúcej sa klasifikácie pri vstupe Bootstrap snímku:

| **Parameter** | **hodnota** | **Parameter** | **hodnota** |
|---|---|---|---|
| button_centered | 0.4 | textarea_centered_left | 0.35 |
| button_centered_int | -0.3x0.3xm | textarea_centered_left_int | -1x0xl |
| button_text | 0.3 | textarea_centered_top | 0.35 |
| button_text_int | 1x30xc | textarea_centered_top_int | -1x0.2xm |
| button_size | 0.3 | textarea_size | 0.3 |
| button_size_int | 0.0x0.6xc | textarea_size_int | 0.0x0.2xm |
| checkbox | 0.5 | radio_button | 0.5 |

Tabuľka 5.1: Tabuľka použitej konfigurácie pri vstupe snímku Bootstrap.

### 5.1.1 Nepresnosti klasifikácie

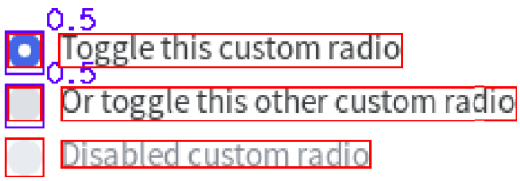
Generovaný výstup bol porovnaný so skutočnými vlastnosťami vstupu, no keďže kritériá pre príslušnosť k triedám button, radio button, checkbox a text input field nie sú dostatočne flexibilné, výsledok sa líši od očakávaného. Nie je totiž možné len na základe poznania pixelov stanoviť kritéria klasifikácie dostatočne presne, aby bol nástroj bez interakcie schopný klasifikovať objekty tak, ako je schopný človek pri bežnom používaní.

Výsledky porovnánia sú zobrazené v nasledujúcej tabuľke. Percentuálne vyjadrenie správne klasifikovaných objektov vyjadruje koľko objektov bolo klasifikovaných podľa očakávania. Nesprávna klasifikácia objektov v skutočnosti nepatriacich do danej triedy, alebo nedostatočná klasifikácia skutočne patriacich objektov do danej triedy sú v tomto ukazateli zohľadnené:

| **Typ komponenty** | **Správne klasifikovaných** | **Minimálne prijateľné ohodnotenie pravdepodobnosti** |
|---|---|---|
| Button | 85% | 0.9 |
| Checkbox | 100% | 0.5 |
| Radio button | 80% | 0.5 |
| Text input field | 50% | 0.7 |

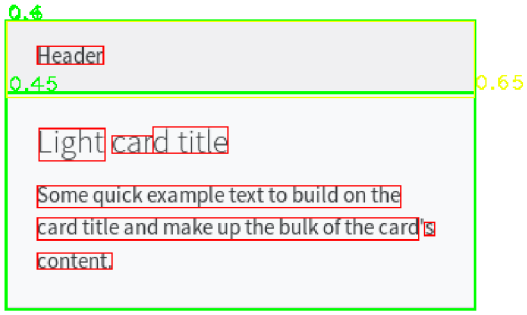
Tabuľka 5.2: Štatistiky úspešnosti aplikácie na Bootstrap obrázku.

V tabuľke je vidieť, že sa počty získané nástrojom Wirec nezhodujú s počtami reálnymi. Nasledujúcimi obrázkami bude vysvetlené, na aké problémy je možné naraziť.




Obr. 5.1: Snímok nesprávne detekovaného objektu triedy radio button.

Na obrázku 5.1 je znázornené rozpoznanie prvých dvoch objektov triedy radio button, tretí nebolo možné rozoznať z dôvodu príliš nízkeho kontrastu.

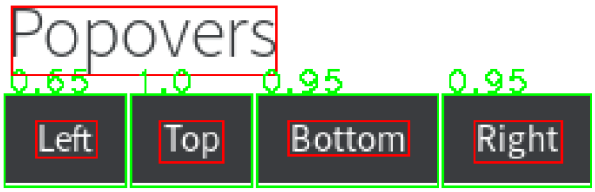



Obr. 5.2: Snímok nesprávne detekovaného bootstrap panel komponent.

Nadpis panelu bol vyhodnotený ako text input field na 65%, keďže text je zarovnaný vľavo, neobsahuje žiadne súrodenecké objekty, ktoré by zarovnanie narušovali a tiež spĺňa kritérium pomeru obsahu oproti rodičovskému objektu. Tento nesprávne klasifikovaný text input field je rovnako nutné vyvrátiť interakciou s GUI.

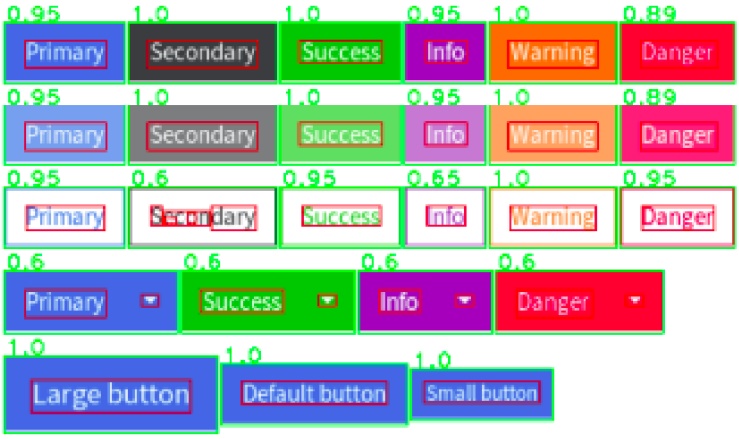

Obr. 5.3: Snímok nesprávne detekovaného objektu triedy button.

Na obrázku 5.3 je vidieť, že daná komponenta spĺňa predpoklady na to, aby bola klasifikovaná do triedy button a zároveň do triedy text input field. Pravá funkcionalita je znova overiteľná interakciou.




Obr. 5.4: Nedostatočne klasifikovaný objekt triedy button.

Tri zo štyroch tlačítok boli veľmi vysoko ohodnotené ako tlačítka, no obsah prvého tlačítka nebol rozoznaný pomocou OCR ako text, preto hodnota klasifikácie je nižšia.




Obr. 5.5: Snímok nedostatočne klasifikovaných tlačítok.

Text všetkých tlačítok okrem prvých dvoch párnych tlačítok tretieho riadku bol rozpoznaný pomocou OCR, v tomto prípade ale tlačítka predposledného riadku porušujú pravidlo zarovnania všetkých podobjektov komponenty, keďže pravý potomok je veľmi výrazne zarovnaný vpravo, čím zamedzuje pridanie hodnoty klasifikátora pre zarovnanie na stred podľa konfigurácie.

