Multiplatformska detekcija strojno čitljive zone na osobnim dokumentima

Bajtl, Saša

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:923212

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-07**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

SAŠA BAJTL

MULTIPLATFORMSKA DETEKCIJA STROJNO ČITLJIVE ZONE NA OSOBNIM DOKUMENTIMA

Diplomski rad

Pula, lipanj, 2022.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

SAŠA BAJTL

MULTIPLATFORMSKA DETEKCIJA STROJNO ČITLJIVE ZONE NA OSOBNIM DOKUMENTIMA

Diplomski rad

JMBAG: 0303038678, izvanredni student

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Nikola Tanković

Pula, lipanj, 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Saša Bajtl**, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Saša Bajtl

U Puli, 28. lipnja, 2022. godine

IZJAVA
o korištenju autorskog djela

Ja, **Saša Bajtl** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Multiplatformska detekcija strojno čitljive zone na osobnim dokumentima“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 28. lipnja, 2022. godine

Potpis

Saša Bajtl

Sadržaj

Sadržaj 1
1. Uvod....3
2. Strojno čitljivi putni dokumenti....4
2.1 Dokument Tip 1....4
2.2 Dokument Tip 2....6
2.3 Dokument Tip 3....7
2.4 Strojno čitljive vize....9
2.5 Kontrolna znamenka....11
2.6 Složene kontrolne znamenke....12
3. Računalni vid....13
3.1 Razlika između ljudskog vida i računalnog vida....13
3.2 OpenCV biblioteka....15
3.2.1 Moduli OpenCV biblioteke....15
4. Tehnologije korištene za implementaciju....17
4.1 WebAssembly....17
4.1.1 Asm.js....17
4.1.2 Minimalno održiv Wasm....19
4.1.3 Razlozi za korištenje WebAssembly....19
4.1.4 WebAssembly Modul....20
4.2 Emscripten....24
4.2.1 LLVM....24
4.2.2 Reprezentacija memorije....27
4.2.3 Alatni lanac....28
4.3 CMake....29
4.3.1 In source izgradnja....30
4.3.2 Out of source izgradnja....30
4.3.3 Generiranje projektnih datoteka....31
4.3.4 Pokretanje alata za izgradnju....33
5. Implementacija programskog rješenja....35
5.1 Struktura i okruženje projekta Document Extractor....35
5.2 Struktura i okruženje projekta Emscripten....44
5.3 Implementacija detekcije....51
5.4 Test projekt....58
5.5 Web-test dio projekta....60

6. Evaluacija radnih karakteristika....64
7. Zaključak....66
Literatura....68
Popis slika....70
Popis tablica....71
Sažetak....72
Abstract....73

1. Uvod

Multiplatformska softverska rješenja omogućuju prenosivost koda na različite platforme poput Windowsa i Linuxa, ali i mnoge druge. Postoji više razina prenosivosti: ponekad se za svaku od platformi zahtijeva posebna konfiguracija prilikom prevođenja, dok se u drugim slučajevima korištenja raznih interpretera programski kod može direktno izvršavati. Postoje i razni programski razvojni okviri (eng. framework) koji omogućuju razvoj za nekoliko platformi, poput Xamarin Forms-a ili Flutter-a. Općenito, takvi razvojni okviri za multiplatformski razvoj služe za razvoj mobilnih i desktop aplikacija s jednom instancom izvorišnog koda (eng. *code base*) s mogućnošću izvršavanja tog istog koda na iOS, Android i desktop Windows i MacOS platformama.

Ovaj rad opisuje razvoj aplikacije za detekciju strojno čitljive zone na putnim dokumentima, gdje se za razvoj koristi programski jezik C++ i biblioteka računalnog vida OpenCV. Aplikacija se razvila multiplatformski te se može odvijati na platformama Unix baze, poput Linuxa i MacOS-a, uz odgovarajuću konfiguraciju za izgradnju za svaku platformu. Za prevođenje izvorišnog koda koristi se sustav CMake. Dodatno, aplikacija će se prevesti pomoću okvira Emscripten koji služi za prevođenje C++ koda u WebAssembly (wasm). U tom obliku aplikacija se može izvršavati na bilo kojoj platformi koja ima web preglednik.

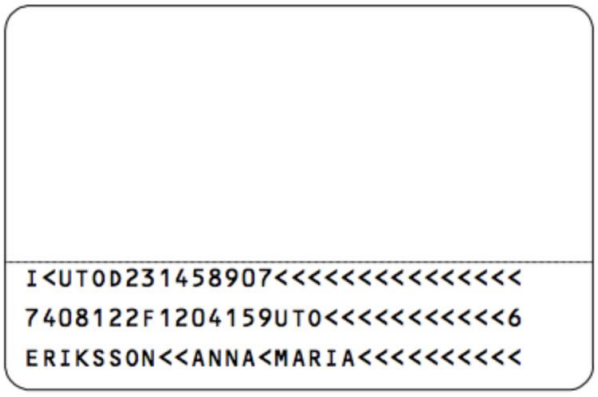
Rad se, pored uvoda, sastoji od šest poglavlja. U drugom poglavlju opisuju se tipovi osobnih dokumenata koji sadrže strojno čitljivu zonu, te postupak izračuna kontrolnih znamenki. Treće poglavlje opisuje osnovne pojmove računalnog vida, razliku između ljudskog i računalnog vida, te kratak opis o OpenCV biblioteke i njenih modula. Četvrto poglavlje govori o tehnologijama koje se koriste u implementaciji programskog rješenja, odnosno WebAssembly općenito, gdje se nastoji ukratko opisati povijest WebAssembly-a, razlozi njegovog korištenja, te kratak opis samog modula. Pored navedenog, poglavlje opisuje i osnovne karakteristike Emscripten razvojnog okvira, te sustav CMake. Peto poglavlje opisuje strukturu projekta i raspodjelu direktorija unutar projekta, te strukturu potprojekata, implementaciju detekcije strojno čitljive zone i postupak prevođenja i testiranja aplikacije. U šestom poglavlju su prikazani rezultati mjerenja brzine izvođenja na nekoliko platformi. Završnom poglavlje iznosi neke od prednosti i nedostataka multiplatformskog razvoja te iznosi završnu riječ.

2. Strojno čitljivi putni dokumenti

Dokument od strane Međunarodne organizacije civilnog zrakoplovstva (engl. *International Civil Aviation Organization*, skraćeno ICAO) 9303 opisuje tri glavna standardizirana tipa dokumenata (Tip 1, Tip 2, Tip 3) koji na stranici identiteta u donjem dijelu dokumenta sadrže strojno čitljivu zonu (engl. *Machine Readable Zone*, skraćeno MRZ), kodirana pomoću formata za optičko prepoznavanje znakova (engl. *Optical Character Recognition*, skraćeno OCR), kojeg su 1980. godine mnoge zemlje počele koristiti (Microblink, 2022) (vlastiti prijevod). Odobreni su od strane Međunarodne organizacije za standardizaciju (engl. *International Organisation for Standardization*, skraćeno ISO) i Međunarodne elektrotehničke komisije (engl. *International Electrotechnical Commisssion*, skraćeno IEC) kao ISO/IEC 7501-1. Za osobne iskaznice se koristi Tip 1, a za putovnice Tip 3. Tip 2 se uglavnom rijetko koristi (iDenfy, 2022) (vlastiti prijevod). Unutar strojno čitljive zone se nalaze informacije kao što su ime, broj dokumenta, državljanstvo, datum rođenja, spol i datum isteka dokumenta, uz dodatni prostor za opcionalne informacije kao na primjer informacije o zemlji i slično (Microblink, 2022) (vlastiti prijevod). Pored glavnih tipova putnih dokumenata postoji još i strojno čitljive vize (engl. *Machine Readable Visas*, skraćeno MRV), koje se sastoje od dva pod tipa MRVA i MRVB.

2.1 Dokument Tip 1

Strojno čitljiva zona putne isprave tipa 1 obuhvaća tri retka, a svaki redak ima 30 znakova. Slika 1 prikazuje stražnju stranu dokumenta tipa 1 na kojem se nalazi strojno čitljiva zona.



Slika 1: Stražnjeg dijela dokumenta sa strojno čitljivom zonom (Doubango, 2011)

U nastavku je u tablicama prikazano objašnjenje za znakove strojno čitljive zone za svaki redak i po pozicijama znakova dokumenta tipa 1.

Tablica 1: Prvi redak dokument tip 1 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 2	A, C ili I kao prvi znak označava da je dokument službena putna isprava. Još jedan dodatni znak može se koristiti za daljnju identifikaciju dokumenta, ali to ne može biti V ili C.
3 do 5	Slijedeća tri slova označavaju državu izdavanja dokumenta.
6 do 14	Slijedeći znakovi predstavljaju broj dokumenta koji sadrži do 9 alfanumeričkih znakova.
15	Dalje u nizu je kontrolna znamenka broja dokumenta.
16 do 30	Neobavezni podaci prema nahođenju države izdavatelja. Može sadržavati prošireni broj dokumenta.

Tablica 2: Drugi redak dokument tip 1 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 6	Datum rođenja vlasnika u formatu YYMMDD.
7	Kontrolna znamenka za datum rođenja vlasnika.
8	Oznaka spola vlasnika dokumenta, muški ili ženski (engl. Male / Female – M / F).
9 do 14	Datum valjanosti dokumenta u formatu YYMMDD.
15	Kontrolna znamenka za datum valjanosti.
16 do 18	Nacionalnost vlasnika dokumenta prikazana sa tri znaka.
19 do 29	Neobavezni podaci prema nahođenju države izdavanja.
30	Sve ukupna kontrolna znamenka za gornje i srednje strojno čitljive redove.

Tablica 3: Treći redak dokument tip 1 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 8	Primarni identifikator (prezime). Ako postoji više od jedne komponente, one se odvajaju jednim znakom za popunjavanje (<).
9 do 10	Dvostruki znakovi za popunjavanje označavaju kraj primarnog identifikatora.
11 do 20	Sekundarni identifikator (ime). Svaka komponenta je odvojena jednim znakom za popunjavanje.

21 do 30	Znakovi za popunjavanje koji se koriste za dovršavanje donjeg strojno čitljivog retka. Označava da nisu uključene druge komponente imena.

2.2 Dokument Tip 2

Za razliku od dokumenta tipa 1, ovaj dokument se sastoji od dva retka od kojih svaki ima 36 znakova, te se nešto rjeđe koristi nego dokument tipa 1. Slika 2 prikazuje primjer dokumenta tipa 2.

Slika 2: Primjer dokumenta tipa 2 sa strojno čitljivom zonom (Doubango, 2011)

U nastavku je u tablicama prikazano objašnjenje za znakove strojno čitljive zone za svaki redak i po pozicijama znakova dokumenta tipa 2.

Tablica 4: Prvi redak dokument tip 2 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 2	Kao i kod dokumenta tipa 1, A, C ili I kao prvi znak označava da je dokument službena putna isprava, gdje se dodatni znak može se koristiti za daljnju identifikaciju dokumenta i ne može biti V ili C.
3 do 5	Slijedeća tri slova označavaju državu izdavanja dokumenta.
6 do 13	Primarni identifikator (prezime). Ako postoji više od jedne komponente, one se odvajaju jednim znakom za popunjavanje (<).
14 do 15	Dvostruki znakovi << označavaju kraj primarnog identifikatora.
16 do 25	Sekundarni identifikator (ime). Svaka komponenta je odvojena jednim znakom za popunjavanje.

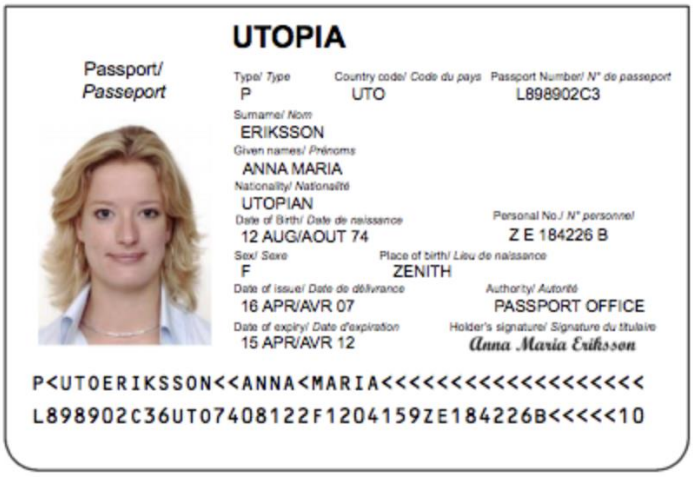
26 do 36	Znakovi koji služe za popunjavanje gornje strojno čitljive linije i ukazuju da nema uključenih drugih komponenata imena.

Tablica 5: Drugi redak dokument tip 2 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 9	Znakovi predstavljaju broj dokumenta koji sadrži do 9 alfanumeričkih znakova.
10	Kontrolna znamenka za broj dokumenta.
11 do 13	Nacionalnost vlasnika dokumenta prikazana sa tri znaka.
14 do 19	Datum rođenja vlasnika u formatu YYMMDD.
20	Kontrolna znamenka za datum rođenja.
21	Oznaka spola vlasnika dokumenta.
22 do 27	Datum valjanosti dokumenta u formatu YYMMDD.
28	Kontrolna znamenka datuma valjanosti.
29 do 35	Neobavezni podaci prema nahođenju države izdavatelja. Može sadržavati prošireni broj dokumenta.
36	Sve ukupna kontrolna znamenka za donji strojno čitljivi red.

2.3 Dokument Tip 3

Strojno čitljiva putovnica (engl. *Machine Readable Passport*, skraćeno MRP) je strojno čitljivi putni dokument tipa 3 (engl. *Machine Readable Travel Document*, skraćeno MRTD), kod kojih se strojno čitljiva zona sastoji od dva retka, gdje svaki redak ima 44 znaka. Strojno čitljive putovnice omogućuju imigracijskim službama bržu obradu putnika, veću preciznost i veću brzinu kod unosa podataka, te usklađivanje podataka s imigracijskim bazama podataka. Slika 3 prikazuje primjer putovnice koja pored ostalih podataka na dnu identifikacijske stranice sadrži strojno čitljivu zonu.



Slika 3: Primjer putovnice sa strojno čitljivom zonom (Doubango, 2011)

U nastavku je u tablicama prikazano objašnjenje za znakove strojno čitljive zone za svaki redak i po pozicijama znakova dokumenta tipa .

Tablica 6: Prvi redak dokument tip 3 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 2	Znak P ukazuje da je dokument putovnica. Još jedan dodatni znak se može koristiti za daljnju identifikaciju dokumenta prema nahođenju države izdavanja.
3 do 5	Slijedeća tri slova označavaju državu izdavanja dokumenta.
6 do 13	Primarni identifikator (prezime). Ako postoji više od jedne komponente, one se odvajaju jednim znakom za popunjavanje (<).
14 do 15	Dvostruki znakovi << označavaju kraj primarnog identifikatora.
16 do 25	Sekundarni identifikator (ime). Svaka komponenta je odvojena jednim znakom za popunjavanje.
26 do 44	Znakovi koji služe za popunjavanje gornje strojno čitljive linije i ukazuju da nema uključenih drugih komponenata imena.

Tablica 7: Drugi redak dokument tip 3 (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 9	Znakovi predstavljaju broj dokumenta koji sadrži do 9 alfanumeričkih znakova. Na nekorištenim pozicijama u broju dokumenta nalazi se znak za popunjavanje.
10	Kontrolna znamenka broja dokumenta.
11 do 13	Nacionalnost vlasnika dokumenta prikazana sa tri znaka.
14 do 19	Datum rođenja vlasnika u formatu YYMMDD.
20	Kontrolna znamenka za datum rođenja.
21	Oznaka spola vlasnika dokumenta.
22 do 27	Datum valjanosti dokumenta u formatu YYMMDD.
28	Kontrolna znamenka datuma valjanosti.
29 do 42	Neobavezni podaci prema nahođenju države izdavatelja.
43	Kontrolna znamenka za opcionalne podatke.
44	Sve ukupna kontrolna znamenka donjeg strojno čitljivog retka.

2.4 Strojno čitljive vize

Pored nabrojanih strojno čitljivih dokumenata, postoji još jedan tip dokumenta koji ima strojno čitljivu zonu, a to je viza i sastoji se od dva pod tipa, MRVA i MRVB. Na slici 4 je prikazan primjer strojno čitljive vize.

Slika 4: Primjer vize sa strojno čitljivom zonom (Doubango, 2011)

Tablica 8: Prvi redak strojno čitljive vize (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 2	Znak V ukazuje da je dokument viza. Još jedan dodatni znak se može koristiti za daljnju identifikaciju dokumenta prema nahođenju države izdavanja.
3 do 5	Slijedeća tri slova označavaju državu izdavanja dokumenta.
6 do 13	Primarni identifikator (prezime). Ako postoji više od jedne komponente, one se odvajaju jednim znakom za popunjavanje (<).
13 do 15	Dvostruki znakovi << označavaju kraj primarnog identifikatora.
16 do 25	Sekundarni identifikator (ime). Svaka komponenta je odvojena jednim znakom za popunjavanje.
26 do 44	Znakovi koji služe za popunjavanje gornje strojno čitljive linije i ukazuju da nema uključenih drugih komponenata imena.

Tablica 9: Drugi redak strojno čitljive vize (Doubango, 2011) (vlastiti prijevod)

Pozicija	Opis
1 do 9	Znakovi predstavljaju broj dokumenta koji sadrži do 9 alfanumeričkih znakova. Na nekorištenim pozicijama u broju dokumenta nalazi se znak za popunjavanje.
10	Kontrolna znamenka broja dokumenta.
11 do 13	Nacionalnost vlasnika dokumenta prikazana sa tri znaka.
14 do 19	Datum rođenja vlasnika u formatu YYMMDD.
20	Kontrolna znamenka za datum rođenja.
21	Oznaka spola vlasnika dokumenta.
22 do 27	Datum valjanosti dokumenta u formatu YYMMDD.
28	Kontrolna znamenka datuma valjanosti.
29 do 44	Neobavezni podaci prema nahođenju države izdavatelja.

MRVA i MRVB dva dokumenta sa praktički sličnom strojno čitljivom zonom koji se razlikuju u broju znakova i veličini dokumenta, gdje se MRVA sastoji od 2 retka od po 44 znaka, dok MRVB se sastoji od 2 retka od po 36 znakova.

2.5 Kontrolna znamenka

Kontrolna znamenka je jedna znamenka koja se računa iz ostalih znamenki u nizu. Dakle izračunavaju se na određenim brojčanim elementima podataka u strojno čitljivoj zoni i omogućuju čitaču da provjere ispravnost interpretacije podataka. Usvojen je poseban izračun kontrolnih znamenki za upotrebu u strojno čitljivim putnim dokumentima.

Kontrolne znamenke izračunavaju se s modulom 10 uz kontinuirano ponavljajuće znamenaka od 731 731 itd., te se računa u nekoliko koraka (Microblink, 2022) (vlastiti prijevod):

1. Idući s lijeva na desno, pomnoži se svaka znamenku odgovarajućeg numeričkog elementa podataka s težinskim brojem koji se pojavljuje u odgovarajućem sekvencijalnom položaju.
2. Zbroji se proizvod svakog umnoška.
3. Podjeli se suma s 10 (modul).
4. Ostatak je kontrolna znamenka.

Za elemente podataka u kojima broj ne zauzima sve raspoložive znakovne pozicije, simbol < koristi se za popunjavanje slobodnih mjesta i dobiva vrijednost nula u svrhu izračunavanja kontrolne znamenke (Microblink, 2022) (vlastiti prijevod). Kada se izračun kontrolne znamenke primjenjuje na elemente podataka koji sadrže abecedne znakove, znakovi od A do Z imaju vrijednosti od 10 do 35 uzastopno (Microblink, 2022) (vlastiti prijevod). Slika 5 prikazuje raspored znakova vrijednosti.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

Slika 5: Raspored abecednih znakova od A do Z u odnosu na vrijednosti (Microblink, 2022)

Primjer izračuna kontrolne znamenke:

- Za primjer uzmemo datum 18.07.1984.

Datum	**8**	**4**	**0**	**7**	**1**	**8**
Težina	7	3	1	7	3	1

Produkt	56	12	0	49	3	8

Tablica 10: Primjer izračuna kontrolne znamenke

- Pomnožimo vrijednosti datuma i težine i rezultat je produkt. Dakle **7 x 8 = 56, 3 x 4 = 12, 1 x 0 = 0, 7 x 7 = 49, 3 x 1 = 3, 1 x 8 = 8**.
- Suma produkta: **56 +12 + 0 + 49 + 3 + 8 = 128**.
- Dijeljenje sa brojem 10: **128 / 10 = 12,8**.
- Kontrolna znamenka je ostatak, dakle to je broj **8**.