### 5.1.2 Zhodnotenie

Demonštráciou niektorých výnimočných prípadov GUI štýlovaného pomocou Bootstrap bolo ukázané, že nástroj Wirec vo všeobecnosti na štandardne štylovaných GUI pracuje správne, hoci väčšinu klasifikovaných komponent je dobré interaktívne overiť, keďže nie všetky sú bez kontextu jednoznačne klasifikovateľné.

## 5.2 Testy na aplikácií GitHub

Funkcionalita overovaná na aplikácií GitHub potrebovala zmeny v konfigurácií, keďže ako bude neskôr ukázané, obsahuje komponenty spĺňajúce kritériá pre príslušnosť do triedy text input field. Z kontextu je ale zjavné, že sa jedná skôr o tlačítka - odkazy v menu.

Zmena v konfigurácií je závislá na štandardnom štýlovaní GUI, na ktoré sa nástroj Wirec použije, a v tomto prípade je nutné nastaviť interval klasifikácie tak, aby bral do úvahy pri klasifikácií tlačítok aj tlačítka, ktoré majú obsah zarovnaný vľavo a nie len na stred.

Časť konfigurácie pre klasifikáciu vyzerá teda nasledovne:

| **Parameter** | **hodnota** | **Parameter** | **hodnota** |
|---|---|---|---|
| button_centered | 0.4 | textarea_centered_left | 0.35 |
| button_centered_int | -1x0.3xm | textarea_centered_left_int | -1x0xl |
| button_text | 0.3 | textarea_centered_top | 0.35 |
| button_text_int | 1x30xc | textarea_centered_top_int | -1x0.2xm |
| button_size | 0.3 | textarea_size | 0.3 |
| button_size_int | 0.0x0.6xc | textarea_size_int | 0.0x0.2xm |

| | | | |
|---|---|---|---|
| checkbox | 0.5 | radio_button | 0.5 |

Tabuľka 5.3: Tabuľka použitej konfigurácie pri vstupe snímku GitHub.

Výsledky porovnánia sú zobrazené v nasledujúcej tabuľke. Percentuálne vyjadrenie správne klasifikovaných objektov vyjadruje koľko objektov bolo klasifikovaných podľa očakávania. Nesprávna klasifikácia objektov v skutočnosti nepatriacich do danej triedy, alebo nedostatočná klasifikácia skutočne patriacich objektov do danej triedy sú v tomto ukazateli zohľadnené:

| **Typ komponenty** | **Správne klasifikovaných** | **Minimálne prijateľné ohodnotenie pravdepodobnosti** |
|---|---|---|
| Tlačítko | 77% | 0.65 |
| Checkbox | 100% | 0.5 |
| Text input field | 25% | 0.65 |

Tabuľka 5.4: Štatistiky úspešnosti aplikácie na GitHub obrázku.

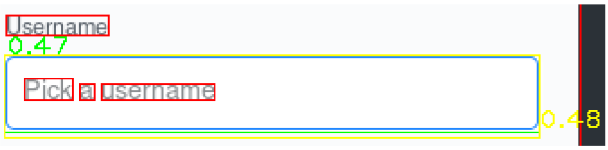
Keďže štýly použité na aplikácií GitHub sú pestrejšie, boli použité aj nižšie akceptovateľné hodnoty pri tvorení štatistík.

### 5.2.1 Nepresnosti klasifikácie

Pri spustení nástroja Wirec so vstupným obrázkom 3.3 je výstup relatívne uspokojivý, no nachádzajú sa v ňom nasledujúce nepresnosti:


Obr. 5.6: Snímok nesprávne klasifikovaného loga.

Logo aplikácie kruhového tvaru je vyhodnotené ako objekt triedy radio button, kvôli tomu, že spĺňa prednastavené kritérium vzdialenosti nasledujúceho objektu vo vzťahu. Takúto nepresnosť je možné vyriešiť znížením maximálnej povolenej vzdialenosti objektov vo vzťahu, alebo interakciou s GUI, pre overenie klasifikácie.

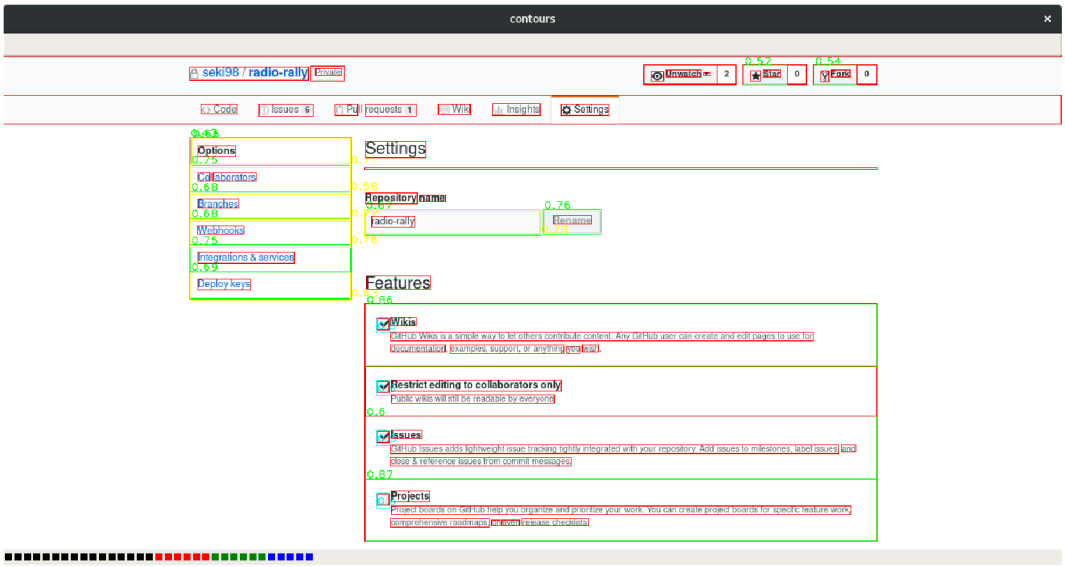



Obr. 5.7: Snímok nedostatočne presne klasifikovaného text input field.

Príslušnosť objektu k triede text input field je len o niečo vyššia ako príslušnosť ku triede button. Nepresnosť je spôsobená tým, že veľká váha je dávaná detekcií textu pri klasifikácií triedy button a tiež zarovnaniu nahor pri klasifikácií text input field. Konfiguráciou alebo interakciou s GUI je možné klasifikáciu upresniť.


Obr. 5.8: Snímok nedostatočne klasifikovaného tlačítka.

Objekt s jednoznačnými črtami tlačítka je klasifikovaný ako objekt triedy button len na 78% z dôvodu konfigurácie, ktorá počíta s najlepšou zhodou v strede intervalu medzi zarovnaním vľavo a zarovnaním v strede. Upresnenie nie je možné, z dôvodu charakteru GUI, ktorý počíta s obomi zarovnaniami.




Obr. 5.9: Snímok nedostatočne klasifikovaných tlačítok.

Na ľavej strane ďalšieho GitHub snímku je možné vidieť menu s viacerými položkami spĺňujúce kritéria pre príslušnosť triedam button a aj text input field. Na tomto príklade je možné vidieť, že existuje ešte veľký priestor pre zlepšenie nástroja Wirec, ako napríklad zisťovanie informácie o farbe textu a rozdiel medzi farbou textu a jeho pozadím. Týmto by bolo možné stanoviť ďalšie kritérium - množstvo odtieňov farby minimálne potrebne na to, aby bol objekt triedy button rozoznaný od objektu triedy text input field.

Ďalšou nepresnosťou je klasifikácia tlačítok v pravom hornom rohu, no keďže tlačítka obsahujú viacero podobjektov rôzne zarovnaných, nie je možné ich s istotou klasifikovať správne.

Poslednou zásadnou nepresnosťou klasifikácie sú panely pod nadpisom "Features", ktoré sú s vysokou mierou ohodnotené ako tlačítka. No keďže sa v priebehu návrhu, vývoja a testovania nevynašiel spôsob, ktorým by bolo úplne jednoznačne možné klasifikovať objekt triedy button bez akýchkoľvek "False positives"[1] a bez interakcie, je nutné sa s týmto výsledkom uspokojiť.

### 5.2.2 Zhodnotenie

Overenie funkcionality nástroja Wirec na reálnej aplikácií GitHub bolo osvedčené s uspokojivými výsledkami, hoci niektoré komponenty ukázané v tejto podkapitole ukazujú mierne nedostatky. Jedným z hlavných nedostatkov je klasifikácia objektov do tried, do ktorých ľudským okom jednoznačne nepatria.

## 5.3 Testovacia zostava

Všetky experimentálne a testovacie spustenia nástroja Wirec boli uskutočnené na stroji s týmito parametrami:

- Intel®Core™i5-7200U CPU @ 2.50GHz x 4.
- Intel®HD Graphics 620.
- 8GB RAM.
- Operačný systém - Fedora 27.

[1] Nesprávne vyhodnotené za pravdivé

# Kapitola 6

# Závěr

Práca bola zameraná na detekciu objektov GUI, získavanie maximálneho možného množstva informácií o detekovaných objektoch a následne na klasifikáciu detekovaných objektov na základe získaných informácií a zadanej konfigurácie.

Pri návrhu bol kladený dôraz na rozšíriteľnosť nástroja, čo sa uskutočnilo pomocou rozhrania poskytujúceho možnosť definovania vlastných zásuvných modulov uzavretých do skupín nezávislých na sebe. Ďalšou dôležitou vlastnosťou implementovaného riešenia je relatívna rýchlosť, ktorou dokáže bez prípravy klasifikovať zatiaľ malé množstvo tried komponent GUI bez nutnosti predchádzajúceho zberu testovacích dát a učenia ako je tomu napríklad pri neurónových sieťach. Aplikácia je čiastočným prienikom nástrojov založených na základe analýzy pixelov a na základe poznania objektov GUI.

Implementované riešenie vykazuje nepresnosti vo fáze detekcie a klasifikácie objektov, čo v prípade použitia neštandardných štýlov pri tvorení GUI môže viesť k vynechaniu detekcie niektorých objektov, alebo nesprávnemu priradeniu pravdepodobnosti príslušnosti detekovaného objektu k triede.

Aplikácia bola navrhovaná tak, aby bolo možné jej výstupy jednoducho použiť pre vstupy ďalších aplikácií a tiež tak, aby bolo možné pristupovať k jednotlivým fázam behu, čím je možné docieliť spresnenie výstupu. Použitím ďalších nástrojov používajúcich interakciu s GUI je možné veľmi značne spresniť klasifikáciu nástroja Wirec, prípadne vyvrátiť nesprávnu klasifikáciu problematických komponent.

Výhodou implementovaného riešenia je prenositeľnosť na všetky systémy obsahujúce potrebné závislosti. Týmito systémami sú operačné systémy používajúce Linux, ako aj operačný systém Windows.

Výstup aplikácie je napriek jeho nedostatkom možné použiť samostatne a to aj vďaka jednoduchej konfigurovateľnosti, ktorou je možné ovplyvniť klasifikáciu GUI s jedinečným použitím štýlov. Nedostatky práce poskytujú veľký priestor na budúcu činnosť, predovšetkým v rozšírení množstva klasifikovaných tried objektov, tiež v doplnení ďalších prístupov detekcie objektov, ako aj získavaní zatiaľ nezisťovaných informácií o detekovaných objektoch.

# Literatúra


[1] *Bootswatch: Free themes for Bootstrap*. [Online; navštíveno 08.04.2018].
URL https://bootswatch.com/

[2] *GitHub*. [Online; navštíveno 08.04.2018].
URL https://github.com/

[3] *GitHub Login Page*. [Online; navštíveno 08.04.2018].
URL https://github.com/login

[4] *Image Gradients*. [Online; navštíveno 08.04.2018].
URL https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_gradients/py_gradients.html

[5] *OpenCV Template Matching*. [Online; navštíveno 08.04.2018].
URL
https://docs.opencv.org/trunk/d4/dc6/tutorial_py_template_matching.html

[6] *Skupina Testos. Domovská stránka projektu Testos*. [Online; navštíveno 08.04.2018].
URL http://testos.org

[7] *Testos / Aggt - GitLab*. [Online; navštíveno 08.04.2018].
URL https://pajda.fit.vutbr.cz/testos/aggt

[8] *Testos / Wirec - GitLab*. [Online; navštíveno 08.04.2018].
URL https://pajda.fit.vutbr.cz/testos/wirec

[9] Aggarwal, R. M. . D. H.: *Study and Comparison of Various Image Edge Detection Techniques* . [Online; navštíveno 08.04.2018].
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.927&rep=rep1&type=pdf

[10] Bradski, G.; Kaehler, A.: *Learning OpenCV*. O'REILLY, 2008, ISBN 978-0-596-51613-0.

[11] Eikhout, V.: *An Overview on Template Matching Methodologies and its Applications* . International Journal of Research in Computer and Communication Technology, Vol 2, Issue 10, 2013, ISBN 2278- 5841.