2.6 Složene kontrolne znamenke

Kod dokumenta tipa 1, za računanje kontrolne znamenke koristi se prvi red strojno čitljive zone s pozicijom znakova od 1 do 30, i srednji red strojno čitljive zone s pozicijom znakova od 1 do 29. Kod dokumenta tipa 2, za računanje kontrolne znamenke koristi se donji red strojno čitljive zone s pozicijom znakova od 1 do 35, dok kod dokumenta tipa 3 se koristi isto donji red strojno čitljive zone sa pozicijom znakova od 1 do 43. U nastavku je prikazano izračunavanje sveukupne kontrolne znamenke za dokument tipa 1.

I<YTO**D231458907<<<<<<<<<<<<<<** (prvi red strojno čitljive zone)

3407127M**9507122**YTO**<<<<<<<<<<<** (srednji red strojno čitljive zone)

Tablica 11: Primjer izračuna složene kontrolne znamenke

Element	**D**	**2**	**3**	**1**	**4**	**5**	**8**	**9**	**0**	**7**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**
Vrijednost	13	2	3	1	4	5	8	9	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0
Težina	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7
Produkt	91	6	3	7	12	5	56	27	0	49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tablica 12: Primjer izračuna složene kontrolne znamenke (nastavak od prethodne)

Element	**3**	**4**	**0**	**7**	**1**	**2**	**7**	**9**	**5**	**0**	**7**	**1**	**2**	**2**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**	**<**
Vrijednost	3	4	0	7	1	2	7	9	5	0	7	1	2	2	0	0	0	0	0	0	0	0	0	0	0
Težina	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3	1	7	3
Produkt	9	4	0	21	1	14	21	9	35	0	7	7	6	2	0	0	0	0	0	0	0	0	0	0	0

Izračun kontrolne znamenke:

91 + 6 + 3 + 7 + 12 + 5 + 56 + 27 + 0 + 49 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 9 + 4 + 0 + 21 + 1 + 14 + 21 + 9 + 35 + 0 + 7 + 7 + 6 + 2 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 392; 392 / 10 = 39,2; Kontrolna znamenka srednjeg reda je broj **2**.

3. Računalni vid

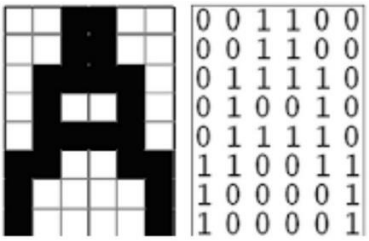
Računalni vid (eng. *Computer Vision*) je interdisciplinarno znanstveno područje koje se bavi time kako računala mogu steći visoko razumijevanje iz digitalnih slika ili videa. Iz perspektive inženjerstva, nastoji razumjeti i automatizirati zadatke koje ljudski vizualni sustav može obaviti (Wikipedia, 2022a) (vlastiti prijevod). Zadaci računalnog vida uključuju metode za prikupljanje, obradu, analizu i razumijevanje digitalnih slika te izdvajanje visoko dimenzionalnih podataka iz stvarnog svijeta kako bi se proizvele numeričke ili simboličke informacije (Wikipedia, 2022a) (vlastiti prijevod).

3.1 Razlika između ljudskog vida i računalnog vida

Ljudski vid se bazira na svjetlost i ne uključuje ponavljanje ili nekakve obrasce izračunavanja. Drugim riječima, ne trebamo učiti da bi vidjeli, jer je vid biološki ugrađen u nas. Ljudski vid se sastoji od nekoliko koraka. Prvo se svjetlost odbija od slike i ulazi u oči kroz rožnicu. Zatim rožnica usmjerava svjetlost na zjenice i šarenicu, koji zajedno kontroliraju količinu svjetlosti koja ulazi u oko. Nakon što svjetlost prođe kroz rožnicu, ona ulazi u mrežnicu koja ima posebne senzore zvane čunjevi i štapići, koji su uključeni u vid boja (Leo Gamboa Uribe, 2020) (vlastiti prijevod).
Dakle, na prvom mjestu, zadatak vida je obrada informacija jer da bi razumio da postoji slika, naš mozak mora biti u stanju interpretirati te informacije kao boju, oblik, pokret, detalje i ljepotu (Leo Gamboa Uribe, 2020) (vlastiti prijevod). Slika 6 prikazuje način na koji se interpretira ljudski vid.

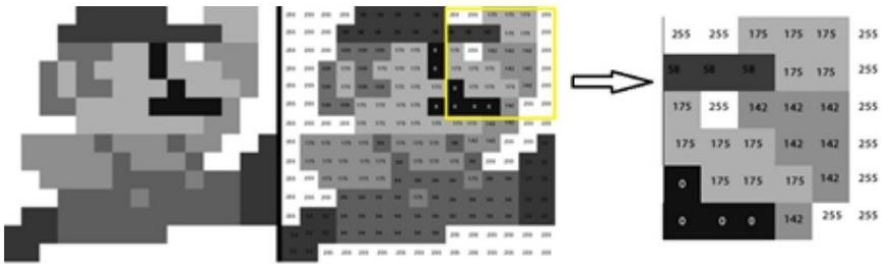
Slika 6: Ljudski vid (Leo Gamboa Uribe, 2020)

Za razliku od ljudskog vida, računalni vid interpretira sliku kao digitalnu u obliku matrice koja se sastoji od konačnog broja elemenata poredanih u matrici na Kartezijskoj ravnini sa x i y koordinatama. Koordinate x, y predstavljaju točku ili piksel koji prikazuje boju te slike, odnosno boju piksela (Leo Gamboa Uribe, 2020) (vlastiti prijevod). Reprezentacija boje slike u pikselu može biti binarna gdje 1 znači crna boja a 0 znači bijela u slučaju da je slika crno bijela, kao na slici 7 ispod.



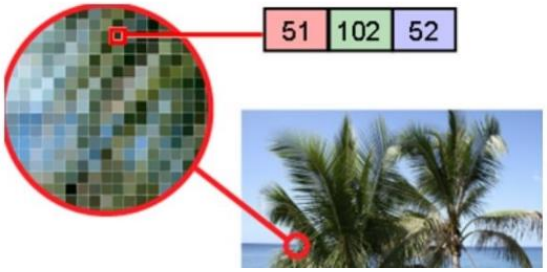
Slika 7: Binarni prikaz boje slike (Leo Gamboa Uribe, 2020)

Reprezentacija slike može biti s 256 razina intenziteta gdje se svaki piksel predstavlja kao cijeli broj i tumači se kao razina intenziteta u rasponu od 0 do 256, kao što je prikazano na slici 8 (Leo Gamboa Uribe, 2020) (vlastiti prijevod).



Slika 8: Reprezentacija u 256 razina intenziteta (Leo Gamboa Uribe, 2020)

U slučaju slike u boji, svaki piksel je prikazan s tri vrijednosti koje predstavljaju kombinaciju boja crvene, zelene i plave (eng. *red green blue*, skraćeno RGB), gdje će rezultirajuća boja piksela biti definirana količinom intenziteta koju svaka komponenta ima kako je prikazano na slici 9 (Leo Gamboa Uribe, 2020) (vlastiti prijevod).



Slika 9: Reprezentacija u boji (Leo Gamboa Uribe, 2020)

3.2 OpenCV biblioteka

OpenCV je biblioteka otvorenog koda (engl. *open source*) napisana u optimiziranom C++ jeziku i služi za razvoj aplikacija računalnog vida na gotovo svim platformama, kako na Windows, Linux i MacOS, tako i na mobilnim i web platformama, te se može koristiti u akademske ili komercijalne svrhe pod BSD[1] licencom koja to dozvoljava (Escriva, Laganiere and Safari, 2019) (vlastiti prijevod). Ima mogućnost izvršavanja algoritama računalnog vida u realnom vremenu i omogućava izvršavanje programa na više nitnim (engl. *multi thread*) sustavima, te sadrži nekoliko stotina algoritama. OpenCV je modularne strukture, što znači da sadrži nekoliko dijeljenih ili statičkih biblioteka, od kojih svaki modul pripada jednoj grupi problema računalnog vida (OpenCV, 2022) (vlastiti prijevod).

3.2.1 Moduli OpenCV biblioteke

U nastavku su opisani glavni moduli u OpenCV biblioteci. Osim glavnih modula, postoje još dodatni moduli koji su potpuno odvojeni od glavnog modula. U nastavku su navedeni i u kratko opisani glavni moduli:

- **core** – Osnovna funkcionalnost, modul koji sadrži osnovne strukture podataka, uključujući višedimenzionalno polje i osnovne funkcije koje koriste svi ostali moduli (OpenCV, 2022) (vlastiti prijevod).
- **imgproc** – Obrada slike, modul za obradu slike koji uključuje linearno i nelinearno filtriranje slike, geometrijske transformacije slike (promjenu veličine, afino i perspektivno iskrivljenje, generičko preslikavanje temeljeno na tablici), pretvorbu prostora boja, histograme i tako dalje (OpenCV, 2022) (vlastiti prijevod).
- **imgcodecs** – Učitavanje i spremanje slikovnih datoteka, modul za video analizu koji uključuje procjenu pokreta, oduzimanje pozadine i algoritme za praćenje objekata (OpenCV, 2022) (vlastiti prijevod).
- **videoio** – Video I/O, jednostavno sučelje za snimanje videa i video kodeci (OpenCV, 2022) (vlastiti prijevod).

[1] BSD licenca – https://en.wikipedia.org/wiki/BSD_licenses

- **highgui** – Grafičko korisničko sučelje, jednostavna izrada i manipulacija prozorima koji mogu prikazivati slike, te dodavanje alatne trake koja može upravljati akcijama miša i tipkovnice (Fernández Villán, 2019) (vlastiti prijevod).
- **video** – Modul za video analizu, uključujući pozadinsko oduzimanje, procjenu gibanja i algoritme za praćenje objekata (Fernández Villán, 2019) (vlastiti prijevod).
- **calib3d** – Kalibracija kamere i 3D rekonstrukcija, osnovni algoritmi geometrije višestrukih prikaza, kalibracija jedne i stereo kamere, procjena položaja objekta, algoritmi stereo korespondencije i elementi 3D rekonstrukcije (OpenCV, 2022) (vlastiti prijevod).
- **features2d** – Programski okvir 2D značajki, uključuje značajku detektora, deskriptora i uparivača deskriptora.
- **objdetect** – Detekcija objekata i instanci unaprijed definiranih klasa (na primjer, lica, oči, šalice, ljudi, automobili i tako dalje).
- **dnn** – Modul duboke neuronske mreže, sadrži API za kreiranje novih slojeva skup izgrađenih korisnih slojeva, API za konstruiranje i modificiranje neuronskih mreža iz slojeva, funkcionalnost za učitavanje serijaliziranih modela mreža iz različitih okvira dubokog učenja (Fernández Villán, 2019) (vlastiti prijevod).
- **ml** – Strojno učenje, biblioteka strojnog učenja (eng. *Machine Learning Library*, skraćeno MLL) skup je klasa i metoda koje se mogu koristiti u svrhu klasifikacije, regresije i grupiranja (Fernández Villán, 2019) (vlastiti prijevod).
- **flann** – (eng. *Fast Library Approximate Nearest Neighbours*, skraćeno FLANN) klasteriranje i pretraga multidimenzionalnog prostora, zbirka algoritama koji su vrlo prikladni za brza pretraživanja najbližih susjeda (Fernández Villán, 2019) (vlastiti prijevod).
- **photo** – Računalna fotografija, sadrži neke funkcije za računalnu fotografiju.
- **stitching** – Ovaj modul implementira automatsko spajanje panoramskih slika.
- **gapi** – Graph API (eng. *Graph Aplication Programming Interface*, skraćeno G-API), novi OpenCV modul usmjeren na brzu i prenosivu regularnu obradu slika. Ova dva cilja postižu se uvođenjem novog modela izvedbe koji se temelji na grafovima.

4. Tehnologije korištene za implementaciju

S napretkom tehnologija, došlo je do sve većeg premještanja aplikacija na web, što je uzrokovalo jedan dodatni izazov, jer web podržava samo jedan programski jezik, JavaScript. To je ujedno i prednost JavaScripta jer se kod napiše jednom i može se izvršavati na svim web preglednicima, što ne znači da će raditi 100% ispravno jer svaki preglednik možda ne podržava neke značajke koje podržava neki drugi, pa se svejedno ta aplikacija mora testirati na raznim preglednicima.

Jedna od najvažnijih stvari, kada se radi o izradi aplikacije na web je sama izvedba web stranice, što znači koliko se brzo učitava ili kolika joj je responzivnost (Gallant, 2019) (vlastiti prijevod). Smatra se da ako se stranica ne učita u roku 3 sekunde otkad je posjetitelj posjeti, 40% posto posjetitelja će napustiti stranicu prije nego se učita i taj postotak se povećava za svaku sekundu duže. 79% posjetitelja tvrde da je manja vjerojatnost da će kupovati s ovakve web lokacije (An and Meenan, 2016).

Nedostatak web preglednika je taj da ako se želi premjestiti aplikacija na web koja nije pisana u JavaScriptu, već je napisana u jeziku poput C++. JavaScript je izvrstan jezik, ali je sve veća potreba da radi jako brzo i da obavlja zahtjevne izračune u industriji igara, gdje C++ dolazi do izražaja (Gallant, 2019) (vlastiti prijevod).

4.1 WebAssembly

WebAssembly (skraćeno Wasm) je binarni format instrukcija za virtualni stroj baziran na stogu. Wasm je dizajniran kao portabilni kompilacijski cilj (eng. *compilation target*) za programske jezike, omogućavajući raspoređivanje (eng. *deployment*) na web za klijentske i poslužiteljske aplikacije (WebAssembly, 2022) (vlastiti prijevod).

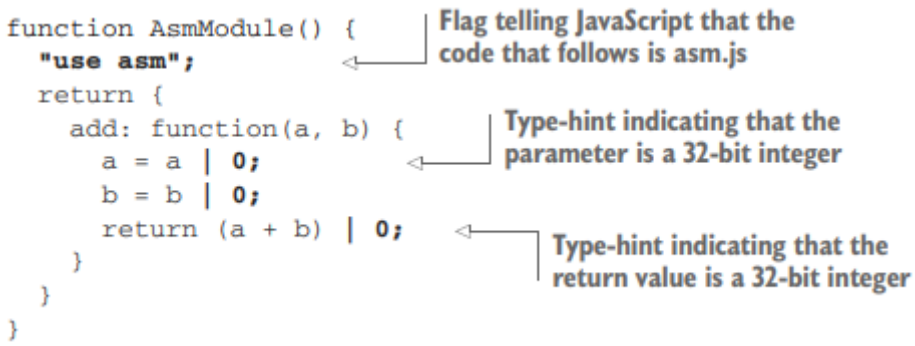
Kako su proizvođači preglednika tražili načine za poboljšanje performansi JavaScripta, Mozilla (koja čini preglednik Firefox) definirala je podskup JavaScripta nazvan asm.js (Gallant, 2019) (vlastiti prijevod).

4.1.1 Asm.js

Asm.js je vrlo mali striktni podskup JavaScripta, koji dopušta konstrukcije kao što su while, if, brojevi, funkcije najviše razine i mnoge druge jednostavne konstrukcije, nastao prije WebAssembly (Mozilla, 2005) (vlastiti prijevod). Dakle ne dopušta ništa što alocira hrpu, kao na primjer, objekti, nizovi i slično. Vrlo je optimiziran i brz u izvođenju jer koristi moderne tehnike prevođenja Just-In-Time (JIT), te ideja

izvršavanja je da bude veoma brzo, te u koliko to nije, znači da se radi o grešci u programu (eng. *bug*) (Mozilla, 2005) (vlastiti prijevod). Zastario je i ne koristi se više.
Prednosti asm.js:

- Kod se ne piše direktno u JavaScriptu, već se sva logika piše u C ili C++, gdje se kod transpilira (eng. *transpilling*), odnosno pretvara u JavaScript, obično pomoću programskog okvira kao Emscripten o kojem će se kasnije govoriti u ovom radu (Gallant, 2019).
- Veoma velika brzina izvršavanja koda kod zahtjevnih izračuna, gdje preglednik uz posebnu naredbu „use asm;“, dobije instrukcije da koristi operacije sustava niske razine, umjesto skupih JavaScript operacija (Gallant, 2019) (vlastiti prijevod).
- Brzo izvršavanje već kod samo prvog poziva, jer sadrži instrukcije tipova (eng. *type hint*) u kodu gdje unaprijed govori JavaScriptu kakvog će varijabla biti tipa, na primjer a | 0. To govori da će varijabla a koristiti 32 bitnu vrijednost cijelog broja, što znači da JavaScript engine ne mora pratiti kod kako bi saznao kojeg su tipa i može jednostavno izvršiti kako su deklarirane (Gallant, 2019) (vlastiti prijevod). Slika 10 prikazuje primjer jednostavne funkcije u asm.js modulu.



Slika 10: Asm modul (Gallant, 2019)

Nedostatci asm.js:

- Instrukcije tipova čine veoma veliku datoteku (Gallant, 2019).
- Datoteka asm.js je JavaScript datoteka, tako da je još uvijek mora pročitati i analizirati JavaScript engine. To postaje problem na uređajima poput telefona jer sva obrada usporava vrijeme učitavanja i troši bateriju (Gallant, 2019).
- Da bi dodali dodatne značajke, proizvođači preglednika morali bi izmijeniti sam JavaScript jezik, što nije poželjno (Gallant, 2019).
- JavaScript je programski jezik i nije mu za cilj biti prevoditelj (Gallant, 2019).

4.1.2 Minimalno održiv Wasm

Nakon asm.js proizvođači web preglednika razmišljaju o unapređenju asm.js, pritom uzimajući u obzir pozitivne aspekte asm.j, dolaze do WebAssembly prvog minimalno održivog proizvoda (eng. *Minimum Viable Product*, skraćeno MVP), uzimajući za cilj unapređenje nedostataka asm.js, nakon čega 4 velika proizvođača web preglednika Google, Microsoft, Mozzila i Apple ažuriraju svoje preglednike u svrhu podrške za Wasm (Gallant, 2019). Nekoliko značajki koje karakteriziraju Wasm:

- WebAssembly je jezik niske razine koji ima mogućnost brzine izvođenja gotovo blizu izvornoj (eng. *native*) brzini, koji se može izvršavati na svim modernim web i mobilnim preglednicima (Gallant, 2019) (vlastiti prevod).
- Dizajniran je na način kao cilj prevođenja (eng. *compile target*) što omogućava jezicima poput C++, Rust i drugi, izvođenje u web preglednicima.
- WebAssembly datoteke imaju kompaktan dizajn, što znači da im je prijenos i preuzimanje veoma brzo, kao i raščlanjivanje i inicijalizacija (Gallant, 2019).

4.1.3 Razlozi za korištenje WebAssembly

Bolje performanse – Kod JavaScripta se od razvojnih inženjera zahtjeva da donesu odluke na koji način će se dizajnirati JavaScript engine. Kao na primjer, moguće je optimizirati JavaScript pomoću JIT prevoditelja, kojem će brzina izvršavanja biti u samom vrhu, ali zato zahtjeva više vremena za pokretanje. Alternativno se može koristiti interpreter koji odmah počne izvršavati kod, ali neće dostići sam vrh performansi prevoditelj koji optimizira JIT. Većina inženjera koristi implementaciju na oba načina, ali to zahtjeva veću alokaciju memorije. WebAssembly osigurava brže pokretanje i dostiže sam vrh performansi bez dodatnog opterećenja memorije. Međutim, nemoguće je na lak način prepisati JavaScript kod u C++, Rust i slično, bez ulaganja dodatnog napora, što kod prepisa može rezultirati lošim performansama, zbog ne razumijevanja WebAssemblyja, koje se mogu neznatno povećati kod postavljanja zastavica optimizacije (Battagline, 2021).

Integracija naslijeđenih (eng. *legacy*) biblioteka – Dvije popularne biblioteke za prijenos postojećih biblioteka na WebAssembly su wasm-pack za Rust i Emscripten za C/C++ koji se i koristi u ovom radu (Battagline, 2021) (vlastiti prijevod). Korištenje WebAssembly idealno je kada postoji kod napisan na C/C++ ili Rust koji je potrebno koristiti u web aplikacijama ili prebaciti cijele postojeće desktop aplikacije. Emscripten

alatni lanac posebno je učinkovit u prijenosu postojećih C++ desktop aplikacija na web koristeći WebAssembly (Battagline, 2021).