[12] Hocke, R.: *Hello World (Windows) - Sikuli X 1.0 documentation*. [Online; navštíveno 08.04.2018].
URL http://doc.sikuli.org/tutorials/helloworld/helloworld-win.html

[13] Hocke, R.: *SikuliX Documentation*. [Online; navštíveno 08.04.2018].
URL
https://media.readthedocs.org/pdf/sikulix-2014/latest/sikulix-2014.pdf

[14] Melo, J.: *Application for Generating GUI Test Suite*. Diplomová práce, Brno University of Technology, Faculty of Information Technology, 2013.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=14063

[15] Muthukadan, B.: *Selenium with Python*. [Online; navštíveno 08.04.2018].
URL http://selenium-python.readthedocs.io/index.html

[16] Rhody, D. H.: *Circle Hough Transform (CHT)*. [Online; navštíveno 29.04.2018].
URL
https://www.cis.rit.edu/class/simg782.old/talkHough/HoughLecCircles.html

# Príloha A

# Používateľská príručka a API

Po tom čo je aplikácia nainštalovaná pomocou inštrukcií v prílohe B, je možné beh aplikácie ovplyvniť bez zásahu kódu pridaním nových, alebo úpravou existujúcich zásuvných modulov.

## A.1 Manipulácia so zásuvnými modulmi

Zásuvné moduly sa nachádzajú v jednom zo štyroch existujúcich adresárov:

- `classifiers/`
- `comparators/`
- `detectors/`
- `property_extractors/`

Poradie ich spustenia je zobrazené v diagrame 3.4. Voliteľným typom zásuvných modulov v rámci nástroja Wirec sú zásuvné moduly typu "comparators". Tieto sú spustené ako posledné a na svoju funkcionalitu potrebujú stav GUI iného behu nástroja Wirec vo forme JSON, alebo vstupný obrázok iného behu. Inak nie je v súčasnej implementácií možné porovnať dva rôzne stavy GUI.

### A.1.1 Inštalácia zásuvného modulu

Každý zásuvný modul musí byť nainštalovaný nakopírovaním do príslušného adresára a tiež pridaním referencie naňho v súbore `setup.py`. Súčasný stav zásuvných modulov v tomto súbore je:

```
entry_points={
  [...]
  'detectors': [
    'laplacian = wirec.detectors.laplacian:Laplacian',
  ],
  'property_extractors': [
    'tesseract = wirec.property_extractors.tesseract:Tesseract',
    'relations = wirec.property_extractors.relations:Relations',
    'hash = wirec.property_extractors.hash:Hash',
```

```
  ],
  'classifiers': [
    'button = wirec.classifiers.button:Button',
  ],
  'comparators': [
    'hash = wirec.comparators.hash:Hash',
  ],
},
```

Teda každý nainštalovaný zásuvný modul je identifikovaný reťazcom a cestou ku modulu a triede obsahujúcej funkciu triedy *run()* vo tvare:

*názov-zásuvného-modulu = wirec.adresár-triedy-modulov.modul:Trieda-s-funkciou-run*

### A.1.2 Povinný obsah zásuvného modulu

Modul môže obsahovať ľubovoľnú funkcionalitu, ale trieda, na ktorú je v súbore `setup.py` odkazované, musí obsahovať funkciu *run()*, ktorá je vstupným bodom funkcionality modulu. Rozhranie funkcie je rôzne pre triedy modulov:

| **Trieda zásuvného modulu** | **Rozhranie** |
|---|---|
| `detectors` | `def run(cls, conf, input_image):` |
| `property_extractors` | `def run(cls, conf, objects, input_image):` |
| `classifier` | `def run(cls, conf, objects):` |
| `comparators` | `def run(cls, conf, objects, input_image):` |

Tabuľka A.1: Tabuľka rozhraní zásuvných modulov.

| **Trieda zásuvného modulu** | **Rozhranie** |
|---|---|
| `cls` | Referencia na triedu zásuvného modulu obsahujúcu funkciu *run()* |
| `conf` | Inštancia konfigurácie z modulu *wirec/config.py:Configuration* |
| `input_image` | Inštancia vstupného obrázku typu *numpy.ndarray* |
| `objects` | Zoznam detekovaných objektov typu *wirec/objects.py:GuiObject* |

Tabuľka A.2: Tabuľka významu premenných zásuvných modulov.

**Príklad obsahu funkcie run**

Funkcia run má v rôznych moduloch prístup ku spomenutým parametrom. Je dôležité, aby zásuvný modul tieto parametre len obohatil o nové informácie a neodstraňoval nič z doposiaľ získaných informácií, aby nedošlo ku znehodnoteniu výstupov predošlých zásuvných modulov. Príklad zásuvného modulu:

```
import logging
import numpy

from wirec.plugin import Plugin
logger = logging.getLogger('default_logger')

class Checkbox(Plugin):
        speed = 2
        @classmethod
        def classify(cls, conf, objects):
                for o~in objects:
                        o.new_property = 'value'
                return objects

        @classmethod
        def run(cls, conf, objects):
                logger.info('Running Checkbox classifier')
                objects = cls.classify(conf, objects)
                return objects
```

Týmto prístupom je možné pomocou rôznych zásuvných modulov nezávisle od ostatných získavať nové informácie, ktoré v konečnom dôsledku obohatia celý výstup aplikácie.

## A.2 Rozhranie triedy Wirec

Štandardným spôsobom inicializácie aplikácie je zavolanie konštruktora triedy Wirec, ktorý prijíma parametre konfigurácie a vstupného obrázku typu *numpy.ndarray*:

```
def __init__(self, conf, image):
```

Po inicializácií je spustená metoda *Wirec.run()*, v ktorej sú postupne všetky triedy zásuvných modulov spustené. V prípade potreby vyhnutia sa tomuto rozhraniu je možné pristupovať k rozhraniu triedy Wirec nasledovne:

| **Typ funkcie** | **Rozhranie** |
|---|---|
| property_extractors runner | `run_property_extractors(cls, conf, objects, input_image):` |
| classifier runner | `run_classifiers(cls, conf, objects):` |
| comparators runner | `run_comparators(cls, conf, objects, input_image):` |
| GuiObject loader | `load_objects_from_json(conf, json_list)` |
| output saver | `store_json(conf, objects)` |

Tabuľka A.3: Tabuľka rozhraní verejných funkcií Wirec.

| **Návratový typ** | **Rozhranie** |
|---|---|
| zoznam `GuiObject` objektov | `run_property_extractors(cls, conf, objects, input_image):` |
| zoznam `GuiObject` objektov | `run_classifiers(cls, conf, objects):` |
| zoznam `GuiObject` objektov | `run_comparators(cls, conf, objects, input_image):` |
| zoznam `GuiObject` objektov | `load_objects_from_json(conf, json_list)` |
| null | `store_json(conf, objects)` |

Tabuľka A.4: Tabuľka návratových typov funkcií tabuľky A.3.

| **Parameter funkcie** | **Význam** |
|---|---|
| `cls` | Referencia na inštanciu triedy Wirec |
| `conf` | Inštancia konfigurácie z modulu `wirec/config.py:Configuration` |
| `input_image` | Inštancia vstupného obrázku typu *numpy.ndarray* |
| `objects` | Zoznam detekovaných objektov typu GuiObject |
| `json_list` | Zoznam objektov GuiObject vo forme JSON |

Tabuľka A.5: Tabuľka významu parametrov rozhrania triedy Wirec z tabuľky A.4.

| **Trieda a názov funkcie** | **Význam a návratová hodnota** |
|---|---|
| `CLI.parse_arguments([])` | Spracovanie command line argumentov vo forme zoznamu v tvare `[--param, value]`. Vracia spracované argumenty vo forme `Namespace` objektu Python modulu `parser`. |
| `Configuration.get_config(conf)` | Spracovanie parametru `conf` získaného z volania metódy `CLI.parse_arguments()`. Vracia inštanciu konfigurácie z modulu `wirec/config.py:Configuration` |

Tabuľka A.6: Tabuľka rozhrania triedy Configuration a CLI.

Typ `GuiObject` použitý v tabuľkách odkazuje na `wirec/objects.py:GuiObject`. Prístup k týmto funkciám je možný individuálne a pre získanie konfigurácie a zoznamu objektov požadovaných typov je potrebné nasledovné:

```
# Vytvorenie instancie konfiguracie s~moznymi command line parametrami
conf = CLI.parse_arguments(['--results_dir', 'wirec_output',
'--threshold', '10'])
conf = Configuration.get_config(conf)

# Konvertovanie objektov z~json suboru
# do pozadovaneho zoznamu GuiObject objektov
objs = Wirec.load_objects_from_json(conf, json.load(open('objects.json')))

# Nezavisle spustenie klasifikatorov
objs = Wirec.run_classifiers(conf, objs)

# Ulozenie vystupu
Wirec.store_json(objs, conf)
```

Týmto spôsobom je možné spustiť nástroj Wirec so zoznamom detekovaných objektov vo forme json súboru (získaného napríklad predošlými spusteniami) a pristupovať k jednotlivým fázam samostatne. Takýto prístup môže byť vhodný napríklad v prípade, keď je potrebné spustiť len niektorú z fáz behu a vyhnúť sa fázam, ktoré sú v danej situácií nepotrebné.

## A.3 Rozhranie pre Testos Bus

Testos Bus (zbernica Testos [6]) je nástroj riadiaci komunikáciu medzi nástrojmi, ktoré sú súčasťou platformy Testos. Testos Bus umožňuje dva spôsoby komunikácie. Pomocou tzv. signálov, kde koncový bod odošle správu a neočakáva odpoveď a pomocou tzv. žiadostí, kde koncový bod po odoslaní správy očakáva odpoveď na túto správu.

Každý nástroj má možnosť špecifikovať dvojice hodnôt - kľúč hodnota - ktoré vyžaduje na daný typ žiadosti či signálu. Nástroj Wirec poskytuje rozhrania 2 typy žiadostí, na ktoré odpovedá.

### A.3.1 org.testos.wirec.run

Wirec očakáva pri žiadosti na túto cestu dvojice:

- config - Wirec konfigurácia vo forme JSON ukázanej v kapitole 4.1.1.
- image - Vstupný obrázok typu `numpy.ndarray`.

Po kompletnom behu nástroja Wirec, nástroj vracia dvojice:

- image - Výstupný obrázok typu `numpy.ndarray`.
- objects - JSON objekt pre každý klasifikovaný objekt.

### A.3.2 org.testos.wirec.reclassify

Wirec očakáva pri žiadosti na túto cestu dvojice:

- config - Wirec konfigurácia vo forme JSON ukázanej v kapitole 4.1.1.
- objects - JSON objekt pre každý klasifikovaný objekt.

Po spustení fáze klasifikácia so zadanými objektami nástroj Wirec vracia dvojice:

- objects - JSON objekt pre každý novo klasifikovaný objekt.

### A.3.3 org.testos.vnc.run

V rámci bakalárskej práce bol implementovaný adaptér poskytujúci komunikáciu s klientom VNC[1]. Rozhranie bolo pripravené, no nie dokončené, z dôvodu neexistujúceho Python rozhrania pre komunikáciu s Testos zbernicou.

VNC klient očakáva pri žiadosti na túto cestu dvojice:

- config - Konfigurácia vo forme JSON.
- commands - Zoznam príkazov pre VNC klienta.

Odpoveďou na žiadosť je nasledovná dvojica:

- images - Zoznam vytvorených snímkov GUI typu `numpy.ndarray`.

Príklad vstupu pre VNC klienta:

```
[
{
  'config':
  {
    'host': 'sample_remotehost',
    'user': 'sample_user',
  }
  'args':
    {
      'commands': [['lsuper'],['s', 'e', 't', 't', 'i', 'n', 'g', 's'],
      ['enter'], ['take_screenshot']],
    }
}
]
```

Príklad vstupu zadá postupnosť krokov, pre vyhľadanie, otvorenie a zosnímanie nastavení systému.

[1] Virtual Network Computing

# Príloha B

# Demonštrácia použitia aplikácie

V tejto kapitole budú ukázané praktické príklady použitia aplikácie a tiež jeho inštalácia.

## B.1 Závislosti

Aplikácia vyžaduje nasledovné systémové závislosti, ktoré musia byť nainštalované manuálne.