Portabilnost i sigurnost – WebAssembly je započeo kao tehnologija za pokretanje u pregledniku, ali se brzo proširuje i postaje zaštićeno okruženje za pokretanje bilo gdje. WebAssembly za ugrađene sustave i internet stvari (eng *Internet of Things*, skraćeno IoT), radna grupa WebAssemblyja stvara visoko sigurno vrijeme izvođenja koje sprječava zlonamjerne aktere da kompromitiraju kod (Battagline, 2021) (vlastiti prijevod). Iako se radna skupina WebAssembly usredotočuje na sigurnost, nijedan sustav nije potpuno siguran.

4.1.4 WebAssembly Modul

Glavni dio WebAssembly programa, kako za binarnu verziju koda, tako i za prevedenu verziju koda, naziva se modul (Gallant, 2019) (vlastiti prijevod). WebAssembly trenutno podržava samo četiri tipa vrijednosti podataka:

1. 32 – bitni cijeli brojevi,
2. 64 – bitni cijeli brojevi,
3. 32 – bitni float,
4. 64 – bitni float.

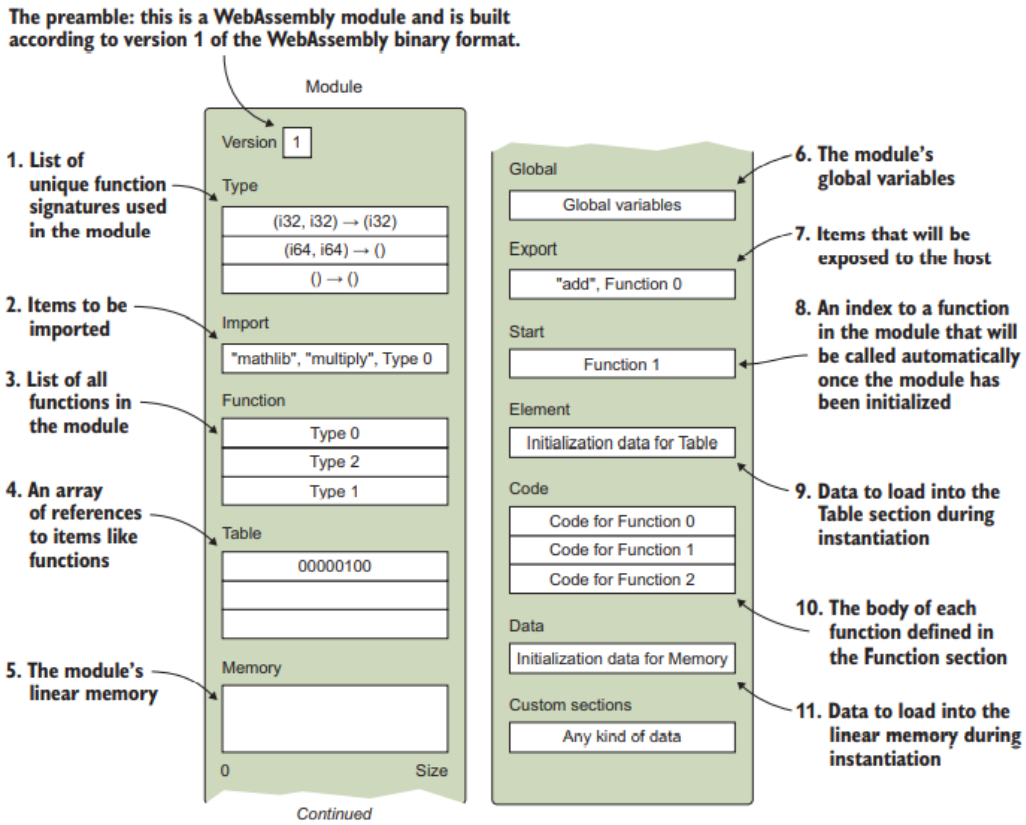
Bool vrijednost se reprezentiraju pomoću 32 – bitnog cijelog broja, gdje je 0 laž (eng. *false*) a brojčana vrijednost različita od 0 predstavlja istinitost (eng. *true*). Sve druge vrijednosti poput stringova, predstavljaju se u linearnoj memoriji modula. Sastoji se od tri glavne sekcije od kojih su uvodna sekcija (eng. *Preamble section*), poznata sekcija (eng. *Known section*) i prilagođena sekcija (eng. *Custom section*) (Gallant, 2019) (vlastiti prijevod).

Preamble – zasebna sekcija koja označava da je ovo modul i predstavlja verziju binarnog formata WebAssemblyja koja se koristi (Gallant, 2019) (vlastiti prijevod).

Known sections – mogu se uključiti samo jednom i pojavljuju se točno određenim redoslijedom. U poznate sekcije spadaju sekcije Type, Import, Function, Table, Memory, Global, Export, Start, Element, Code, Data. Svaka od ovih poznatih sekcija imaju specifičnu svrhu, te su dobro provjerene i definirane prilikom istanciranja. Ove sekcije su opcionalne (Gallant, 2019) (vlastiti prijevod).

Custom sections – Prilagođena sekcija pruža način uključivanja podataka unutar modula za namjene koje se ne odnose na poznate odjeljke (Gallant, 2019) (vlastiti prijevod). Mogu se pojaviti bilo gdje u modulu (prije, između ili nakon poznatih

odjeljaka) nebrojeno puta, koji čak mogu ponovno koristiti isto ime. Za razliku od poznatih sekcija, ako prilagođeni odjeljak nije ispravno postavljen, neće izazvati pogrešku provjere valjanosti. Prilagođene sekcije može lijeno učitavati okvir, što znači da podaci koje sadrže možda neće biti dostupni do nekog trenutka nakon inicijalizacije modula (Gallant, 2019) (vlastiti prijevod). Za WebAssembly MVP definirana je prilagođena sekcija pod nazivom **name**. Ideja ovog odjeljka je da možete imati verziju za otklanjanje pogrešaka WebAssembly modula, a ovaj odjeljak bi sadržavao nazive funkcija i varijabli u tekstualnom obliku za korištenje prilikom otklanjanja pogrešaka (Gallant, 2019) (vlastiti prijevod). Za razliku od drugih prilagođenih odjeljaka, ovaj bi se odjeljak trebao pojaviti samo jednom i samo nakon odjeljka data (Gallant, 2019) (vlastiti prijevod). Ove sekcije su također opcionalne. Na slici 11 je prikazana struktura WebAssembly modula, gdje se vidi da modul započinje s uvodnom sekcijom iz koje se vidi verzija modula.



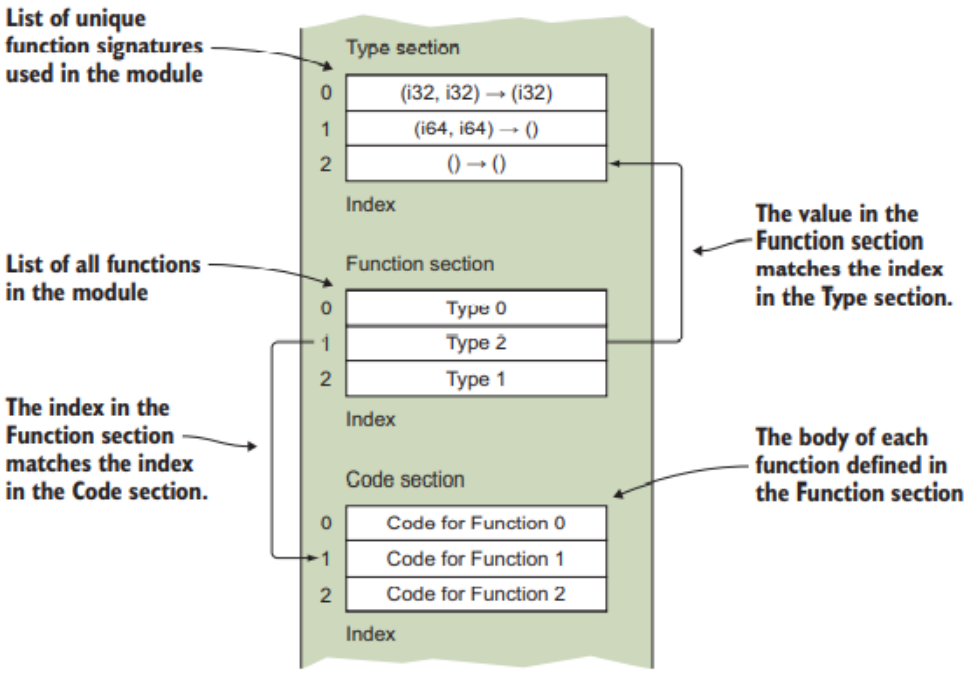
Slika 11: Osnovna struktura WebAssembly binarnog koda (Gallant, 2019)

Nakon uvodne sekcije, ako je neka od poznatih sekcija uključena, mora biti poredana po slijedećem redoslijedu:

- **Type** – deklarira popis svih jedinstvenih potpisa funkcija koji će se koristiti u modulu, uključujući one koji će biti uvezeni. Više funkcija može dijeliti isti

potpis. Primjer je prikazan na slici 11 gdje je vidljivo da se prva funkcija sastoji od dva 32 – bitna cjelobrojna (**i32**) parametra, te je isto tako vrijednost koja se vraća 32 – bitni cijeli broj. (Gallant, 2019) (vlastiti prijevod).

- **Import** – deklarira sve uvoze (eng. *import*) koji će se koristiti u modulu, što može uključivati funkcije, tablice, memoriju ili globalne uvoze. Uvozi su dizajnirani tako da moduli mogu dijeliti kod i podatke, ali i dalje dopuštaju da se moduli prevedu i spremaju zasebno. Uvoze osigurava okruženje domaćina kada je modul instanciran (Gallant, 2019) (vlastiti prijevod).
- **Function** – popis svih funkcija u modulu. Položaj deklaracije funkcije na ovom popisu predstavlja indeks tijela funkcije u *Code* sekciji. Vrijednost navedena u sekciji *Function* označava indeks potpisa funkcije u sekciji *Type*. Kako je vidljivo na slici 12, vrijednost druge funkcije je indeks potpisa funkcije koji nema nikakve parametre ili povratnu vrijednost. Indeks druge funkcije ukazuje na odgovarajući indeks u sekciji *Code* (Gallant, 2019) (vlastiti prijevod).



Slika 12: Veza između *Type*, *Function* i *Code* sekcija (Gallant, 2019)

- **Table** – sadrži upisani niz referenci, poput funkcija, koji se ne mogu pohraniti u linearnu memoriju modula kao neobrađeni bajtovi. Ovaj odjeljak pruža jedan od temeljnih sigurnosnih aspekata WebAssemblyja dajući WebAssembly okviru način za mapiranje objekata na siguran način. Kod nema izravan pristup

referencama pohranjenim u tablici. Umjesto toga, kada se želi pristupiti podacima navedenim u ovoj sekciji, traži da okvir radi na stavci na određenom indeksu u tablici. WebAssembly okvir zatim čita adresu pohranjenu na tom indeksu i izvodi akcija. Kada se radi o funkcijama, na primjer, to omogućuje korištenje pokazivača funkcija navođenjem indeksa tablice (Gallant, 2019) (vlastiti prijevod). Ovoj sekciji dodijeljena je inicijalna veličina, te opcionalno i maksimalna veličina, gdje veličinu *Table* sekcije predstavlja broj elemenata u njoj.

- **Memory** – sadrži linearnu memoriju koju koristi instanca modula. Također je temeljni sigurnosni aspekt WebAssemblyja jer moduli nemaju izravan pristup memoriji uređaja. Umjesto toga, okruženje koje instancira modul prolazi kroz ArrayBuffer koji instancu modula koristi kao linearnu memoriju. Što se koda tiče, ova linearna memorija djeluje kao hrpa u C++, ali svaki put kada kod pokuša pristupiti ovoj memoriji, okvir provjerava je li zahtjev unutar granica niza. Memorija modula definirana je kao WebAssembly stranice koje imaju po 64 KB svaka (1 KB je 1.024 bajta, dakle jedna stranica sadrži 65.536 bajtova) (Gallant, 2019) (vlastiti prijevod).
- **Global** – omogućuje definiranje globalnih varijabli za modul.
- **Export** – sadrži popis svih objekata koji će biti vraćeni u okruženje domaćina nakon što se modul instancira (dijelovi modula kojima okruženje domaćina može pristupiti). To može uključivati *Function*, *Table*, *Memory* ili *Global* izvoz (Gallant, 2019) (vlastiti prijevod).
- **Start** – deklarira indeks funkcije koja će se pozvati nakon što je modul inicijaliziran, ali prije nego što se izvezene funkcije mogu pozvati. Funkcija start može se koristiti kao način inicijalizacije globalnih varijabli ili memorije, te ako je navedena, funkcija se ne može uvesti, što znači da mora postojati unutar modula (Gallant, 2019) (vlastiti prijevod).
- **Element** – deklarira podatke koji se učitavaju u odjeljak tablice modula tijekom instanciranja (Gallant, 2019) (vlastiti prijevod).
- **Code** – sadrži tijelo svake funkcije deklarirane u sekciji *Function*. Svako tijelo funkcije mora se pojaviti istim redoslijedom kako je deklarirano.
- **Data** – deklarira podatke koji se učitavaju u linearnu memoriju modula tijekom instanciranja.

4.2 Emscripten

Emscripten je skup alata, odnosno potpuni alatni lanac (eng. *toolchain*) za prevođenje otvorenog koda koji se uvelike oslanja na LLVM[2] (eng. *Low Level Virtual Machine*, skraćeno LLVM). Omogućava prevođenje C i C++ koda, ili bilo kojeg drugog jezika koji koristi LLVM, u WebAssembly kojeg je moguće pokrenuti na webu, Node.js ili drugim wasm okolinama. Isto tako omogućava prevođenje C/C++ okoline drugih jezika u WebAssembly, nakon čega je moguće pokrenuti kod na tim drugim jezicima na neizravan način, kao na primjer, Python i Lua (Emscripten Contributors, 2015) (vlastiti prijevod).

Praktički svaki prijenosni C ili C++ kod se može sastaviti u WebAssembly pomoću Emscriptena, u rasponu od igara visokih performansi koje trebaju renderirati grafiku, reproduciranja zvukova, te učitavanja i obrade datoteka, do aplikacijskih okvira kao što je Qt. Emscripten se već koristio za pretvaranje vrlo dugog popisa baza koda iz stvarnog svijeta u WebAssembly, uključujući komercijalne proizvode kao što su Unreal Engine 4 i Unity Engine (Emscripten Contributors, 2015) (vlastiti prijevod). Emscripten generira mali i brz kod, izlaznog wasm formata koji je vrlo lak za optimizaciju i radi gotovo jednako brzo kao izvorni kod, a istovremeno je prenosiv i siguran (Emscripten Contributors, 2015) (vlastiti prijevod). Emscripten pažljivo automatski optimizira kod integracijom s LLVM – om, Binaryenom, Closure Compilerom i drugim alatima (Emscripten Contributors, 2015).

4.2.1 LLVM

LLVM projekt je zbirka modularnih tehnologija prevoditelja i alata, te unatoč svom imenu, LLVM jako malo ima veze s tradicionalnim virtualnim strojevima, gdje sam naziv LLVM nije akronim, već je puni naziv projekta. Započeo je 2000. godine kao istraživački projekt na Sveučilištu u Illinoisu, s ciljem pružanja moderne strategije kompilacije temeljene na SSA (eng. *Static Single Assignment*) koja može podržati statičku i dinamičku kompilaciju proizvoljnih programskih jezika. Od tada, LLVM je izrastao u glavni projekt koji se sastoji od niza pod projekata, od kojih se mnogi koriste u proizvodnji u raznim komercijalnim i otvorenim projektima (LLVM, 2000) (vlastiti prijevod). LLVM se sastoji od nekoliko pod projekata, a neki od njih su:

- **LLVM Core** – moderni neovisni optimizator, s podrškom za generiranje koda za mnoge CPU-ove. Ove biblioteke su izgrađene oko dobro specificiranog prikaza

[2] LLVM http://llvm.org/

koda poznatog kao LLVM među reprezentacija LLVM IR[3] (eng. *Intermediate Representation*). LLVM Core biblioteka je dobro dokumentirana, a moguće je lako napraviti vlastiti programski jezik, ili prenijeti postojeći za korištenje LLVM-a kao optimizatora i generatora koda (LLVM, 2000) (vlastiti prijevod).

- **Clang** – LLVM izvorni C/C++/Objective-C prevoditelj, koji ima za cilj nevjerojatno brzo prevođenje, iznimno korisne poruke o pogreškama i upozorenjima te pružiti platformu za izgradnju sjajnih alata izvorne razine. Clang Static Analyzer i clang – tidy su alati koji automatski pronalaze greške u kodu i odličan su primjeri vrste alata koji se mogu izraditi korištenjem Clang frontenda kao biblioteka za raščlanjivanje C/C++ koda (LLVM, 2000) (vlastiti prijevod).
- **LLDB** – izgrađen od biblioteka LLVM i *Clang* kako bi se pružio odličan izvorni program za ispravljanje pogrešaka. Koristi *Clang* AST-ove i parser izraza, LLVM JIT, LLVM disassembler, itd. tako da pruža iskustvo koje "jednostavno radi". Također je jako brz i mnogo učinkovitiji od GDB-a pri učitavanju simbola (LLVM, 2000) (vlastiti prijevod).
- **libC++ i libC++ ABI** – pružaju standardnu sukladnu implementaciju C++ standardne biblioteke visokih performansi, uključujući punu podršku za C++11 i C++14 (LLVM, 2000) (vlastiti prijevod).
- **compiler – rt** – pruža vrlo prilagođene implementacije rutina za podršku generatora koda niske razine poput *__fixunsdfdi*[4] i drugih poziva generiranih kada cilj nema kratki slijed izvornih uputa za implementaciju osnovne IR operacije. Također pruža implementacije knjižnica izvođenja pri pokretanju (eng. *runtime*) za alate koji se koriste za dinamičko testiranje kao što su AddressSanitizer, ThreadSanitizer, MemorySanitizer i DataFlowSanitizer (LLVM, 2000) (vlastiti prijevod).
- **MLIR** – je novi pristup izgradnji višekratne i proširive prevoditeljske infrastrukture. MLIR ima za cilj riješiti fragmentaciju softvera, poboljšati kompilaciju za heterogeni hardver, značajno smanjiti troškove izgradnje kompilatora specifičnih za domenu i pomoći u povezivanju postojećih kompilatora (LLVM, 2000) (vlastiti prijevod).

[3] Programski jezik niske razine poput asemblera (Wikipedia, 2022b)
[4] Rutina, odnosno funkcija koja pretvara broj u cijeli broj bez predznaka, zaokružujući prema nuli. Sve negativne vrijednosti postaju nula (Soft float library routines, 1988).

- **OpenMP** – potprojekt pruža OpenMP runtime za korištenje s OpenMP implementacijom u Clang (LLVM, 2000) (vlastiti prijevod).
- **Polly** – implementira skup optimizacija cache – locality kao i autoparalelizam i vektorizaciju koristeći poliedarski model[5] (LLVM, 2000) (vlastiti prijevod).
- **Libclc** – ima za cilj implementaciju standardne biblioteke OpenCL (LLVM, 2000).
- **Klee** – implementira simbolički virtualni stroj koji koristi dokazivač teorema da pokuša procijeniti sve dinamičke putove kroz program u nastojanju da pronađe greške i dokaže svojstva funkcija. Glavna značajka je da može proizvesti test u slučaju da otkrije grešku (LLVM, 2000) (vlastiti prijevod).
- **LLD** – novi linker i zamjena za sistemske linkere i radi puno brže.

Ispod haube, Emscripten prevodi C++ u JavaScript u dvije glavne faze. Prvo, Emscripten poziva LLVM-ov Clang da prevede C i C++ izvorni kod u LLVM IR, koji je prikaz programa negdje između izvornog koda i prevedenog objektnog koda. Nakon toga, Emscripten prevodi LLVM IR u optimizirani JavaScript, kako je prikazano na slici 13 (McAnlis *et al.*, 2014) (vlastiti prijevod).

Slika 13: Tijek rada Emscripten prevodioca (McAnlis *et al.*, 2014)

Clang prevodi izvorni kod u LLVM IR, koji je u osnovi tipski i hardverski neovisan asemblerski jezik. LLVM IR definira funkcije, a njegove upisane lokalne varijable otprilike odgovaraju CPU registrima. Na primjer, funkcija napisana u C pod nazivom lerp za linearnu interpolaciju dva broja s pomičnim zarezom prikazana na slici 14 (McAnlis *et al.*, 2014) (vlastiti prijevod).

```
float lerp(float a, float b, float t) {
    return (1 - t) * a + t * b;
}
```

Slika 14: Primjer lerp funkcije napisane u C

Ista funkcija lerp bila bi predstavljena u LLVM IR kao što je prikazano na slici 15.