### B.1.1 Operačný systém Fedora/CentOS/RHEL

```
python3 python3-devel git gcc libxml2-devel python3-virtualenv tesseract
gcc-g++ tesseract-devel libSM make
```

### B.1.2 Operačný systém Debian/Ubuntu

```
python3 python3-dev git gcc tesseract-ocr libtesseract-dev libleptonica-dev
   build-essential libdpkg-perl libxml2-dev virtualenv libsm6 make
```

### B.1.3 Závislosti na Python balíkoch

Tieto závislosti môžu byť nainštalované z PyPI[1] rôznymi neskôr popísanými spôsobmi: `progress behave pyhamcrest vncdotool pillow opencv-python tesserocr`

## B.2 Možnosti inštalácie

### B.2.1 Virtuálne prostredie

Odporúčaný postup inštalácie balíka je vytvorenie virtuálneho prostredia pomocou balíku `virtualenv` príkazom:

```
$ virtualenv --system-site-packages wirec
# Aktivacia virtualneho prostredia
$ source wirec/bin
# Aktivacia je zobrazena pred "command prompt"
(wirec) $
```

[1] Python Package Index

Python závislosti môžu byť do virtuálneho prostredia nainštalované manuálne pomocou binárneho súboru `pip` z virtuálneho prostredia, ale budú doňho v prípade neprítomnosti nainštalované automaticky pri inštalácií aplikácie wirec použitím:

V súčasnosti je potrebné v prostredí Ubuntu/Debian nainštalovať tesserocr Python balík manuálne, z dôvodov nekompatibility verzií v balíčkovom systéme Ubuntu, klonovaním repozitára https://github.com/sirfz/tesserocr a spustením príkazu:

```
(wirec)$ git clone https://github.com/sirfz/tesserocr
(wirec)$ pip install tesserocr/
```

Následne je potrebné inštalovať ostatné Python závislosti projektu:

```
(wirec)$ pip install $(cesta k~adresaru projektu wirec)
```

V tejto chvíli je možné nástroj spustiť príkazom:

```
(wirec)$ wirec --filepath img.png
```

Testy je možné spustiť príkazom `make test` v adresári projektu.

### B.2.2 Manuálna inštalácia

Python závislosti je možné inštalovať použitím systémového binárneho súboru `pip`, ale nie je to odporúčané, keďže je možné, že niektoré zo závislostí potrebujú ďalšie závislosti vo verziách iných, aké používa systém, čo môže spôsobiť rôzne problémy. Po inštalácií všetkých závislostí, je možné spustiť priamo cez Python skript `wirec.py`:

```
$ python3 --filepath img.png
```

## B.3 Použitie

Oboma spomenutými spôsobmi je možné použiť parametre terminálového rozhrania spomenuté v tabuľke 4.6. Príklad:

```
(wirec)$ wirec --filepath img.png --results_dir wirec_output/
```

Pričom je používateľ o priebehu informovaný výstupom podobným nasledovnému:

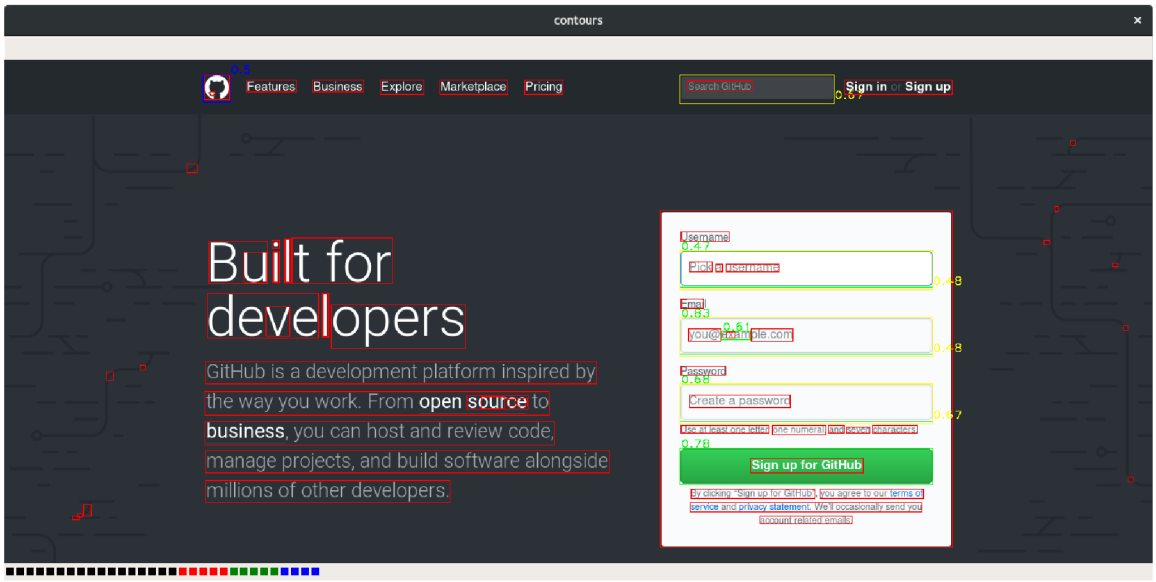
```
Getting contours with Laplacian detector.
Finding relations between objects
Running Hash property extractor |################################| 71/71
Running Relations property ex. |################################| 71/71
Getting text using Tesseract property extractor.
Running Tesseract property ex. |################################| 71/71
Running Button classifier
Finished! See results in wirec_output/
```

Po tom je možné nahliadnuť do adresára `results_dir/` (prednastavená implicitná cesta ku adresáru výstupu aplikácie), v ktorom sa nachádza celý výstup aplikácie no predovšetkým:

- `objects.json` - súbor s informáciami o všetkých detekovaných objektoch.
- `out.png` - snímok s vykreslenými objektami a tiež ich znázornenou klasifikáciou.

Je možné použiť parameter `--display_contours`, pomocou ktorého snímok s vykreslenými klasifikovanými objektami bude zobrazený hneď po ukončení klasifikácie, ktorý je nutné zatvoriť stiskom ktorejkoľvek klávesy.

```
(wirec)$ wirec --filepath img.png --display_contours
```




Obr. B.1: Zobrazený snímok klasifikovaných objektov aplikáciou.

### B.3.1 Príklad vygenerovaného JSON súboru

JSON súbor `objects.json` vygenerovaný v adresári `results_dir/` obsahuje zoznam JSON objektov. Príklad JSON výstupu je nasledovný:


```
[
        {
        "avg_bg_color": [
                0,
                0,
                0
                ],
        "box": false,
        "centered": 0.005826499784203687,
        "circle": false,
        "filepath": "results_dir/9697out.png",
        "hash": 0,
        "height": 1,
        "horizontal_edge": 1365.0,
        "id": 9697,
        "merged": false,
        "o_children": [],
        "o_parent": -1,
        "rectangular": true,
```

```
        "rel_size": 0.00010780752352770859,
        "role": {
               "button": 0.4,
               "checkbox": 0.5,
               "radio": 0,
               "textinput": 0.3
               },
        "size": 1365,
        "vertical_edge": 1.0,
        "vertically_next_dist": 76,
        "vertically_next_id": 9696,
        "vertically_next_related": false,
        "vertices": 2,
        "width": 1365,
        "x": 0,
        "y": 54
        },
        {...},
]
```