[5] Poliedar https://bs.wikipedia.org/wiki/Poliedar

```
define internal hidden float @_lerp(float %a, float %b, float %t) nounwind readnone inlinehint ssp {
  %1 = fsub float 1.000000e+00, %t
  %2 = fmul float %1, %a
  %3 = fmul float %t, %b
  %4 = fadd float %2, %3
  ret float %4
}
```

Slika 15: lerp u LLVM IR (McAnlis *et al.*, 2014)

LLVM IR hvata semantičko značenje C koda, ali mu daje dosljednu strukturu tako da ga Emscripten može prevesti u JavaScript. Nakon što Clang pretvori izvorni kod C ili C++ u LLVM IR, tada Emscripten prevodi LLVM IR u JavaScript (McAnlis *et al.*, 2014) (vlastiti prijevod). Koristi se činjenica da JavaScript ima operatore i izraze koji odgovaraju semantici C za 32-bitnu cjelobrojnu matematiku s predznakom i bez predznaka (McAnlis *et al.*, 2014) (vlastiti prijevod). Gore spomenuta funkcija lerp bi se prevela i optimizirala u JavaScript prikazan na slici 16.

```
function _lerp(a, b, t) {
  return(1 - t) * a + t * b
}
```

Slika 16: lerp prevedena u JavaScript (McAnlis *et al.*, 2014)

4.2.2 Reprezentacija memorije

Specifikacija tipiziranog polja (eng. *Typed Array*) uvodi mehanizam pomoću kojeg JavaScript može čitati i pisati u susjedne blokove binarnih podataka. ArrayBuffer pohranjuje niz susjednih bajtova kojima se može pristupiti i interpretirati kroz objekte ArrayBufferView (McAnlis *et al.*, 2014) (vlastiti prijevod). Na primjer, Int8Array objekt izlaže ArrayBuffer kao da su potpisani 8-bitni cijeli brojevi, a Float32Array izlaže istu memoriju kao da je riječ o 32-bitnim floatovima (McAnlis *et al.*, 2014) (vlastiti prijevod). Emscripten memorijski prostor je jedan JavaScript ArrayBuffer kojem se pristupa kroz jedan od ArrayBufferView (McAnlis *et al.*, 2014). Na primjer, HEAP8 je Int8Array kroz koji su potpisane 8-bitne cjelobrojne vrijednosti čitaju i zapisuju u memoriju programa. Pokazivači su jednostavno numerički indeksi u hrpi (eng. *heap*). Emscripten ima ArrayBufferViews za 32-bitne i 64-bitne float, kao i cijeli niz 8-bitni, 16-bitni, i 32-bitni cijeli brojevi, predpisani (eng. *signed*) i neoznačeni (eng. *unsigned*) (McAnlis *et al.*, 2014) (vlastiti prijevod). U JavaScriptu su svi brojevi 64-bitni float. Korištenje brojeva s pomičnim zarezom za predstavljanje pokazivača u hrpi bilo bi glupo i sporo, te kroz pametnu analizu tipova, JavaScript engine može čak primijetiti kada je moguće

reducirati 64-bitne precizne operacije s pomičnim zarezom na 32-bitne precizne operacije s pomičnim zarezom bez promjene semantike koda (McAnlis *et al.*, 2014). Zato je JavaScript generiran sa Emscripten jako brz. Koristi kompaktni ArrayBuffer za memoriju, tako da sakupljač smeća (eng. *garbage collector*) ne mora ništa raditi. Varijable su uvijek brojevi, a ne objekti, koji znači da nema poziva dinamičkih metoda ili traženja u hash tablici (McAnlis *et al.*, 2014) (vlastiti prijevod). Pametno korištenje JavaScript operatora |, >> i >>> označava da kod ima cjelobrojnu semantiku. Sve to znači da optimizatori samo na vrijeme mogu izravno prevesti JavaScript u brzi strojni kod. Ove JavaScript konvencije visokih performansi kodificirane su u standardu nazvanom asm.js o kojem je ranije u ovom radu bilo govora (McAnlis *et al.*, 2014).

4.2.3 Alatni lanac

Emscripten se sastoji od tri dijela, prevoditelja od LLVM-a do JavaScripta, skupa implementacija biblioteka koje olakšavaju prijenos postojećih kodnih baza, te korisnih alata i skripti za upravljanje cijelim procesom prevođenja. Jedan od tih alata je emcc, ponaša se slično kao gcc i može se ubaciti u većinu postojećih sustava (Za C++, em++ je Emscripten zamjena za g++) (McAnlis *et al.*, 2014) (vlastiti prijevod). Umjesto da proizvode izvorne objektne datoteke, emcc i em++ proizvode LLVM IR datoteke. Prilikom povezivanja, umjesto generiranja izvorne izvršne datoteke, konačni cilj izvršne datoteke je ili HTML stranica ili jedna JavaScript datoteka, ovisno o tome hoće li program biti samostalan ili će biti integriran u postojeću web aplikaciju. emcc i em++ nude desetke opcija specifičnih za Emscripten, uključujući prilagođene optimizacije i izlazni format prilagodbe (McAnlis *et al.*, 2014) (vlastiti prijevod). Dakle, prevođenje jednostavnog programa „Hello World“ s Emscriptenom je jednostavno kako je prikazano na slici 17 (McAnlis *et al.*, 2014).

```
$ cat helloworld.c
#include <stdio.h>
int main() {
    printf(“Hello world!\n”);
}
$ emcc -o helloworld.html helloworld.c
```

Slika 17: Prevođenje programa sa emcc (McAnlis *et al.*, 2014)

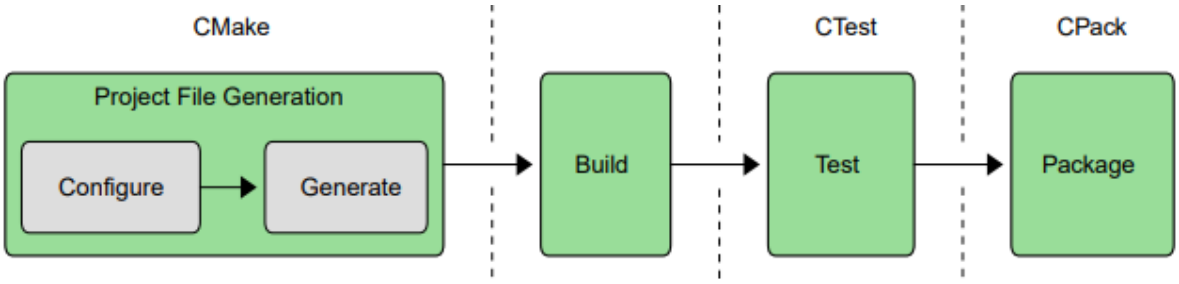
4.3 CMake

CMake je proširivi sustav otvorenog koda koji upravlja procesom izgradnje u operativnom sustavu neovisan o prevoditelju (Cmake, 2000) (vlastiti prijevod). Za razliku od mnogih višeplatformskih sustava, CMake je dizajniran za korištenje u sprezi s izvornim (eng. *native*) okruženjem za izgradnju (Cmake, 2000) (vlastiti prijevod). Jednostavne konfiguracijske datoteke (CMakeLists.txt) smještene u svaki izvorni direktorij, koriste se za generiranje standardnih datoteka za izgradnju (npr. makefiles na Unixu i na Windows MSVC) koje se koriste na uobičajeni način (Cmake, 2000) (vlastiti prijevod). CMake može generirati izvorno okruženje za izgradnju koje će prevesti izvorni kod, kreirati biblioteke, generirati programske omotače oko nekog koda (eng. *wrappers*) i graditi izvršne datoteke u proizvoljnim kombinacijama (Cmake, 2000) (vlastiti prijevod).

Podržava in source i out of source gradnje, te stoga može podržati višestruku izgradnju iz jednog izvornog stabla. Također podržava statičku i dinamičku izgradnju biblioteka. Još jedna lijepa značajka CMakea je da generira cache datoteku koja je dizajnirana za korištenje s grafičkim editorom (Cmake, 2000) (vlastiti prijevod). Na primjer, kada se CMake pokrene, locira datoteke, knjižnice i izvršne datoteke i može naići na neobavezne direktive izgradnje. Te se informacije skupljaju u predmemoriju koju korisnik može promijeniti prije generiranja datoteka izvorne gradnje. CMake je dizajniran da podržava složene hijerarhije direktorija i aplikacije koje ovise o nekoliko knjižnica (Cmake, 2000) (vlastiti prijevod). Na primjer, CMake podržava projekte koji se sastoje od više skupova alata, tj. biblioteka, gdje svaki alat može sadržavati nekoliko direktorija, a aplikacija ovisi o kompletima alata i dodatnom kodu (Cmake, 2000).

Također može rješavati situacije u kojima se izvršne datoteke moraju izgraditi kako bi se generirao kod koji se zatim kompilira i povezuje u konačnu aplikaciju. Budući da je otvorenog koda i ima jednostavan, proširiv dizajn, može se proširiti prema potrebi kako bi podržao nove značajke. Proces izgradnje kontrolira se stvaranjem jedne ili više datoteka CMakeLists.txt u svakom direktoriju, uključujući poddirektorije koji čine projekt. Svaki CMakeLists.txt sastoji se od jedne ili više naredbi. Svaka naredba ima oblik COMMAND (args…) gdje je COMMAND naziv naredbe, a args je popis argumenata odvojen razmacima (Cmake, 2000) (vlastiti prijevod). CMake nudi mnoge unaprijed definirane naredbe, ali ako trebate svoje vlastite naredbe. Osim toga, napredni korisnik može dodati druge makefile generatore za određenu kombinaciju prevoditelj / operacijski sustav (Cmake, 2000) (vlastiti prijevod).

Prva faza uzima generički opis projekta i generira projektne datoteke specifične za platformu prikladne za korištenje s uobičajenim alatom za izgradnju po izboru (npr. make, Xcode, Visual Studio, itd.). Iako je ova faza postavljanja ono po čemu je CMake najpoznatiji, CMake paket alata također uključuje CTest i CPack za upravljanje kasnijim fazama testiranja i pakiranja, koji se za potrebe ovog diplomskog rada ne koriste. Cijeli proces od početka do kraja može se pokretati iz samog CMakea, a koraci testiranja i pakiranja dostupni su jednostavno kao dodatni ciljevi u izradi, te CMake može pozvati čak i alat za izgradnju (Scott, 2018) (vlastiti prijevod). Na slici 18 prikazano je proces izgradnje projekta sa CMake.



Slika 18: Proces izgradnje projekta sa CMake (Scott, 2018)

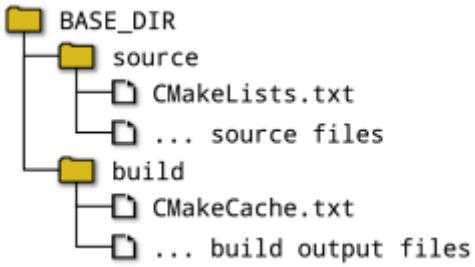
4.3.1 In source izgradnja

Pod in source izgradnjom se podrazumijeva da se sve datoteke, koje čine projekt, kao na primjer C++ source, C++ headeri, nalaze unutar jednog direktorija, te sva konfiguracija za CMake se nalazi unutar tog istog direktorija u datoteci CMakeLists.txt. Na prvu izgleda jednostavno i praktično držati sve na istom mjestu, ali u praksi to nije tako, naročito ako projekt naraste, mogući su (gotovo sigurno) scenariji miješanja source i build datoteka, te mnogih drugih datoteka i pod direktorija unutar projekta (Scott, 2018) (vlastiti prijevod). To također otežava rad sa sustavima kontrole verzija, budući da postoji mnogo datoteka koje kreira build koje alat za kontrolu sourcea mora znati zanemariti ili programer mora ručno isključiti tijekom komita (Scott, 2018) (vlastiti prijevod). Još jedan od nedostatak je taj što može biti netrivijalno obrisati sve izlazne podatke builda i početi ponovno s čistim source stablom, stoga se ovaj tip izgradnje izbjegava (Scott, 2018) (vlastiti prijevod).

4.3.2 Out of source izgradnja

Kod out of source načina izgradnje projekta source i build, te ostali direktoriji su odvojeni jedan od drugog, od kojih svaki direktorij može imati nekolicinu datoteka i pod

direktorija, čime se izbjegavaju problemi miješanja s kojima se susreću in source buildovi. Unutar svakog direktorija nalazi se CMakeLists.txt datoteka u kojoj su definirane CMake komande i druge naredbe. Polazna točka je unutar CMakeLists.txt koja se nalazi u glavnom direktoriju projekta, u kojoj su umetnuti (eng. *include*), to jest definirani svi pod direktoriji i datoteke projekta. Prednost ovog tipa izgradnje projekta je i ta što je moguće imati više definicija, kako sustava za provjeru verzija, tako i konfiguracije kao na primjer produkcijske ili razvojne konfiguracije, koje je moguće izvršavati ovisno o razvojnoj okolini, platformi itd. (Scott, 2018) (vlastiti prijevod). Jednostavan primjer strukture projekta prikazan je na slici 19.



Slika 19: Out of source struktura (Scott, 2018)

Moguća je i varijacija izgradnje da direktorij za build postave kao poddirektorij source direktorija. To nudi većinu prednosti izgradnje izvan izvora, ali još uvijek sa sobom nosi neke nedostatke, osim ako ne postoji dobar razlog za takvo strukturiranje stvari, umjesto toga preporuča se držanje direktorija build potpuno izvan source stabla (Scott, 2018) (vlastiti prijevod).

4.3.3 Generiranje projektnih datoteka

Nakon odabira strukture direktorija, pokreće se CMake, gdje se čita konfiguracija iz datoteke CMakeLists.txt iz glavnog direktorija projekta i stvara projektne datoteke u direktoriju source. Odabire se vrsta projektne datoteke koju će stvoriti odabirom određenog generatora projektne datoteke.

Neki od generatora generiraju projekte koji podržavaju više konfiguracija (npr. Debug, Release, itd.). Oni omogućuju odabir između različitih konfiguracija izgradnje bez potrebe za ponovnim pokretanjem CMakea, što je prikladnije za generatore koji stvaraju projekte za korištenje u IDE okruženjima kao što su Xcode i Visual Studio. Za generatore koji ne podržavaju više konfiguracija, potrebno je ponovno pokrenuti CMake kako bi se prebacila izgradnja za Debug, Release, itd. Oni su jednostavniji i

često imaju dobru podršku u IDE okruženjima koja nisu tako blisko povezana s određenim prevodiocem (Qt Creator , KDevelop itd.). Dakle, održan je niz različitih generatora, od kojih su najčešće korišteni navedeni u tablici (Scott, 2018).

Tablica 13: CMake generatori (Scott, 2018)

Kategorija	Primjeri generatora	Multi konfiguracija
Visual Studio	**Visual Studio 15 2017** **Visual Studio 14 2015** **…**	**DA**
Xcode	**Xcode**	**DA**
Ninja	**Ninja**	**NE**
Makefiles	**Unix Makefiles** **MSYS Makefiles** **MinGW Makefiles** **NMake Makefiles**	**NE**

Najosnovniji način za pokretanje CMakea je putem komande u komandnoj liniji, tako da se proslijedi putanja datoteke koja se nalazi unutar build direktorija, te se CMake komandi proslijedi putanja odredišnog direktorija gdje se želi buildati i tip genratora koji se želi koristiti za source stablo, kako je jednostavnim primjerom prikazano na slici 20 (Scott, 2018) (vlastiti prijevod).

```
mkdir build
cd build
cmake -G "Unix Makefiles" ../source
```

Slika 20: Izvršavanje cmake komande u komandnoj liniji (Scott, 2018)

Komandom mkdir build se kreira direktorij build, gdje se nakon kreiranja komandom cd build navigira unutar build direktorija u kojem bi se trebala nalaziti datoteka CMakeLists.txt. Tada se izvršava komanda cmake -G “Unix Makefiles“ ../source, -G predstavlja oznaku za odabir generatora nakon čega se navodi željeni generator i putanja do source direktorija u koji će se generirati projektne datoteke nakon čega bi ispis u konzoli trebao izgledati kao na slici 21 (Scott, 2018).

```
-- Configuring done
-- Generating done
-- Build files have been written to: /some/path/build
```

Slika 21: Ispis nakon izvršenja cmake komande (Scott, 2018)

Ako je opcija -G izostavljena, CMake će odabrati zadani tip generatora na temelju platforme na kojoj se izvršava cmake. Za sve tipove generatora, CMake će provesti

niz testova i ispitati sustav kako bi utvrdio kako postaviti projektne datoteke. To uključuje stvari kao što je provjera rada prevoditelja, određivanje skupa podržanih značajki prevoditelja i razne druge zadatke (Scott, 2018) (vlastiti prijevod).

Kreiranje projektne datoteke u ovom slučaju, zapravo uključuje dva koraka, konfiguriranje i generiranje. Tijekom faze konfiguriranja, CMake čita datoteku CMakeLists.txt i izgrađuje interni prikaz cijelog projekta. Nakon što je to učinjeno, faza generiranja stvara datoteke projekta. Kada CMake izvrši pokretanje, generira se datoteka CMakeCache.txt u direktoriju build, u koju se spremaju pojedinosti tako da, ako se CMake ponovno pokrene, može koristiti informacije izračunate prvi put i ubrzati generiranje projekta (Scott, 2018) (vlastiti prijevod). Alternativa alata za izgradnju u komandnoj liniji je cmake – gui.

4.3.4 Pokretanje alata za izgradnju

U ovom trenutku nakon konfiguriranja i generiranja, moguće je odabrati alat koji se želi koristiti za izgradnju, GUI ili neki drugi tipa alata poput komandne linije. --build opcija ukazuje na build direktorij koji koristi prilikom izvršavanja koraka generiranja projekta CMake (Scott, 2018) (vlastiti prijevod). Za generatore s multi konfiguracijom, opcija --config određuje koju će konfiguraciju izgraditi, dok će generatori pojedinačnih konfiguracija zanemariti opciju --config i umjesto toga oslanjati se na informacije koje su dostupne kada je izveden korak generiranja CMake projekta (Scott, 2018) (vlastiti prijevod). Opcija --target može se koristiti da se alatu za izgradnju kaže što da izgradi, ili ako je izostavljena, bit će izgrađen zadani cilj. Primjer izgradnje prikazuje slika 22 (Scott, 2018).

```
cmake --build /some/path/build --config Debug --target MyApp
```

Slika 22: Izvršavanje komande za generiranje projekta (Scott, 2018)

--build opcija ukazuje na build direktorij koji koristi CMake korak generiranja projekta. Za generatore s više konfiguracija, opcija --config određuje koju konfiguraciju treba izgraditi, dok će generatori pojedinačnih konfiguracija zanemariti opciju --config i umjesto toga, oslanjati se na informacije koje su dane kada je izveden korak generiranja CMake projekta (Scott, 2018) (vlastiti prijevod). Opcija --target može se koristiti da se alatu za izgradnju kaže što da izgradi, ili ako je izostavljena, bit će izgrađen zadani cilj. Koristeći ovaj pristup, primjer komadni za izvršavanje može izgledati kao na slici 23 (Scott, 2018).

```
mkdir build
cd build
cmake -G "Unix Makefiles" ../source
cmake --build . --config Release --target MyApp
```

Slika 23: Generiranje projekta sa dodatnim parametrima (Scott, 2018)

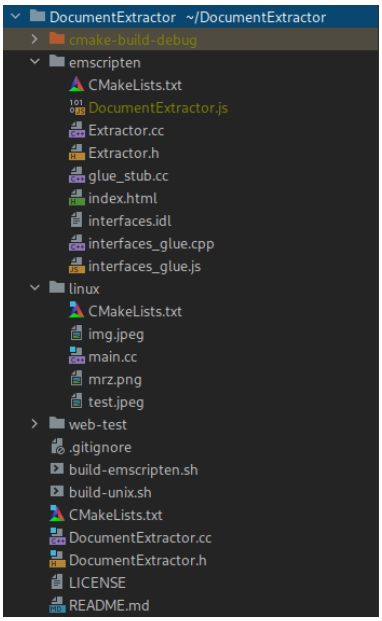
Dobra praksa je postavljanje dvije ili više različitih verzija za isti izvorni direktorij. Jedna se verzija može konfigurirati s postavkama za Debug, a druga za verziju Release. Druga mogućnost je korištenje različitih generatora projekata za različite direktorije izgradnje, kao što su Unix Makefiles i Xcode (Scott, 2018) (vlastiti prijevod). To može pomoći da se otkriju sve nenamjerne ovisnosti o određenom alatu za izradu ili da se provjeri postoje li različite postavke prevoditelja između tipova generatora (Scott, 2018) (vlastiti prijevod). Iako u početku se mogu koristiti generatori pojedinačnih konfiguracija, projekti imaju naviku da narastu izvan svog početnog opsega i može biti relativno uobičajeno da trebaju podržavati dodatne platforme i stoga različite vrste generatora (Scott, 2018) (vlastiti prijevod). Periodično provjeravanje build-a s drugačijim generatorom projekta od onog što se obično koristi može uštedjeti znatne buduće muke kod koda koji je specifičan za generator tamo gdje nije potreban. Ovo također ima prednost jer je projekt u dobroj poziciji da iskoristi prednosti bilo kojeg novog tipa generatora u budućnosti (Scott, 2018) (vlastiti prijevod). Dobra strategija bila bi osigurati da se projekt gradi sa zadanim tipom generatora na svakoj platformi od interesa, plus još jedan tip (Scott, 2018) (vlastiti prijevod). Ninja generator je izvrstan izbor za potonje, budući da ima najširu platformsku podršku od svih generatora i također stvara vrlo učinkovite konstrukcije (Scott, 2018) (vlastiti prijevod). Ako je projekt skriptiran, pozove se alat za izgradnju putem cmake --build umjesto izravnog pozivanja alata za izgradnju. To omogućuje skripti da se lako prebaci između tipova generatora bez potrebe za izmjenom (Scott, 2018).

5. Implementacija programskog rješenja

Projekt je nazvan Document Extractor jer postoji mogućnost proširenja same biblioteke za detekciju znakova na osobnim dokumentima, dakle nije strogo definirano da projekt vrši samo detekciju strojno čitljive zone i ekstrakciju podataka na osobnim dokumentima. Moguće je proširiti funkcionalnost biblioteke da se, npr. detektira slika na osobnom dokumentu i na osnovu te slike izvrši prepoznavanje lica (eng. *face detection*) i slično, ali ovaj projekt se bavi samo implementacijom detekcije i prepoznavanje znakova strojno čitljive zone.

5.1 Struktura i okruženje projekta Document Extractor

Struktura projekta je rađena po principu out of source gdje direktoriji za build i source, koji mogu biti nazvani kako god se želi, potpuno odvojeni jedan od drugog kako je objašnjeno u prethodnom poglavlju. Glavni direktorij DocumentExtractor, prije izvršenja komande cmake za izgradnju biblioteke, sastoji se od 3 pod direktorija emscripten, linux i web-test, od kojih svaki od njih sadrži datoteku CMakeLists.txt u kojima su definirana pravila izgradnje za svaki od direktorija, osim direktorija web-test iako se nalazi unutar glavnog direktorija projekta, potpuno je neovisan od ostatka projekta, te je pritom minimalan projekt koji služi za testiranje biblioteke na web pregledniku. Na slici 24 prikazana je struktura projekta.



Slika 24: Struktura projekta DocumentExtractor

U glavnom direktoriju projekta, pored datoteke CMakeLists.txt, koja je početna točka samog projekta, nalazi se još i nekoliko datoteka od kojih su:

- build-emscripten.sh
- build-unix.sh
- DocumentExtractor.h
- DocumentExtractor.cc

Razlog korištenja sh skripti je taj što korištenjem takve skripte olakšava definiranje parametara i postavki za generiranje biblioteka Emscripten i OpenCV, gdje pri tome pokretanjem takve skripte u terminalu izvršava sve naredbe unutar skripte koje su definirane kao na primjer s kreiranje potrebnih direktorija, skidanje sourcea s nekog od repozitorija, konkretno u ovom slučaju radi se o OpenCV biblioteci. Inače bi se sve ove komande morale ručno izvršiti unutar terminala.

Emscripten biblioteka je za potrebe rada instalirana unutar sistemskog direktorija, ali je isto, kao i biblioteku OpenCV moguće skinuti source u direktorije projekta i prevesti biblioteku za korištenje. Datoteka s ekstenzijom sh je datoteka naredbi jezika skriptiranja koja sadrži računalni program koji pokreće Unix ljuska. Može sadržavati niz naredbi koje se pokreću uzastopno za obavljanje operacija kao što su obrada datoteka, izvršavanje programa i drugi slični zadaci. Oni se izvršavaju iz sučelja naredbenog retka od strane korisnika ili u paketu kako bi se izvršilo više operacija u isto vrijeme. Datoteke skripte mogu se otvoriti u uređivačima teksta kao što su Notepad, Notepad++, Vim, Apple Terminal i druge slične aplikacije na Windows, MacOS i Linux OS (Fileformat, 2001) (vlastiti prijevod).

U nastavku se opisuju sadržaj sh skripti, od kojih je build-emscripten.sh konfigurirana da izvrši izgradnju vezanu uz Emscripten, dok build-unix.sh služi za izgradnju vezanu uz Unix[6] sustave, poput Linux i MacOS. Windows platforma je za sada izostavljena.

Datoteka build-emscripten.sh, kao i build-unix.sh, pokreće se pomoću komande sh build-emscripten.sh ili ako je već postavljena kao izvršna (eng. *executable*) dovoljno je pokrenuti ./build-emscripten.sh u terminalu, izvršava definirane naredbe jednu po jednu, koje bi se inače ručno morale utipkavati u terminal. Na slici 25 prikazan je dio skripte build-emscripten.sh.

```
1.  #!/bin/sh
2.  EmscriptenToolChainFile=""
3.
4.  case "$OSTYPE" in
5.    linux*)
    EmscriptenToolChainFile=/usr/lib/Emscripten/cmake/Modules/Platform/Emscripten.cmake ;;
```

[6] Unix Systems - http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html

```
6.    darwin*)
    EmscriptenToolChainFile=/usr/local/Cellar/Emscripten/2.0.25/libexec/cmake/Modules/Platfor
    m/Emscripten.cmake ;;
7.  cmake -D CMAKE_BUILD_TYPE=RELEASE \
8.        -D CMAKE_INSTALL_PREFIX="$cwd"/lib-Emscripten/opencv \
9.        -D CMAKE_TOOLCHAIN_FILE="$EmscriptenToolChainFile" \
10.       -D ENABLE_PIC=FALSE \
11.       -D CPU_BASELINE='' \
12.       -D CPU_DISPATCH='' \
13.       -D CV_TRACE=OFF \
14.       -D BUILD_SHARED_LIBS=OFF \
15.       -D BUILD_PROTOBUF=OFF \
16.       -D WITH_PROTOBUF=OFF \
17.       -D WITH_1394=OFF \
18.       -D WITH_ADE=OFF \
19.       -D WITH_VTK=OFF \
20.       -D WITH_EIGEN=OFF \
21.       -D WITH_FFMPEG=OFF \
22.       -D WITH_GSTREAMER=OFF \
23.       -D WITH_GTK=OFF \
24.       -D WITH_GTK_2_X=OFF \
25.       -D WITH_IPP=OFF \
26.       -D WITH_JASPER=OFF \
27.       -D WITH_JPEG=OFF \
28.       -D WITH_WEBP=OFF \
29.       -D WITH_OPENEXR=OFF \
30.       -D WITH_OPENGL=OFF \
31.       -D WITH_OPENVX=OFF \
32.       -D WITH_OPENNI=OFF \
33.       -D WITH_OPENNI2=OFF \
34.       -D WITH_PNG=OFF \
35.       -D WITH_TBB=OFF \
36.       -D WITH_TIFF=OFF \
37.       -D WITH_V4L=OFF \
38.       -D WITH_OPENCL=OFF \
39.       -D WITH_OPENCL_SVM=OFF \
40.       -D WITH_OPENCLAMDFFT=OFF \
41.       -D WITH_OPENCLAMDBLAS=OFF \
42.       -D WITH_GPHOTO2=OFF \
43.       -D WITH_LAPACK=OFF \
44.       -D WITH_ITT=OFF \
45.       -D WITH_QUIRC=OFF \
46.       -D BUILD_ZLIB=ON \
47.       -D BUILD_opencv_core=ON \
48.       -D BUILD_opencv_apps=OFF \
49.       -D BUILD_opencv_calib3d=OFF \
50.       -D BUILD_opencv_dnn=OFF \
51.       -D BUILD_opencv_features2d=ON \
52.       -D BUILD_opencv_flann=ON \
53.       -D BUILD_opencv_gapi=OFF \
54.       -D BUILD_opencv_ml=OFF \
55.       -D BUILD_opencv_photo=OFF \
56.       -D BUILD_opencv_imgcodecs=OFF \
57.       -D BUILD_opencv_shape=OFF \
58.       -D BUILD_opencv_videoio=OFF \
59.       -D BUILD_opencv_videostab=OFF \
60.       -D BUILD_opencv_highgui=OFF \
61.       -D BUILD_opencv_superres=OFF \
62.       -D BUILD_opencv_stitching=OFF \
63.       -D BUILD_opencv_java=OFF \
64.       -D BUILD_opencv_js=OFF \
65.       -D BUILD_opencv_python2=OFF \
66.       -D BUILD_opencv_python3=OFF \
67.       -D BUILD_EXAMPLES=OFF \
68.       -D BUILD_PACKAGE=OFF \
69.       -D BUILD_TESTS=OFF \
70.       -D BUILD_PERF_TESTS=OFF \
71.       -D BUILD_DOCS=OFF \
72.       -D WITH_PTHREADS_PF=ON \
73.       -D CV_ENABLE_INTRINSICS=ON \
```

```
74.       -D BUILD_WASM_INTRIN_TESTS=OFF \
75.       -D CMAKE_C_FLAGS="-s WASM=1 -s USE_PTHREADS=0    -fsigned-char -W -Wall -
    Werror=return-type -Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -
    Wformat -Werror=format-security -Wmissing-declarations -Wmissing-prototypes -Wstrict-
    prototypes -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -Wuninitialized -
    Winconsistent-missing-override -Wno-delete-non-virtual-dtor -Wno-unnamed-type-template-
    args -Wno-comment -Wno-deprecated-enum-enum-conversion -Wno-deprecated-anon-enum-enum-
    conversion -fdiagnostics-show-option -pthread -Qunused-arguments -ffunction-sections -
    fdata-sections  -fvisibility=hidden -fvisibility-inlines-hidden -O3 -DNDEBUG  -DNDEBUG -
    msimd128" \
76.       -D CMAKE_CXX_FLAGS="-s WASM=1 -s USE_PTHREADS=0    -fsigned-char -W -Wall -
    Werror=return-type -Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -
    Wformat -Werror=format-security -Wmissing-declarations -Wmissing-prototypes -Wstrict-
    prototypes -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -Wuninitialized -
    Winconsistent-missing-override -Wno-delete-non-virtual-dtor -Wno-unnamed-type-template-
    args -Wno-comment -Wno-deprecated-enum-enum-conversion -Wno-deprecated-anon-enum-enum-
    conversion -fdiagnostics-show-option -pthread -Qunused-arguments -ffunction-sections -
    fdata-sections  -fvisibility=hidden -fvisibility-inlines-hidden -O3 -DNDEBUG  -DNDEBUG -
    msimd128" ..
77.
78. make
79. make install
80.
81. #export OpenCV_DIR="$cwd"/lib/opencv/lib/cmake/opencv4/OpenCVConfig.cmake
82. [[ ":$PATH:" != *":$cwd/lib-Emscripten/opencv/lib/cmake/opencv4/OpenCVConfig.cmake:"* ]]
    && PATH="$cwd/lib-Emscripten/opencv/lib/cmake/opencv4/OpenCVConfig.cmake:${PATH}"
83. echo "$PATH"
84.
85. cd "$cwd" || exit
86.
```

Slika 25: Primjer build-emscripten.sh skripte

U skripti se prvo izvršavaju komande za čišćenje opencv direktorija ako već postoji i nije prazan ili se kreira novi prilikom kloniranja opencv sourcea s repozitorija. Isto tako se kreira i direktorij u koji će se source izgraditi, što je u konkretnom slučaju lib-emscripten, te nakon što se klonira OpenCV source, u opencv direktoriju generira se novi build direktorij u koji se izgradi source. Nakon postavljanja putanje do Emscripten toolchain datoteke, u kojoj se nalazi konfiguracija za izgradnju Emscriptena i koja dolazi uz instalaciju Emscriptena, pokreće se komanda cmake s nekoliko dodatnih postavki s opcijom -D, kao na slici 26.

```
1.  -D CMAKE_INSTALL_PREFIX="$cwd"/lib-Emscripten/opencv  \
2.  -D CMAKE_TOOLCHAIN_FILE="$EmscriptenToolChainFile" \
```

Slika 26: Primjer dodatnih opcija build-emscripten.sh skripte

Na liniji 1. definira se putanja na kojoj će se nalaziti izgrađeni source, dok na liniji 2. definirana je putanja do Emscripten toolchan datoteke koja je spremljena u zasebnu varijablu i ta putanja ovisi o sustavu na kojem se izvršava gradnja, recimo Linux i MacOS nemaju isti instalacijski put. Pored mnogih opcija koje služe za generiranje koda za OpenCV izgradnju, nekoliko njih je vrlo važnih koje služe za uključivanje ili

isključivanje OpenCV modula, pošto je OpenCV modularno napisana biblioteka kako se već ranije spomenulo u radu. U nastavku na slici 27 su prikazane opcije za module OpenCV koji su uključeni ili isključeni iz izgradnje za Emscripten.

```
1.  -D BUILD_opencv_core=ON \
2.  -D BUILD_opencv_imgproc=ON \
3.  -D BUILD_opencv_apps=OFF \
4.  -D BUILD_opencv_calib3d=OFF \
5.  -D BUILD_opencv_dnn=OFF \
6.  -D BUILD_opencv_features2d=OFF \
7.  -D BUILD_opencv_flann=OFF \
8.  -D BUILD_opencv_gapi=OFF \
9.  -D BUILD_opencv_ml=OFF \
10. . . .
11. -D BUILD_opencv_videoio=OFF \
12. -D BUILD_opencv_videostab=OFF \
13. -D BUILD_opencv_highgui=OFF \
14. -D BUILD_opencv_superres=OFF \
15. -D BUILD_opencv_stitching=OFF \
16. -D BUILD_opencv_java=OFF \
17. -D BUILD_opencv_js=OFF \
18. -D BUILD_opencv_python2=OFF \
19. -D BUILD_opencv_python3=OFF \
```

Slika 27: OpenCV moduli za Emscripten okolinu

Iz priloženog se vidi da su uključeni samo core modul i imgproc modul za OpenCV build sa Emscripten, gdje su svi ostali isključeni jer dio koji će Emscripten izgraditi za WebAssembly nije niti potrebno. Potrebna je samo funkcionalnost iz core modula i funkcionalnost za obradu slika iz imgproc modula. Pošto se koristi samo dva modula, time je generirani Emscripten source za WebAssembly znatno manje veličine, što uvelike omogućava brže učitavanje modula na web stranici. Isto tako postoji mogućnost za izgradnju koda za programski jezik java, python i JavaScript, za koje je također moguće odrediti koji moduli će se izgraditi u kod, ali je takav način izgradnje isključen jer za potrebe ovog projekta nisu potrebni drugi programski jezici. Unutar skripte se nalaze još dvije opcije CMAKE_C_FLAGS i CMAKE_CXX_FLAGS koji su zapravo varijable okoline[7] s dodatnim opcijama za C i C++ prevoditelj i koje su u ovom slučaju identično postavljene za obje vrste prevoditelja i koje se odnose strogo na koji način će se OpenCV izgraditi za Emscripten.

```
1.  -D CMAKE_C_FLAGS="-s WASM=1 -s USE_PTHREADS=0 . . ."
2.  -D CMAKE_CXX_FLAGS="-s WASM=1 -s USE_PTHREADS=0 . . ."
```

Slika 28: Dodatne opcije za generiranje wasm

[7] Varijabla okoline (eng. *environment variable*) – varijabla u trenutnoj okolini nekog procesa na računalu koja može utjecati na taj proces.

Datoteka build-unix.sh se znatno razlikuje od prethodne po opcijama koje se koriste za izgradnju sourcea. U početnom dijelu se na isti način kao i kod prethodne skripte generiraju direktoriji potrebni za OpenCV, ali je u ovom slučaju naziv direktorija za biblioteku lib-Unix. Što se tiče OpenCV modula, za razliku od prethodne skripte, u ovom slučaju se koristi nekoliko njih više koji su prikazani u nastavku na slici 29.

```
1.        -D BUILD_opencv_core=ON \
2.        -D BUILD_opencv_imgproc=ON \
3.        -D BUILD_opencv_imgcodecs=ON \
4.        -D BUILD_opencv_highgui=ON \
5.        -D BUILD_opencv_apps=OFF \
6.        -D BUILD_opencv_calib3d=OFF \
7.        -D BUILD_opencv_dnn=OFF \
8.        -D BUILD_opencv_features2d=ON \
9.        -D BUILD_opencv_flann=ON \
10.       -D BUILD_opencv_gapi=OFF \
11.       -D BUILD_opencv_ml=ON \
12.       -D BUILD_opencv_photo=OFF \
13.       -D BUILD_opencv_shape=OFF \
14.       -D BUILD_opencv_videoio=ON \
15.       -D BUILD_opencv_videostab=OFF \
16.       -D BUILD_opencv_superres=OFF \
17.       -D BUILD_opencv_stitching=OFF \
18.       . . .
```

Slika 29: OpenCV moduli za desktop okolinu

Može se primijetiti da se sada koristi znatno više modula od kojih su core, imgproc, imgcodecs, highgui, flann, videoio i ml, te nema CMAKE_C_FLAGS i CMAKE_CXX_FLAGS kao u prethodnoj skripti. Razlog tome je što ova skripta služi za generiranje sourcea za Unix razvojnu okolinu, koja na drugačiji način prikazuje obradu slika, nego što je to u slučaju WebAssembly na web stranici. Na web stranici se sve u vezi slika prikazuje na HTML Canvasu, dok na desktop okolini se obrada slika prikazuje u zasebnom prozoru, zbog čega je i potrebno uključiti dodatne module. Postavlja se pitanje zašto uopće trebamo dva različita OpenCV sourcea, jedan za Emscripten za generiranje WebAssembly, drugi OpenCV source za desktop razvojnu okolinu? Razlog je prilično jednostavan, jer na desktop razvojnoj okolini možemo testirati radi li implementacija biblioteke kako treba, te kad je rezultat željen, onda se koristi OpenCV kod podešen za Emscripten za generiranje WebAssembly modula koji će raditi na isti način kako i radi na desktop okruženju, s gotovo istim performansama.

CMakeLists.txt unutar glavnog direktorija je polazišna točka generiranja izvorišnog koda cijelog projekta, što znači da sadrži sve definicije pravila na koji način će se kod generirati. Ona definira implementaciju DocumentExtractor biblioteke i predstavlja sustav izgradnje za cijeli projekt. Preko CMakeLists.txt datoteke unutar glavnog

direktorija povezani su i ostali pod direktoriji projekta ali i putanje koje pokazuju gdje se nalazi izvorišni kod od drugih biblioteka kao OpenCV i Emscripten, te datoteke .h[8] i .cc[9] koje trebaju biti uključene i vidljive u bilo kojem dijelu projekta pomoću C/C++ #include[10] direktive. Na slici 30 je prikazana dio CMakeLists.txt datoteke glavnog direktorija projekta.

```
1.  cmake_minimum_required(VERSION 3.1.0)
2.  set(CMAKE_CXX_STANDARD 17)
3.  project (DocumentExtractor)
4.  set (SRCS
5.          DocumentExtractor.cc
6.  )
7.  set (HDRS
8.          DocumentExtractor.h
9.  )
10. if(EMSCRIPTEN)
11.     set(OpenCV_DIR ${CMAKE_SOURCE_DIR}/lib-emscripten/opencv/lib/cmake/opencv4)
12. else(EMSCRIPTEN)
13.     set(OpenCV_DIR ${CMAKE_SOURCE_DIR}/lib-macos/opencv/lib/cmake/opencv4)
14. endif(EMSCRIPTEN)
15.
16. find_package(OpenCV REQUIRED)
17.
18. include_directories(${OpenCV_INCLUDE_DIRS})
19.
20. set (LIBS ${OpenCV_LIBS})
21.
22. if(EMSCRIPTEN)
23.     include(${CMAKE_SOURCE_DIR}/emscripten/CMakeLists.txt)
24. else(EMSCRIPTEN)
25.     add_definitions(-DHAVE_THREADS)
26.     include(${CMAKE_SOURCE_DIR}/linux/CMakeLists.txt)
27. endif(EMSCRIPTEN)
28.
29. ADD_EXECUTABLE(
30.         DocumentExtractor
31.         ${EXTRA_DEPENDENCIES} ${SRCS} ${HDRS}
32. )
```

Slika 30: Primjer CMakeLists.txt unutar glavnog direktorija

Datoteka započinje definiranjem minimalne verzije cmake pomoću komande cmake_minimum_required koja je u konkretnom slučaju 3.1.0. što znači da CMakeLists.txt radi na svim verzijama većim od definirane u datoteci uključujući. U nastavku su opisane neke od ključnih komandi:

- **set(CMAKE_CXX_STANDARD 17)** – definira se C++ standard koji će C++ kod projekta koristiti.