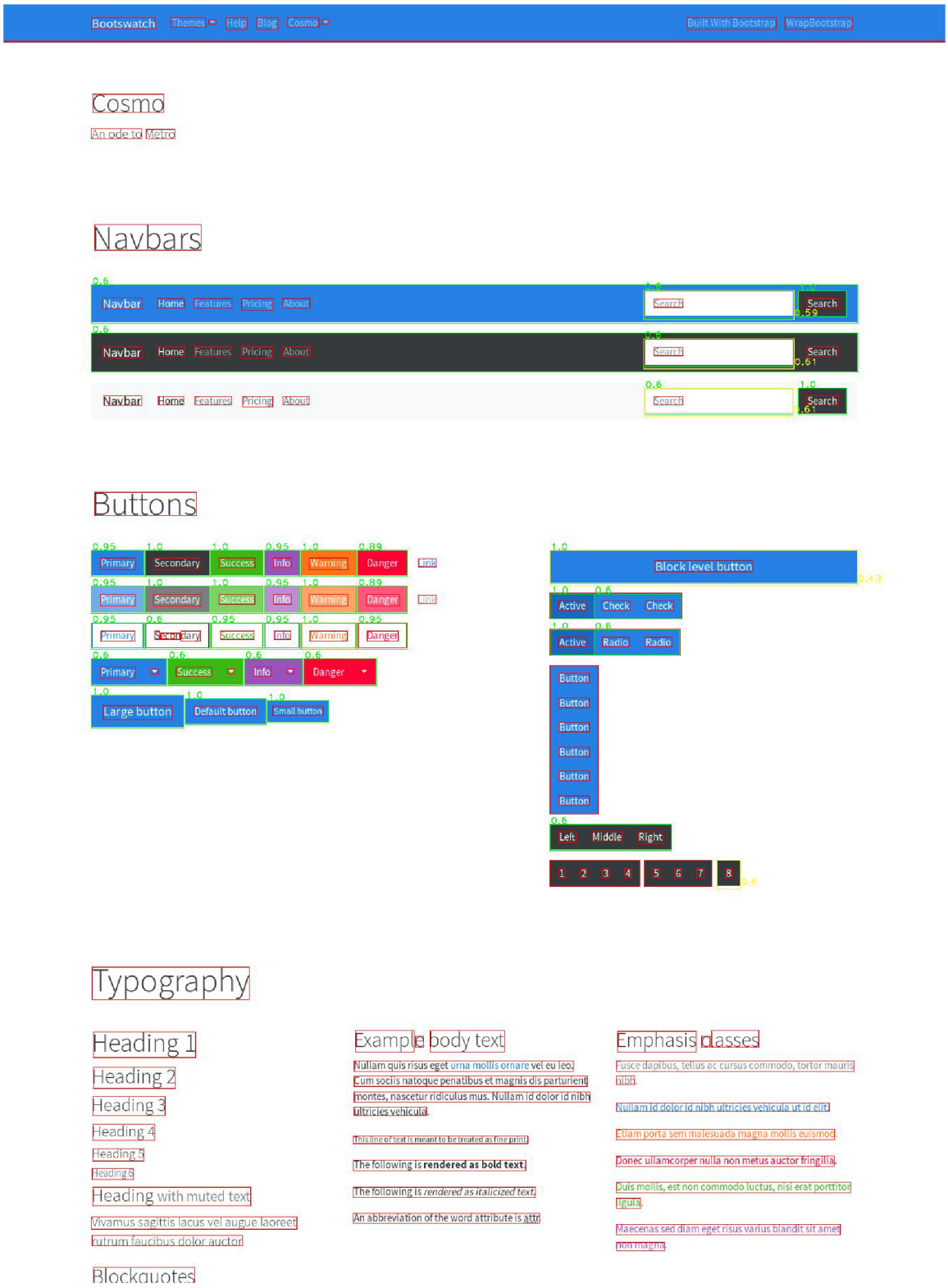
# Príloha C

# Bootstrap snímky

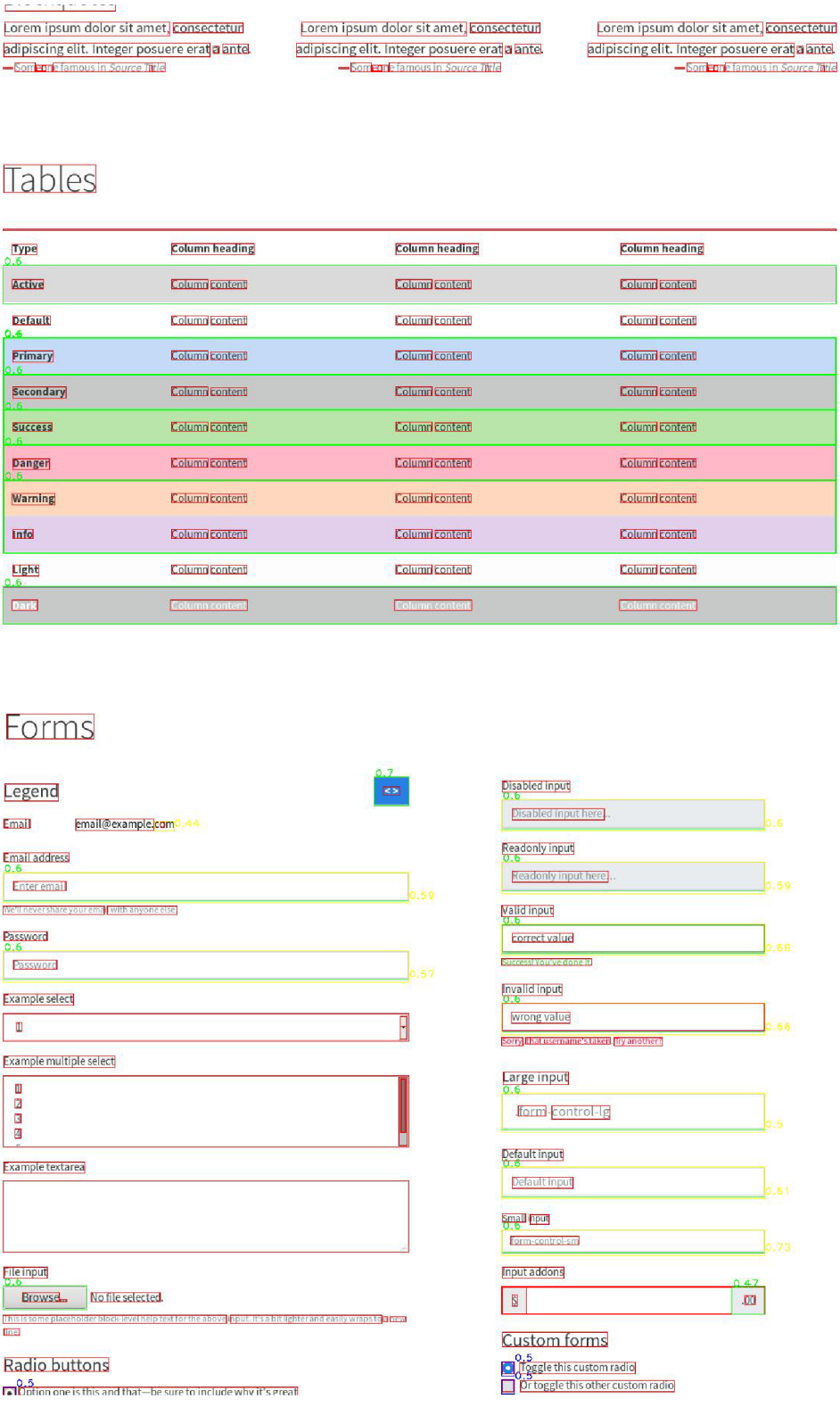
Na nasledujúcich snímkoch je zobrazený výstup aplikácie, ktorej vstupom bol obrázok Bootstrap šablóny. Pre lepšiu demonštráciu výstupu je pravdepodobnosť príslušnosti objektu ku triede objektom vpísaná ku jeho ohraničeniu. Každý objekt mohol byť vyhodnotený tak, že obsahuje viacero pravdepodobností ku rôznym triedam objektov zároveň. Tento prípad je zobrazený jemne prekrývajucími rámčekami.