[8] .h – ekstenzija za C++ header datoteku
[9] .cc – ekstenzija za C++ source datoteku
[10] #include – način uključivanja standardne ili korisnički definirane datoteke u program uglavnom na početku bilo kojeg C/C++ programa.

- **project(DocumentExtractor)** postavlja naziv projekta i sprema ga u varijablu **PROJECT_NAME**.
- **set (SRCS DocumentExtractor.cc)** – postavlja izvorišni kod datoteke vidljive unutar projekta, trenutno je jedna datoteka ali ih može biti više.
- **set (HDRS DocumentExtractor.h)** – postavlja zaglavlje datoteke vidljive unutar projekta, kojih također može biti više.
- **set(OpenCV_DIR path-to-opencv)** – postavlja OpenCV put do direktorija, koji u konkretnom slučaju ovisi da li se konfiguracija odnosi na Emscripten ili Unix.
- **find_package(OpenCV REQUIRED)** – pronalazi sve pakete, odnosno OpenCV module, koji se odnose na prethodno postavljeni put do direktorija i obavezni su.
- **include_directories(${OpenCV_INCLUDE_DIRS})** – uključuje direktorije OpenCV modula zajedno s onim direktorijima koje prevoditelj koristi za pretragu.
- **set (LIBS ${OpenCV_LIBS})** – postavlja OpenCV biblioteke u environment varijablu LIBS.
- **include(${CMAKE_SOURCE_DIR}/emscripten/CMakeLists.txt)** – pored konfiguracije glavnog projekta, uključuje i CMakeLists.txt konfiguraciju iz pod direktorija ili nekog pod modula. U konkretnom slučaju, ako se radi o Emscripten, čita datoteku iz direktorija, inače iz Linux direktorija.
- **ADD_EXECUTABLE(DocumentExtractor ${EXTRA_DEPENDENCIES} ${SRCS} ${HDRS})** – dodaje izvršni cilj (eng. *target*) koji se gradi iz izvornih datoteka navedenih u pozivanju naredbe. U ovom slučaju to su sve .h i .cc datoteke prethodno definirane.

Datoteka DocumentExtractor.h kao i sve druge datoteke s ekstenzijom .h sadrže deklaracije C funkcija i makro definicije koje se mogu dijeliti između nekoliko izvornih datoteka. Postoje dvije vrste datoteka zaglavlja: datoteke koje programer piše i datoteke koje dolaze s prevoditeljem. Uključivanje datoteke .h jednako je kopiranju sadržaja datoteke zaglavlja, ali se to ne čini jer će biti podložno pogreškama i nije dobra praksa kopirati sadržaj datoteke .h u izvorne datoteke, pogotovo ako imati više izvornih datoteka u programu. Jednostavna praksa u C ili C++ programima je da čuvamo sve konstante, makro naredbe, globalne varijable cijelog sustava i prototipove funkcija u datotekama .h i uključujemo tu datoteku gdje god je potrebna. U ovom

projektu, u glavnom direktoriju se koristi DocumentExtractor.h datoteka na kojoj je vidljivo da je definirana struktura[11] nazvana MrzZoneStruct koja predstavlja definiciju detektirane strojno čitljive zone. Sastoji se od dva konstruktora od kojih je jedan glavni, što znači da su članovi strukture inicijalizirani prilikom samog poziva. Drugi tip konstruktora za parametre uzima 4 vrijednosti tipa int, te se pri pozivu ovog tipa struktura inicijalizira prosljeđivanjem parametara konstruktoru. Članovi strukture se sastoje od x i y što predstavlja koordinate gdje se nalazi detektirana zona dok width i heigth predstavlja širinu i visinu detektirane strojno čitljive zone kao što je vidljivo na slici 31.

```
1.  #include <string>
2.  #include <opencv2/core/mat.hpp>
3.
4.  struct MrzZoneStruct {
5.    MrzZoneStruct() {
6.      this->x = 0;
7.      this->y = 0;
8.      this->width = 0;
9.      this->height = 0;
10.   }
11.
12. MrzZoneStruct(int x, int y, int width, int height) {
13.     this->x = x;
14.     this->y = y;
15.     this->width = width;
16.     this->height = height;
17.   }
18.
19.   int x;
20.   int y;
21.   int width;
22.   int height;
23. };
24.
25. class DocumentExtractor {
26. public:
27.   DocumentExtractor() = default;
28.   ~DocumentExtractor() = default;
29.
30.   cv::Mat getImage(uint8_t * position, int width, int height);
31.   MrzZoneStruct *detectMrz(uint8_t * position, int width, int height);
32.
33.   [[nodiscard]] const MrzZoneStruct &getDetectedZone() const;
34.   void setDetectedZone(const MrzZoneStruct &detectedZone);
35.
36. private:
37.   MrzZoneStruct detectedZone;
38.   bool detect(const cv::Mat &original);
39. };
```

Slika 31: Primjer DocumentExtractor.h datoteke

[11] struct – korisnički definirana vrsta podataka, odnosno kombinacija podataka različitih tipova podataka pod jednim imenom.

Pored strukture MrzZoneStruct, nalazi se i definicija klase DocumentExtractor koja u svom tijelu ima glavni konstruktor i destruktor.
Sastoji se od slijedećih javnih funkcija:

- **cv::Mat getImage(uint8_t * position, int width, int height)** – koristi se samo u svrhu testiranja na slikama i trenutno nema poveznice s implementacijom detekcije.
- **[[nodiscard]] const MrzZoneStruct &getDetectedZone() const** i **void setDetectedZone(const MrzZoneStruct &detectedZone)** – klasični getteri i setteri koji služe za dohvaćanje i postavljanje vrijednosti MrzZoneStruct.
- **MrzZoneStruct *detectMrz(uint8_t * position, int width, int height)** – detektira strojno čitljivu zonu tako što za ulazne parametre ima pokazivač na uint8_t[12] poziciju detektirane zone, širinu i visinu zone, te vraća na izlazu detektiranu zonu tipa MrzZoneStruct.

Kao što se vidi iz priloženog .h datoteka sadrži samo definiciju biblioteke DocumentExtractor, koja je u konkretnom slučaju jedna klasa, odnosno objekt. Implementacija DocumentExtractor.h vrši se u datoteci DocumentExtractor.cc, što bi u praksi otprilike značilo da svaka .h datoteka ima svoju datoteku implementacije .cc, ali ne mora nužno biti tako. Moguće je da jedan dio implementacije bude unutar .h datoteke, ali u ovom slučaju svaka .h datoteka će isključivo imati .cc datoteku implementacije. Svaka .cc datoteka počinje s umetanjem pripadajuće .h datoteke i ostalih zaglavlja datoteka koji pripadaju drugim bibliotekama.

5.2 Struktura i okruženje projekta Emscripten

Projekt Emscripten je zapravo poddirektorij glavnog projekta DocumentExtractor, te kao i glavni dio projekta sadrži CMakeLists.txt datoteku koja se odnosi na funkcionalnost i način na koji će se kod glavnog projekta, odnosno biblioteke izgraditi. Na isti način se moglo definirati sh skripta pomoću koje bi se pokrenula izgradnja koda za wasm, zajedno s Emscripten alatnim lancem, ali ovaj put je korišten cmake-gui, grafičko korisničko sučelje za konfiguriranje i generiranje koda iz CMakeLists.txt, iz razloga jer je bilo jednostavnije nego kreirati sh skriptu. Nekoliko je postavki je vrlo

[12] Neoznačeni (eng. *unsigned*) cjelobrojni tip s širinom od točno 8 bita pod uvjetom ako i samo ako implementacija izravno podržava tip

važno za generiranje wasm datoteke pomoću Emscriptena, a neke od njih su prikazane na slici 32.

```
1.  set(IDLS ${CMAKE_SOURCE_DIR}/emscripten/interfaces.idl)
2.  set(STU emscripten/glue_stub.cc )
3.  set(GLUE_COMMAND python)
4.  set(GLUE_SCRIPT /usr/local/Cellar/emscripten/2.0.25/libexec/tools/webidl_binder.py)
```

Slika 32: Postavke u CMakeLists.txt od Emscripten projekta

Iz priloženog se vidi da je postavljeno nekoliko varijabli koje su neophodne za generiranje wasm datoteke, a to su IDLS, varijabla koja pokazuje na datoteku interfaces.idl, STU pokazuje na datoteku glue_stub.cc u kojoj su uključene datoteke Extractor.h i interfaces_glue.cpp, GLUE_COMMAND kojom se definira jezik, u ovom slučaju python, kojim se izvršava komanda za generiranje koda i GLUE_SCRIPT, putanja do webidl_binder.py skripte koja služi za generiranje, odnosno povezivanje C++ i JavaScript koda preko Web IDL interfacea.

Jedna od glavnih razlika ovog direktorija u odnosu na ostale je ta što sadrži jednu dodatnu datoteku interfaces.idl u kojoj je definiran interface, koji zapravo predstavlja neku vrstu mosta između C++ i JavaScript koda.

Web IDL je standard koji definira jezik definicije sučelja (eng. *interface definition language*), koji se može koristiti za opisivanje sučelja koja su namijenjena za implementaciju u web preglednicima (Web IDL, 2022) (vlastiti prijevod). Web IDL je varijanta IDL-a s brojnim značajkama koje omogućuju lakše specificiranje ponašanja uobičajenih skriptnih objekata na web platformi i pruža sintaksu za određivanje površinskih API-ja objekata web platforme, kao i ECMAScript vezivanja koja detaljno opisuju kako se neki API manifestira kao ECMAScript konstrukcija (Web IDL, 2022) (vlastiti prijevod).

To osigurava da uobičajeni zadaci, kao što su instaliranje globalnih svojstava, obrada numeričkih ulaza ili izlaganje ponašanja iteracije, ostaju ujednačeni u specifikacijama web platforme, koja opisuju svoja sučelja pomoću Web IDL-a, a zatim koriste prozu za specificiranje pojedinosti specifičnih za API (Web IDL, 2022) (vlastiti prijevod). Primjer interfaces.idl datoteke je vidljiv na slici 33.

```
1.  interface MrzZoneStruct {
2.    void MrzZoneStruct(long x, long y, long width, long height);
3.    attribute long x;
4.    attribute long y;
5.    attribute long width;
6.    attribute long height;
```

```
};

interface Extractor {
  void Extractor();
  [Const] MrzZoneStruct detectMrzZone(octet position, long width, long height);
};
```

Slika 33: Datoteka interfaces.idl

Vidljivo je da interface MrzZoneStruct predstavlja opis strukture iste kao u C++ dijelu koda gdje uz članove strukture stoji oznaka attribute, i svi članovi su tipa long, dok su u C++ svi članovi strukture tipa int. Nakon strukture, definiran je opis Extractor objekta iz C++ dijela Emscripten projekta, koja predstavlja omotač (eng. *wrapper*) oko glavne biblioteke DocumentExtractor. Važno je napomenuti da je neophodno prije svake upotrebe tipova objekata u idl datoteci definirati taj tip, zatim ga koristiti, kao što je vidljivo na slici 33, prvo je definiran interface MrzZoneStruct a zatim se taj definirani tip koristi unutar Extractor interfacea.

Nekoliko glavnih postavki koje su prikazane na slici 34, odnose se na sam način generiranja wasm datoteke, odnosno čine postavke Emscripten prevoditelja.

```
SET(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} -std=C++17 --bind \
-s ASSERTIONS=1 \
-s DISABLE_EXCEPTION_CATCHING=0 \
-s SINGLE_FILE=1 \
-s WASM=1 \
-s ALLOW_MEMORY_GROWTH=1 \
-s NO_FILESYSTEM=1 \
-s ELIMINATE_DUPLICATE_FUNCTIONS=1 \
-s NO_EXIT_RUNTIME=1 \
-O3 --post-js ${glue_files}")
```

Slika 34: Datoteka interfaces.idl

ASSERTIONS=1 koristi se za omogućavanje provjera vremena izvođenja za uobičajene pogreške u dodjeli memorije (npr. upisivanje više memorije nego što je dodijeljeno). Također definira kako bi Emscripten trebao postupati s pogreškama u tijeku programa. Vrijednost se može postaviti na **ASSERTIONS=2** kako bi se izveli dodatni testovi (Emscripten Contributors, 2015) (vlastiti prijevod).

DISABLE_EXCEPTION_CATCHING=0 služi za isključivanje dohvaćanja iznimki kod pogrešaka. Razlog tome je brže izvođenje i znatno manja veličina wasm datoteke. Iako je ova opcija isključena, to ne znači da se iznimke nisu desile, samo nisu uhvaćene.

SINGLE_FILE=1 služi za spajanje JavaScript koda i wasm u jednu datoteku. Ako je postavljeno na 0, datoteka wasm će se generirati odvojeno od JavaScripta, gdje će zatim biti potrebno u implementaciji, uključiti wasm datoteku.

WASM=1 označava da se generira wasm datoteka, ova opcija može biti isključena.
ALLOW_MEMORY_GROWTH=1 omogućuje promjenu ukupne količine korištene memorije ovisno o zahtjevima aplikacije. Ovo je korisno za aplikacije koje ne znaju unaprijed koliko će im memorije biti potrebno (Emscripten Contributors, 2015) (vlastiti prijevod).
NO_FILESYSTEM=1 se koristi za isključivanje **File System API** biblioteke koja služi za učitavanje datoteka s diska i koja trenutno nije potrebna.
ELIMINATE_DUPLICATE_FUNCTIONS=1 eliminira duple funkcije i dio koda koji je deklariran ali se nigdje ne koristi.
NO_EXIT_RUNTIME=1 prema zadanim postavkama Emscripten postavlja **EXIT_RUNTIME=0**, što znači da se ne uključuje kod za isključivanje vremena izvođenja. To znači da kada main() izvrši, ne poziva se destruktori globalnih C++ objekata. To omogućuje emitiranje manjeg koda prema zadanim postavkama, i obično je ono što se želi na webu, iako se main() izvršio, možda će se kasnije dogoditi nešto asinkrono što će se izvršiti. U konkretnom slučaju ova opcija je uključena.
Nakon što se pozove webidl_binder.py skripta, generiraju se uvezi (eng. *bindings*), odnosno povezuje se JavaScript, wasm i C++, te se stvaraju 3 datoteke:

- **DocumentExtractor.js** – minimizirani wasm i JavaScript kod u jednoj datoteci.
- **interfaces_glue.cpp** – datoteka u kojoj su generirani svi uvezi između Emscripten i Extractor C++ koda.
- **interfaces_glue.js** – reprezentacija prethodne datoteke u JavaScriptu. Ova datoteka nije potrebna ali je generirana čisto iz razloga kako bi se vidjele na koji način su generirane funkcije unutar wasm modulate služi samo za provjeru ispravnosti generiranog koda. Kod unutar ove datoteke postoji i u DocumentExtractor.js ali je minimiziran i nije čitljiv.

Kako je već ranije spomenuto, datoteka Extractor.cc predstavlja omotač oko glavne biblioteke DocumentExtractor. Razlog tome je da Extractor objekt čahuri DocumentExtractor tako što se kreira unutar konstruktora kao novi objekt koristeći pametni pokazivač (eng. *smart pointer*) Extractor::Extractor() :m_extractor(new DocumentExtractor), gdje je m_extractor definiran kao pametni pokazivač std::unique_ptr<DocumentExtractor>. Svrha ovog pametnog pokazivača je da nakon što objekt, u ovom slučaju DocumentExtractor, izleti van opsega (eng. *scope*),

automatski pozove destruktor, i time se ne mora voditi računa o brisanju objekta. Na slici 35 prikazana je implementacija objekta Extractor.

```
1.  #include "Extractor.h"
2.
3.  Extractor::Extractor()
4.  : m_extractor(new DocumentExtractor)
5.  {}
6.
7.  Extractor::~Extractor() = default;
8.
9.  const MrzZoneStruct* Extractor::detectMrzZone(unsigned char* position, int width,int
    height){
10.   static MrzZoneStruct* output;
11.   output = m_extractor->detectMrz(position, width, height);
12.   return output;
13. }
```

Slika 35: Sadržaj datoteke Extractor.cc

Sa slike je vidljivo da je implementacija funkcije const MrzZoneStruct* Extractor::detectMrzZone(unsigned char* position, int width,int height) slična kao i funkcija detectMrz od DocumentExtractor objekta, gdje su im ulazni i izlazni parametri isti, te radi preglednosti koda omotava funkciju detectMrz. Moguće je i direktno uvezati sa DocumentExtractor objektom, ali time je mogućnost pogreške veća a i samim time kod je manje čitljiv.

Nakon pokretanja webidl_binder.py, unutar interfaces_glue.cpp datoteke Emscripten okvir automatski generira C++, odnosno C kod koji sadrži i neke Emscripten značajke. Sadržaj interfaces_glue.cpp datoteke prikazan je na slici 36.

```
1.  #include <emscripten.h>
2.
3.  extern "C" {
4.
5.  // Not using size_t for array indices as the values used by the JavaScript code are
    signed.
6.
7.  EM_JS(void, array_bounds_check_error, (size_t idx, size_t size), {
8.    throw 'Array index ' + idx + ' out of bounds: [0,' + size + ')';
9.  });
10.
11. void array_bounds_check(const int array_size, const int array_idx) {
12.   if (array_idx < 0 || array_idx >= array_size) {
13.     array_bounds_check_error(array_idx, array_size);
14.   }
15. }
16.
17. // VoidPtr
18.
19. void EMSCRIPTEN_KEEPALIVE emscripten_bind_VoidPtr___destroy___0(void** self) {
20.   delete self;
21. }
22.
23. // MrzZoneStruct
```

```
24.
25. MrzZoneStruct* EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_MrzZoneStruct_4(int x,
    int y, int width, int height) {
26.   return new MrzZoneStruct(x, y, width, height);
27. }
28.
29. int EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_get_x_0(MrzZoneStruct* self) {
30.   return self->x;
31. }
32.
33. void EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_set_x_1(MrzZoneStruct* self, int
    arg0) {
34.   self->x = arg0;
35. }
36.
37. int EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_get_y_0(MrzZoneStruct* self) {
38.   return self->y;
39. }
40.
41. void EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_set_y_1(MrzZoneStruct* self, int
    arg0) {
42.   self->y = arg0;
43. }
44.
45. int EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_get_width_0(MrzZoneStruct* self) {
46.   return self->width;
47. }
48.
49. void EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_set_width_1(MrzZoneStruct* self,
    int arg0) {
50.   self->width = arg0;
51. }
52.
53. int EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_get_height_0(MrzZoneStruct* self)
    {
54.   return self->height;
55. }
56.
57. void EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct_set_height_1(MrzZoneStruct* self,
    int arg0) {
58.   self->height = arg0;
59. }
60.
61. void EMSCRIPTEN_KEEPALIVE emscripten_bind_MrzZoneStruct___destroy___0(MrzZoneStruct*
    self) {
62.   delete self;
63. }
64.
65. Extractor* EMSCRIPTEN_KEEPALIVE emscripten_bind_Extractor_Extractor_0() {
66.   return new Extractor();
67. }
68.
69. const MrzZoneStruct* EMSCRIPTEN_KEEPALIVE
    emscripten_bind_Extractor_detectMrzZone_3(Extractor* self, unsigned char* position, int
    width, int height) {
70.   return self->detectMrzZone(position, width, height);
71. }
72.
73. void EMSCRIPTEN_KEEPALIVE emscripten_bind_Extractor___destroy___0(Extractor* self) {
74.   delete self;
75. }
76. }
```

Slika 36: Sadržaj datoteke interfaces_glue.cpp

Osim što je vidljivo da kod generiranja naziva funkcija i objekata, pored imena objekta ili funkcije svugdje je automatski generirano s prefiksom emscripten_bind_, zatim ime

Objekta, npr. MrzZoneStruct, nakon toga naziv funkcije, kao na npr. Emscripten_bind_MrzZoneStruct_get_x_0. U nastavku se opisuje neke od glavnih značajki:

- **extern "C"** – sav kod u datoteci se nalazi unutar extern "C" opsega. **extern "C"** čini da naziv funkcije u C++ ima C vezu tako da prevoditelj ne mijenja ime gdje klijentski C kod može koristiti funkciju koristeći C kompatibilnu datoteku zaglavlja koja sadrži samo deklaraciju funkcije.
- **EM_JS** – omogućuje da se deklarira JavaScript u C kodu kao funkciju koja se može pozvati kao normalna C funkcija.
- **EMSCRIPTEN_KEEPALIVE** – ako se funkcija koristi u drugim funkcijama, LLVM je može umetnuti i neće se pojaviti kao jedinstvena funkcija u JavaScriptu. Ukrašavanje koda pomoću EMSCRIPTEN_KEEPALIVE može biti korisno ako se ne želi pratiti funkcije za eksplicitni izvoz i kada se ti izvozi ne mijenjaju.

Datoteka interfaces_glue.js služi samo za provjeru i pregled koda nakon generiranja, te ju je moguće isključiti u CMake postavkama za izgradnju. interfaces_glue.js datoteka je čista refleksija, odnosno odraz interfaces_glue.cpp datoteke. U datoteci, osim što je generiran kod koji se uvezuje, nalazi se i dosta koda koji Emscripten generira. Na slici 37 prikazan je samo dio koda interfaces_glue.js datoteke.

```
1.  . . .
2.  MrzZoneStruct.prototype = Object.create(WrapperObject.prototype);
3.  MrzZoneStruct.prototype.constructor = MrzZoneStruct;
4.  MrzZoneStruct.prototype.__class__ = MrzZoneStruct;
5.  MrzZoneStruct.__cache__ = {};
6.  Module['MrzZoneStruct'] = MrzZoneStruct;
7.
8.    MrzZoneStruct.prototype['get_x'] = MrzZoneStruct.prototype.get_x = /** @suppress
    {undefinedVars, duplicate} @this{Object} */function() {
9.    var self = this.ptr;
10.   return _emscripten_bind_MrzZoneStruct_get_x_0(self);
11. };
12.     MrzZoneStruct.prototype['set_x'] = MrzZoneStruct.prototype.set_x = /** @suppress
    {undefinedVars, duplicate} @this{Object} */function(arg0) {
13.   var self = this.ptr;
14.   if (arg0 && typeof arg0 === 'object') arg0 = arg0.ptr;
15.   _emscripten_bind_MrzZoneStruct_set_x_1(self, arg0);
16. };
17. . . .
```

Slika 37: Dio datoteke interfaces_glue.js

5.3 Implementacija detekcije

Implementacijska datoteka DocumentExtractor.cc sadrži implementaciju svih funkcija unutar .h datoteke, te funkcionalnost getter funkcije dohvaća detektiranu strojno čitljivu zonu koja je tipa MrzZoneStruct, dok setter funkcija postavlja detektiranu strojno čitljivu zonu tipa MrzZoneStruct, što je i vidljivo na slici 38.

```
1.  void DocumentExtractor::setDetectedZone(const MrzZoneStruct &detectedZone) {
2.    DocumentExtractor::detectedZone = detectedZone;
3.  }
4.
5.  const MrzZoneStruct &DocumentExtractor::getDetectedZone() const {
6.    return detectedZone;
7.  }
```

Slika 38: Implementacija getter i setter funkcija

Osim getter i setter funkcija, tu je i funckija bool detect(const cv::Mat &original) koja je privatna, što znači da je upotrebljiva samo unutar DocumentExtractor objekta i nije dostupna van tog objekta i koristi se unutar funkcije MrzZoneStruct *detectMrz(uint8_t * position, int width, int height) koja je javna, što znači da ju je moguće pozvati bilo gdje na DocumentExtractor instanci objekta. Proces detekcije strojno čitljive zone se sastoji od nekoliko faza obrade slike prije nego se dobije detektirana strojno čitljiva zona. Važno je napomenuti da se kod svih faza obrade slike sama obrada vrši na kopiji slike zato što se uopće ne želi mijenjati originalna slika, osim kod iscrtavanja strojno čitljive zone na slici, koja ovisno o potrebama može biti nacrtana na kopiji ili originalu.

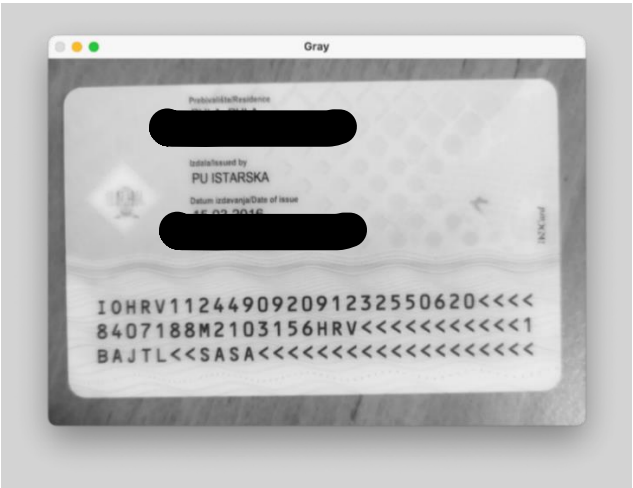
1. korak je promjena veličine slike i pretvaranje slike u sivu boju, odnosno od originalne slike se uzima samo sivi kanal boja (eng. *gray channel*).

```
1.  bool DocumentExtractor::detect(const cv::Mat &original) {
2.    // initialize a rectangular and square structuring kernel
3.    cv::Mat rectKernel = getStructuringElement(cv::MORPH_RECT, cv::Size(13, 5));
4.    cv::Mat sqKernel = getStructuringElement(cv::MORPH_RECT, cv::Size(21, 21));
5.     // resize the image and convert it to grayscale
6.    cv::Mat image;
7.    int scale_percent = 50;
8.    auto width = int(original.size().width * scale_percent / 100);
9.    auto height = int(original.size().height * scale_percent / 100);
10.   resize(original, image, cv::Size(width, height));
11.   cv::Mat gray;
12. #if defined EMSCRIPTEN == 1
13.   cv::cvtColor(image, gray, cv::COLOR_RGBA2GRAY);
14.   cv::cvtColor(gray, image, cv::COLOR_GRAY2RGBA);
15. #else
16.   cv::cvtColor(image, gray, cv::COLOR_RGBA2GRAY);
17. #endif
```

Slika 39: Prikaz implementacije promjene veličine i pretvaranje u sivi kanal slike

Sa slike 39 je vidljivo da se za preračunavanje novih dimenzija slike za širinu uzima širina originalne slike x postotak skaliranja / 100, slično i za visinu visina originalne slike x postotak skaliranja / 100, zatim se funkcijom resize(original, image, cv::Size(width, height)) postavlja nova veličina slike na novi objekt slike, odnosno multi dimenzionalne matrice tipa cv::Mat s objektom cv::Size. Objekt cv::Mat predstavlja multi – dimenzionalni gusti numerički jednokanalni ili višekanalni niz. Može se koristiti za pohranjivanje vektora i matrica realnih ili kompleksnih vrijednosti, slika u nijansama sive ili u boji, vektorskih polja, oblaka točaka, tenzora, histograma i slično (OpenCV, 2022) (vlastiti prijevod).

Objekt cv::Size predstavlja objekt predloška za određivanje veličine slike ili pravokutnika, uključuje dva člana koja se nazivaju širina i visina (OpenCV, 2022) (vlastiti prijevod). Veličina slike se mijenja jer se mora dovesti do odgovarajuće veličine kako bi detekcija strojno čitljive zone bila uspješna. Prevelika slika nije dobra zato što se može desiti da rezolucija slike može biti veća od rezolucije ekrana, recimo mobilnog uređaja, čime je moguće prouzrokovati neispravan rad programa. Funkcijom resize(original, image, cv::Size(width, height)) se postavlja nova preračunata veličina slike i sprema u varijablu image. U slučaju kad se radi o Emscripten buildu, tada se radi malo drukčija konverzija slike u sivu skalu, gdje se funkcijom cv::cvtColor(image, gray, cv::COLOR_RGBA2GRAY) prvo vrši konverzija RGB skale u sivu, zatim se funkcijom cv::cvtColor(gray, image, cv::COLOR_GRAY2RGBA) vraća iz sivog u RGB u izvorni kanal i sprema u varijablu image, čime se dobije ispravan prikaz sivog kanala. To je slučaj samo kad se radi o Emscriptenu, odnosno wasm. Inače za desktop verziju se poziva samo jednom konverzija funkcijom cv::cvtColor(image, gray, cv::COLOR_RGBA2GRAY). Rezultat je vidljiv na slici 40.



Slika 40: Promjena veličine i pretvaranje slike u sivu boju

2. korak je da se na sivoj slici napravi zamagljivanje slike pomoću Gaussovog filtera 3x3 sa funkcijom cv::GaussianBlur(gray, gray, cv::Size(3, 3), 0).

U operaciji Gaussova zamagljivanja, slika se savija s Gaussovim filterom umjesto okvirnog filtera. Gaussov filtar je nisko propusni filter koji uklanja visokofrekventne komponente. Gaussovo zamagljivanje slično je prosječnom zamućenju, ali umjesto jednostavne srednje vrijednosti, koristi ponderiranu sredinu, gdje pikseli susjedstva koji su bliže središnjem pikselu pridonose većoj težini prosjeku, te je konačni rezultat da je slika manje zamagljena, ali više prirodno zamagljenija od prosječne metode, gdje za zamagljivanje koristi jezgru M x N, gdje su i M i N neparni cijeli brojevi (Rosebrock, 2021) (vlastiti prijevod). Rezultat je vidljiv na slici 41.

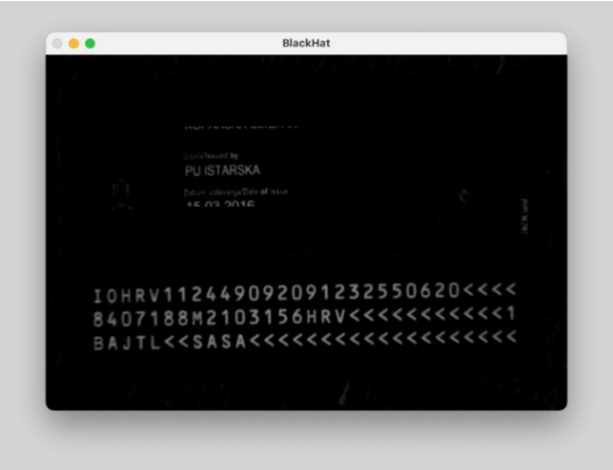
Slika 41: Zamagljivanje Gaussovim filterom

3. korak je korištenje morfološkog operatora cv::MORPH_BLACKHAT da bi se pronašla tamna područja na svijetloj pozadini sa funkcijom cv::morphologyEx(gray, blackhat, cv::MORPH_BLACKHAT, rectKernel) kako je vidljivo sa slike 42.

```
1.    cv::GaussianBlur(gray, gray, cv::Size(3, 3), 0);
2.    cv::Mat blackhat;
3.    cv::morphologyEx(gray, blackhat, cv::MORPH_BLACKHAT, rectKernel);
```

Slika 42: Zamagljivanje i pronalaženje tamnih područja na svijetloj pozadini (developer79433, 2016)

Funkcija cv::morphologyEx izvodi napredne morfološke transformacije koristeći eroziju i dilataciju kao osnovne operacije, te bilo koja od operacija može se obaviti na mjestu. U slučaju višekanalnih slika, svaki kanal se obrađuje zasebno (OpenCV, 2022) (vlastiti prijevod). Rezultat je vidljiv na slici 43.



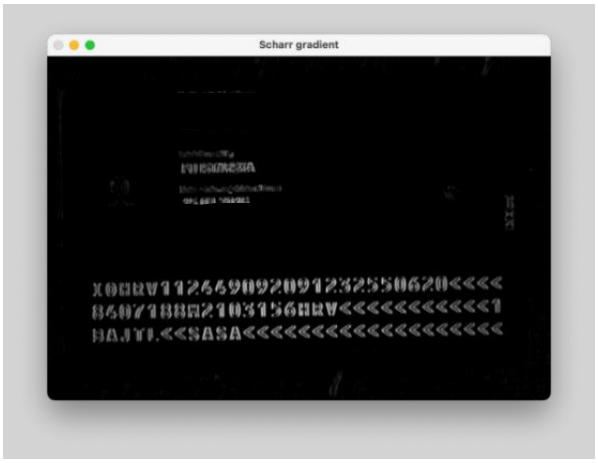
Slika 43: Rezultat nakon korištenja morfološkog operatora

4. korak je izračunavanje Scharrovog gradijenta slike blackhat i skaliranje rezultata u rasponu od 0 do 255 kako je vidljivo na slici 44.

```
1.    cv::Mat gradX;
2.    Sobel(blackhat, gradX, CV_32F, 1, 0, -1);
3.    gradX = abs(gradX);
4.    double minVal, maxVal;
5.    minMaxIdx(gradX, &minVal, &maxVal);
6.    cv::Mat gradXfloat = (255 * ((gradX - minVal) / (maxVal - minVal)));
7.    gradXfloat.convertTo(gradX, CV_8UC1);
```

Slika 44: Izračunavanje Scharrovog gradijenta slike blackhat i skaliranje (developer79433, 2016)

Sobel operator se koristi u obradi slike i računalnog vida, osobito unutar algoritama za detekciju rubova (eng. *edge detection*) gdje stvara sliku koja naglašava rubove. Scharr je proces optimizacije tako što je proizvodi jezgre optimizirane za određenu numeričku preciznost, kao npr. cijeli broj, float i dimenzionalnosti, kao npr. 1D, 2D, 3D (Wikipedia, 2022) (vlastiti prijevod). Rezultat operacija je vidljiv na slici 45.



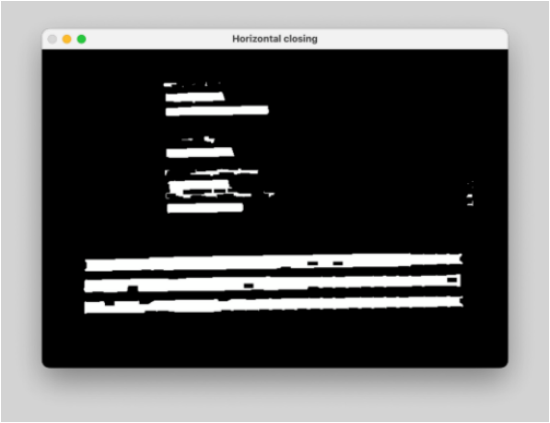
Slika 45: Detekcija rubova Sobel i Scharr

5. korak je primjena operacija zatvaranja pomoću pravokutne jezgre da se zatvore praznine između slova, te se zatim koristi Otsuovu metoda graničnih vrijednosti prikazana na slici 46.

```
1.  cv::morphologyEx(gradX, gradX, cv::MORPH_CLOSE, rectKernel);
2.  cv::Mat thresh;
3.  cv::threshold(gradX, thresh, 0, 255, cv::THRESH_BINARY | cv::THRESH_OTSU);
```

Slika 46: Zatvaranje praznina među znakovima – Otsu binarizacija (developer79433, 2016)

U globalnom određivanju praga koristi se proizvoljno odabrana vrijednost kao prag. Nasuprot tome, Otsuova metoda izbjegava odabir vrijednosti i određuje je automatski. Za to se koristi funkcija cv::threshold(gradX, thresh, 0, 255, cv::THRESH_BINARY | cv::THRESH_OTSU), gdje se THRESH_OTSU prosljeđuje kao dodatna zastavica. Vrijednost praga može se odabrati proizvoljno, te algoritam tada pronalazi optimalnu vrijednost praga koja se vraća kao prvi izlaz. Ovaj proces se naziva još i Otsu binarizacija i kao rezultat se dobije horizontalno zatvaranje praznina kako je vidljivo na slici 47.



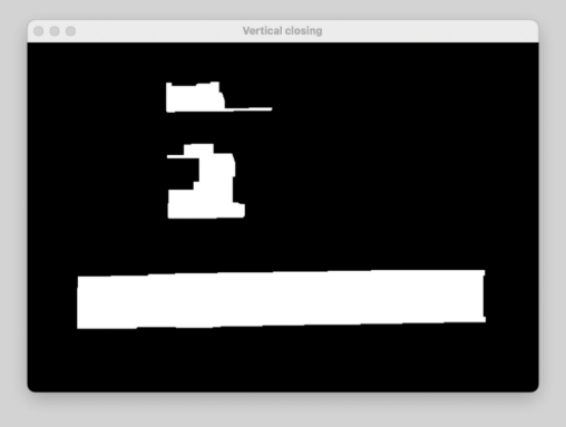
Slika 47: Horizontalno zatvaranje praznina

6. korak je izvođenje još jedne operacije zatvaranja, ovaj put koristeći četvrtastu jezgru za zatvaranje praznina između linija strojno čitljive zone, zatim se izvede niz erozija (eng. *erodes*) kako bi se razdvojile povezane komponente, što prikazuje slika 48.

```
1.  cv::morphologyEx(thresh, thresh, cv::MORPH_CLOSE, sqKernel);
2.  cv::Mat nullKernel;
3.  cv::erode(thresh, thresh, nullKernel, cv::Point(-1, -1), 4);
```

Slika 48: Operacija zatvaranja četvrtastom jezgrom (developer79433, 2016)

Funkcija cv::erode(thresh, thresh, nullKernel, cv::Point(-1, -1), 4) erodira, odnosno nagriza izvornu sliku koristeći specificirani strukturni element koji određuje oblik susjedstva piksela preko kojeg se uzima minimum. Za rezultat se dobije vertikalno zatvaranje praznina između redova znakova kako je prikazano na slici 49.



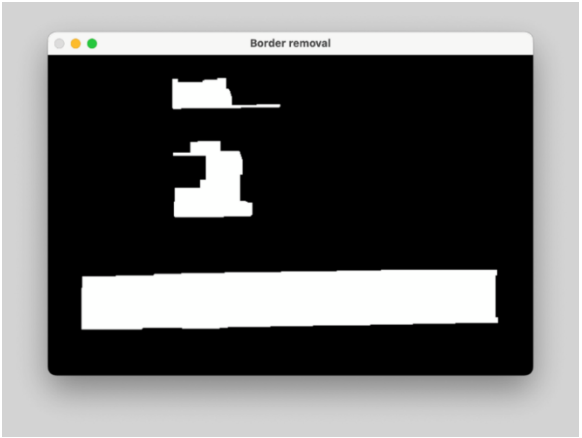
Slika 49: Vertikalno zatvaranje praznina

7. korak je postavljanje 5% od rubova slike na nulu, jer je moguće da su se tijekom postavljanja praga, granični pikseli uključili u prag vidljivo na slici 50.

```
1.  double p = image.size().height * 0.05;
2.  thresh = thresh(cv::Rect(p, p, image.size().width - 2 * p, image.size().height - 2 *
    p));
```

Slika 50: Uklanjanje 5% rubova slike (developer79433, 2016)

Kao rezultat ove operacije dobije se identična slika kao slike 43, samo bez 5% rubova od prijašnje slike što je i prikazano na slici 51.



Slika 51: Rezultat uklanjanja rubova slike

8. posljednji korak pronalazi konture na slici s pragom i razvrstava ih po veličini i sortira konture u padajućem poretku.