| **Ohraničenie** | **Pozícia hodnoty pravdepodobnosti** | **Význam** |
|---|---|---|
| Zelená | Ľavý horný | Pravdepodobnosť objektu patriaceho ku triede objektov button. |
| Tmavo modrá | Pravý horný | Pravdepodobnosť objektu patriaceho ku triede objektov radio button. |
| Tysová | Ľavý dolný roh | Pravdepodobnosť objektu patriaceho ku triede objektov checkbox. |
| Žltá | Pravý dolný roh | Pravdepodobnosť objektu patriaceho ku triede objektov text input field. |
| Červená | | objekt klasifikovaný na menej ako 40% v každej z kategórií. |

Tabuľka C.1: Tabuľka významu použitých farieb.

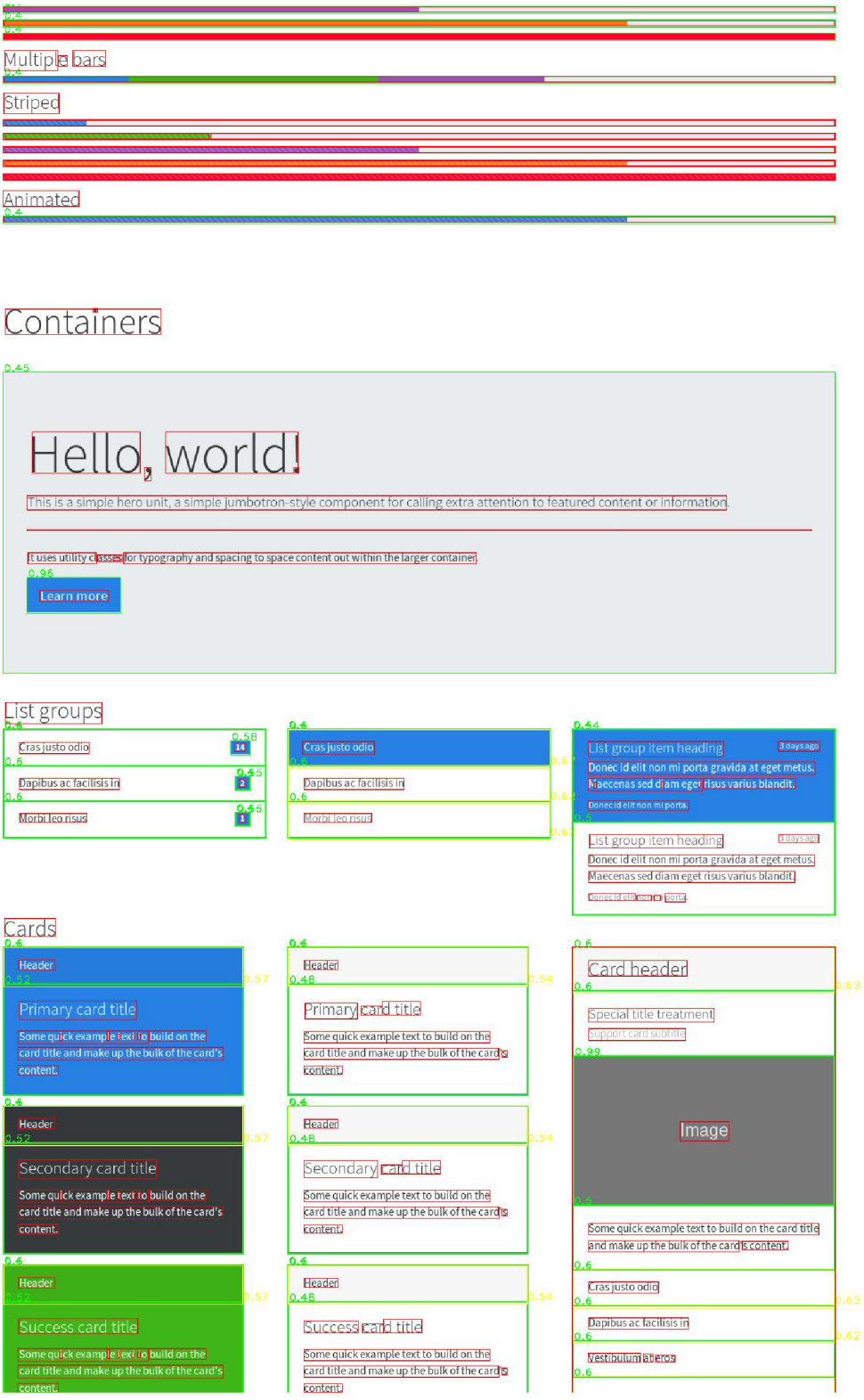



Obr. C.1: Snímok č. 1 GUI používajúceho Bootstrap štylovaný snímok.




Obr. C.2: Snímok č. 2 GUI používajúceho Bootstrap štylovaný snímok.


Obr. C.3: Snímok č. 3 GUI používajúceho Bootstrap štylovaný snímok.




Obr. C.4: Snímok č. 4 GUI používajúceho Bootstrap štylovaný snímok.


Obr. C.5: Snímok č. 5 GUI používajúceho Bootstrap štylovaný snímok.