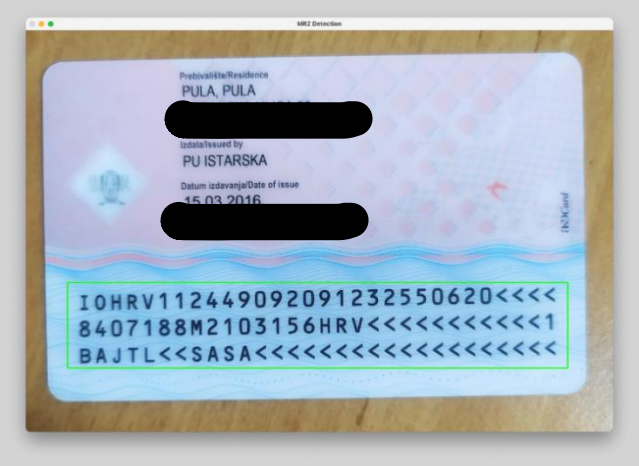
Zatim pronalazi prvu konturu s pravim omjerom i velikom širinom u odnosu na širinu slike i izračunava granični okvir konture, te upotrebljava konturu za izračunavanje omjera i omjera pokrivenosti širine graničnog okvira i širine slike. Zatim se provjerava jesu li omjer slike i širina pokrivenosti unutar prihvatljivih kriterija. Ispravlja se regija interesa (eng. *Region Of Interest*, skraćeno *ROI*) za pomak uklanjanja granice i vrši se skaliranje u odnosu na izvornu sliku. Time se dobije detektirana strojno čitljiva zona koja se sprema u strukturu MrzZoneStruct što je vidljivo na slici 52.

```
1.    // find contours in the thresholded image and sort them by their
2.    // size
3.    std::vector<std::vector<cv::Point> > contours;
4.    findContours(thresh, contours, cv::RETR_EXTERNAL,
5.                 cv::CHAIN_APPROX_SIMPLE);
6.    // Sort the contours in decreasing area
7.    sort(contours.begin(), contours.end(), [](const std::vector<cv::Point>& c1, const
    std::vector<cv::Point>& c2){
8.      return contourArea(c1, false) > contourArea(c2, false);
9.    });
10.
11.   // Find the first contour with the right aspect ratio and a large width relative to the
    width of the image
12.   cv::Rect roiRect(0, 0, 0, 0);
13.   std::vector<std::vector<cv::Point> >::iterator border_iter = find_if(contours.begin(),
    contours.end(), [&roiRect, gray](std::vector<cv::Point> &contour) {
14.     // compute the bounding box of the contour and use the contour to
15.     // compute the aspect ratio and coverage ratio of the bounding box
16.     // width to the width of the image
17.     roiRect = boundingRect(contour);
18.     // dump_rect("Bounding rect", roiRect);
19.     // pprint([x, y, w, h])
20.     double aspect = (double) roiRect.size().width / (double) roiRect.size().height;
21.     double coverageWidth = (double) roiRect.size().width / (double) gray.size().height;
22.     // cerr << "aspect=" << aspect << "; coverageWidth=" << coverageWidth << endl;
23.     // check to see if the aspect ratio and coverage width are within
24.     // acceptable criteria
25.     if (aspect > 5 and coverageWidth > 0.5) {
26.       return true;
27.     }
28.     return false;
29.   });
30.
31.   if (border_iter == contours.end()) {
32.     return false;
33.   }
34.
35.   // Correct ROI for border removal offset
36.   roiRect += cv::Point(p, p);
37.   // pad the bounding box since we applied erosions and now need
38.   // to re-grow it
39.   int pX = (roiRect.x + roiRect.size().width) * 0.03;
40.   int pY = (roiRect.y + roiRect.size().height) * 0.03;
41.   roiRect -= cv::Point(pX, pY);
42.   roiRect += cv::Size(pX * 2, pY * 2);
43. roiRect &= cv::Rect(0, 0, image.size().width, image.size().height);
44.   // Make it relative to original image again
45.   float scale = static_cast<float>(original.size().width) /
    static_cast<float>(image.size().width);
46.   roiRect.x *= scale;
47.   roiRect.y *= scale;
48.   roiRect.width *= scale;
49.   roiRect.height *= scale;
50.
```

```
51.   auto detectedZone = new MrzZoneStruct();
52.   detectedZone->x = roiRect.x;
53.   detectedZone->y = roiRect.y;
54.   detectedZone->width = roiRect.width;
55.   detectedZone->height = roiRect.height;
56.   this->setDetectedZone(*detectedZone);
57.   delete(detectedZone);
58.
59.   return true;
60. }
```

Slika 52: Izračunavanje kontura strojno čitljive zone (developer79433, 2016)

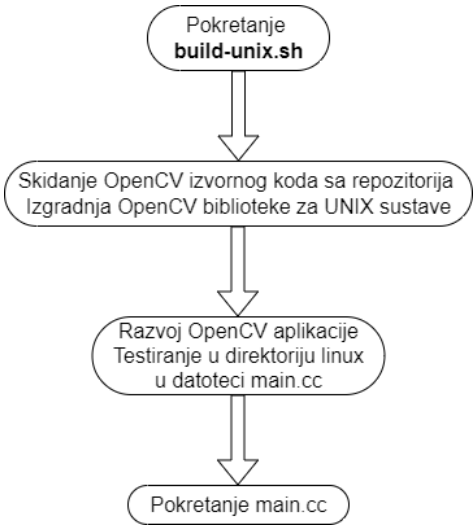
Iz posljednjeg koraka kao rezultat proizlazi originalna slika na kojoj se iscrtava detektirana strojno čitljiva zona u originalnoj nepromijenjenoj veličini slike kakva je bila prije procesa obrade. Na slici 53 prikazan je rezultat detekcije strojno čitljive zone.



Slika 53: Detekcija strojno čitljive zone

5.4 Test projekt

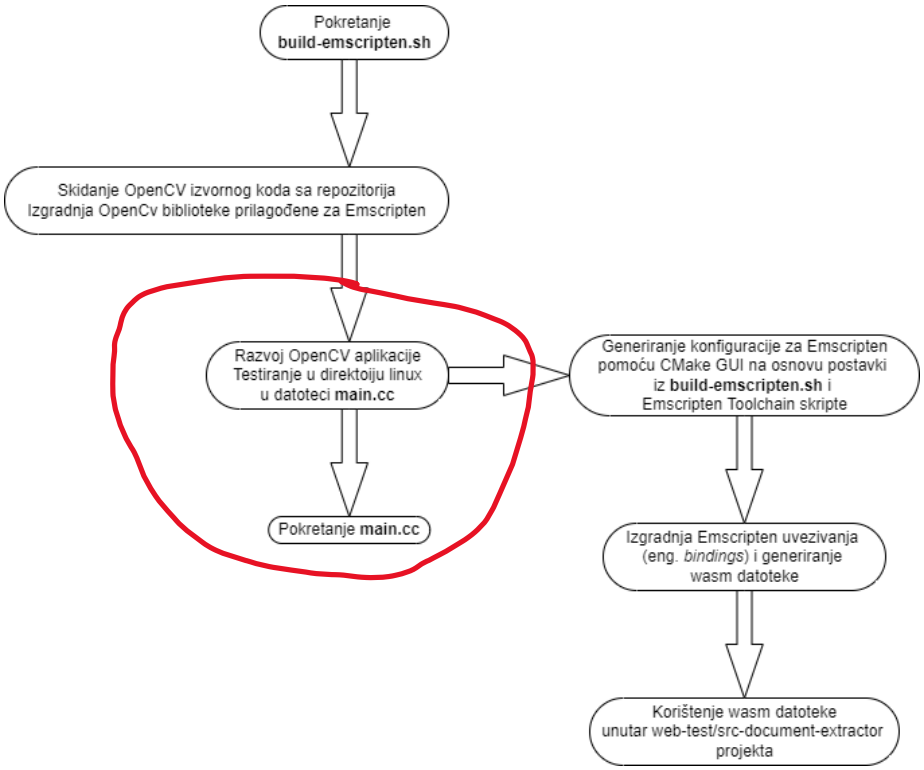
Kako je već sam proces implementacije i izgradnje biblioteke prethodno objašnjen u radu, dolazi se do zaključka, odnosno trenutačno postoje dva načina kako se implementirana biblioteka može koristiti. Može se koristiti u razvoju desktop aplikacija na Unix sustavima i razvoju web aplikacija koje je ujedno moguće koristiti i na web pretraživačima mobilnih uređaja. Što se tiče Windows platforme, isto je moguće razvijati biblioteku, ali potrebno prije postaviti Windows okruženje za razvoj C++ i OpenCV aplikacija, gdje su putanje do potrebnih direktorija OpenCV biblioteke drukčije nego na Unix sustavima a i sam prevoditelj za C++ je drukčiji nego na Unix sustavima, što ovaj rad trenutačno ne podržava. Kada je riječ o desktop razvoju, sljedeći mogući scenarij prikazan je na slici 54.



Slika 54: Prikaz toka od izgradnje biblioteke do razvoja za desktop

Vidljivo je da sam tok izgradnje i razvoja prilično jednostavan, pokrene se build-unix.sh datoteka, OpenCV izvorni kod se preuzme u zaseban direktorij, te se na osnovu konfiguracije iz build-unix.sh izgradi OpenCV biblioteka. Zatim se izgrađena biblioteka koristi za razvoj željene aplikacije, u konkretnom slučaju je to DocumentExtractor biblioteka. Implementirani kod se testira i izvršava u main.cc datoteci. Na isti način bi se razvoj odvijao i na Windows platformi.

Kad je riječ o razvoju za web aplikacije, situacija je mnogo drukčija i kompliciranija, što je i prikazano na slici 55.



Slika 55: Prikaz toka od izgradnje biblioteke do razvoja za desktop i web

Sa slike je vidljivo da se način izgradnje OpenCV biblioteke razlikuje u tome što se sada umjesto build-unix.sh pokreće build-emscripten.sh, u kojoj se nalazi drukčija konfiguracija koja je prethodno objašnjena u radu za izgradnju OpenCV biblioteke, nego u build-unix.sh skripti, te se preuzima s repozitorija i izgrađuje biblioteka. Što se tiče implementacije i testiranja i dalje se kod testira i izvršava u main.cc datoteci, ali za to koristi OpenCV biblioteku generirana s build-unix.sh (označeno crvenom bojom na slici 55). Generiranje konfiguracije za Emscripten, te uvezivanja pozivanjem webidl_binder.py skripte i generiranje wasm datoteke je opisano ranije.

5.5 Web-test dio projekta

Važno je napomenuti da se generirana wasm datoteka može koristiti u bilo kojem web projektu. U konkretnom slučajnu je to mali web projekt koji služi za testiranje wasm datoteke koji se nalazi u direktoriju projekta web-test/src-document-extractor. Wasm datoteka se učitava preko web workera pomoću metode importScripts('script-name'). Web workeri su jednostavno sredstvo za pokretanje skripti u pozadinskim nitima za web sadržaj i mogu obavljati zadatke bez ometanja korisničkog sučelja, te osim toga mogu izvoditi ulazne i izlazne operacije koristeći XMLHttpRequest (mdn web_docs, 2022) (vlastiti prijevod). Jednom kreiran worker može slati poruke JavaScript kodu koji ga je stvorio objavljivanjem poruka event handleru koji je naveden tim kodom i obrnuto (mdn web_docs, 2022) (vlastiti prijevod).

Worker je objekt kreiran pomoću konstruktora (npr. Worker()) koji pokreće imenovanu JavaScript datoteku koja sadrži kod koji će se izvoditi u radnoj niti, workeri rade u drugom globalnom kontekstu koji se razlikuje od trenutnog prozora (mdn web_docs, 2022) (vlastiti prijevod). Moguće je pokrenuti kod unutar radne niti, uz neke iznimke. Na primjer, ne može se izravno manipulirati DOM-om iz unutar radnika ili koristiti neke zadane metode i svojstva objekta prozora (mdn web_docs, 2022) (vlastiti prijevod). Podaci se šalju između workera i glavne niti putem sustava poruka u kojem obje strane šalju svoje poruke pomoću metode postMessage() i odgovaraju na poruke putem rukovatelja događaja onmessage (mdn web_docs, 2022) (vlastiti prijevod).

Konkretni primjer u radu učitavanja wasm datoteke wasm-worker.js u modul prikazan je na slici 56, odakle je vidljivo da se učitava DocumentExtractor.js i WasmHelper.js .

```
var Module = {
    onRuntimeInitialized: () => {
        postMessage({'cmd': 'onRuntimeInitialized'});

        let wasmHelper = new WasmHelper();

        addEventListener('message', onMessage);

        function onMessage(e) {
            switch(e.data.cmd){
                case "processFrame":
                    wasmHelper.ProcessFrame(e.data.image)
                    break;
            }
        }
    }
};
importScripts('DocumentExtractor.js', '../helpers/WasmHelper.js');
```

Slika 56: Učitavanje datoteka u wasm modul

Nakon što je skripta učitana u wasm modul potrebno je neko vrijeme da se skripta učita, te se na događaj onRuntimeInitialized šalje poruka u glavnu nit (eng. *main thread*), odnosno obavještava je da je okolina inicijalizirana s funkcijom postMessage({'cmd': 'onRuntimeInitialized'}). Zatim se kreira nova istanca WasmHelper objekta koja sadrži pomoćne funkcije koje služe za upotrebu DocumentExtractor.js modula, te se registrira jedan slušač događaja (eng. *event listener*) koji služi kako bi na događaj 'message' pokrenuo funkciju onMessage(e) koja pokreće funkciju ProcessFrame koja započinje proces detekcije pozivanjem DocumentExtractor biblioteke. Na slici 57 prikazan je sadržaj datoteke WasmHelper.js.

```
class WasmHelper {
    startTime = 0;
    endTime = 0;

    ToHeap(typedArray) {
        const numBytes = typedArray.length * typedArray.BYTES_PER_ELEMENT;
        this._ptr = Module._malloc(numBytes);
        let heapBytes = Module.HEAPU8.subarray(this._ptr, this._ptr + numBytes);
        heapBytes.set(typedArray)
        return heapBytes;
    }
    Free(heapBytes) {
        Module._free(heapBytes.byteOffset);
    }
    ProcessFrame(image) {
        this.startTime = performance.now();

        this._frame_bytes = this.ToHeap(image.data);
        let extractor = new Module.Extractor();
        let zone = extractor.detectMrzZone(this._frame_bytes.byteOffset, image.width,
image.height);
        this.Free(this._frame_bytes);

        //postMessage({'cmd': 'drawProcessedFrame', 'image':
imageData},[imageData.data.buffer]);
        postMessage({
            'cmd': 'setDetectedZone',
```

```
26.             'zone':
27.                 {
28.                     x: zone.get_x(),
29.                     y: zone.get_y(),
30.                     width: zone.get_width(),
31.                     height: zone.get_height()
32.                 }
33.         });
34.         Module.destroy(extractor);
35.         this.endTime = performance.now();
36.
37.         console.log("ProcessFrame", parseFloat((this.endTime -
    this.startTime)).toFixed(2), "ms")
38.
39.         postMessage({'cmd': 'timeFinished', 'elapsedTime': parseFloat(1000 /
    (this.endTime - this.startTime)).toFixed(1)});
40.     }
41. };
```

Slika 57: Sadržaj datoteke WasmHelper.js

Sa slike je vidljivo da je prije samog kreiranja objekta Module.Extractor() potrebno alocirati memoriju s pomoćnom funkcijom ToHeap(typedArray) implementirana na liniji 6, jer wasm radi u zaštićenom okruženju (eng. *sandbox*) i ne može izravno pristupiti memoriji. Memorija potrebna za izvođenje C/C++ programa predstavljena je nizom upisanim u JavaScriptu što je u konkretnom slučaju HEAPU8, odnosno način na koji je memorija predstavljena i koju čine 8-bitni ne označeni cijeli brojevi što je ekvivalent C++ uint8_t, tip podatka koji je već ranije spomenut u radu.

Veličina memorije potrebne za alokaciju odgovara veličini jednog okvira slike, dakle svake sekunde se uzima nekoliko slika iz kamere i na svakoj slici se vrši proces detekcije strojno čitljive zone. Koliko će slika biti obrađeno u sekundi ovisi o načinu implementacije i optimizaciji koda u C++, te optimizacije od strane Emscripten okvira ali i o kvaliteti i performansama uređaja na kojem se detekcija izvodi. Svaki put kad se memorija alocira i obavi operacija detekcije (18., 19. i 20. linija koda) potrebno je tu istu memoriju i osloboditi s pomoćnom funkcijom Free(heapBytes), vidljivo na liniji 21. Nakon oslobađanja memorije detektirana zona se šalje u glavnu nit programa s funkcijom postMessage na liniji koda 24, kojoj se kao parametar prosljeđuje objekt detektirane zone. Zatim se proslijeđeni objekt detektirane zone čeka u datoteci main.js gdje je postavljen slušač događaja koji sluša nekoliko događaja od kojih se na 'setDetectedZone' postavlja detektirana zona i iscrtava unutar html platna (eng. canvas), što je i vidljivo na slici 58, linija 11.

```
let wasmWorker = new Worker('wasm-worker.js');

. . .

function setListeners(){
    wasmWorker.addEventListener('message', async (e) => {
        switch(e.data.cmd) {
            case 'onRuntimeInitialized':
                await onRuntimeInitialized();
                break;
            case 'setDetectedZone':
                setDetectedZone(e.data.zone);
                break;
            case 'timeFinished':
                setElapsedTime(e.data.elapsedTime)
                break;
        }
    }, false);
}

```

Slika 58: Primjer slušača događaja u main.js datoteci

6. Evaluacija radnih karakteristika

Implementacija DocumentExtractor biblioteke testirana je na nekoliko web preglednika i različitih uređaja, te na desktopu MacOS. Napravljeno je i mjerenje vremena koje je potrebno od kreiranja instance DocumentExtractor objekta, procesa detekcije strojno čitljive zone pa sve do završetka detekcije, kako bi se vidjele radne karakteristike u različitim uvjetima. Mjerenja su pokazala zaista iznenađujuće i neočekivane rezultate o kojima će se govoriti u nastavku. Za testiranje su se koristili web preglednici i uređaji:

- Google Chrome
    - Xiaomi Redmi Note 7
    - Xiaomi Pad 5
    - MacBook Pro (2021, Apple M1 Pro)
    - Lenovo Legion (i7-10750H)
- FireFox
    - Xiaomi Redmi Note 7
    - Xiaomi Pad 5
    - MacBook Pro (2021, Apple M1 Pro)
    - Lenovo Legion (i7-10750H)
- Microsoft Edge
    - Xiaomi Redmi Note 7
    - Xiaomi Pad 5
    - MacBook Pro (2021, Apple M1 Pro)
    - Lenovo Legion (i7-10750H)
- MacOS desktop

U nastavku u tablici 14 prikazani su rezultati za svaki navedeni preglednik.

Tablica 14: Rezultati mjerenja u FPS

	Xiaomi Redmi Note 7	Xiaomi Pad 5	MacBook Pro	Lenovo Legion
Chrome	149,3	263,2	1250	416,7
Firefox	142,9	333,3	1000	500,0
Edge	-	-	1250	454,5
Desktop	-	-	305,2	-

Mjerenje se obavlja po formuli *1000 / (vrijeme kraja – vrijeme početka)* kako bi se rezultat mogao prikazati u mjernoj jedinici slika u sekundi (eng. FPS – Frame Per Second), gdje su rezultati mjerenja informativnog karaktera. Rezultati mjerenja ovise o trenutku uzimanja slika te osvjetljenju i kvaliteti kamere uređaja.

Iz prikazanih rezultata se vidi da na Xiaomi Redmi Note 7 razlika između Chrome i Firefox preglednika nije velika. Ispada da je Chrome brži za 6,4 FPS. Na Xiaomi Pad 5 ispada da je Chrome sporiji nego Firefox za otprilike 70,1 FPS. Niti na jednom od Xiaomi uređaja nije bilo moguće pokrenuti kameru u Edge pregledniku, gdje je prilikom pokretanja vidljiva samo crna slika.

U slučaju MacBook Pro performanse su znatno bolje gdje je Chrome brži od Firefoxa za približno 250 FPS. Zanimljiva stvar je da su Edge i Chrome gotovo isti jer je Edge baziran na Chromium pregledniku koji je otvorenog koda. Može se primijetiti jedna ogromna razlika kod Desktop mjerenja gdje ispada da sve izvršeno s nativnim C++ kodom je mnogo sporije nego izvođenje wasm datoteke na preglednicima. Razlog tome je taj što se u desktop dijelu dodatno koristi OpenCV modul videoio i još mnogi drugi što znatno utječe na rezultate. Na web preglednicima nije moguće recimo koristiti modul videoio i mnoge druge module koje koristi desktop dio aplikacije, sve se odvija unutar canvasa. Razlika između Lenovo prijenosnog računala i MacBook Pro računala je gotovo dvostruka gdje se vidi da je Lenovo očekivano sporiji zbog vrste i jačine procesora. Veličina same wasm datoteke je ok 2,5 MB što je postignuto raznim optimizacijskim zastavicama prilikom prevođenja C++ sa Emscripten u wasm što je već ranije spomenuto. Moguće je još bolje postaviti optimizacijska pravila kako bi se izvođenje wasm datoteke brže odvijalo i sama datoteka bila manja. Na desktop dijelu aplikacije nije rađena nikakva optimizacija, te je trenutno moguće pokrenuti desktop aplikaciju samo na Unix sustavima jer trenutno nije predviđena za razvoj na Windows platformi.

7. Zaključak

Detekcija strojno čitljive zone na osobnim dokumentima izgleda kao lagani i brz proces, ali za uspješnu detekciju potrebno je nekoliko koraka obrade slika uzetih putem kamere, te kompleksnih izračuna koji utječu na brzinu obavljanja zadatka. Rijetko se brzina stavlja ispred stabilnosti, no s druge strane, ako je aplikacija spora isto je vrlo iritirajuće za korisnika, osobito ako je za neki zadatak potrebno više vremena i ako je tih zadataka veoma mnogo. U stvarnom svijetu možda ne zvuči mnogo da se za određeni zadatak utroši 1 do 2 sekunde, no ako je takvih zadataka puno onda to predstavlja veliki problem. Stoga je potrebno pronaći balans između performansi i stabilnosti gdje stabilnost dolazi nekako uvijek ispred svega, čemu se i dalje teži uz potrebu za poboljšanjem brzine izvođenja i optimizacije.

U ovom radu je opisana situacija gdje je uspješno postignuta implementacija detekcije strojno čitljive zone na osobnim dokumentima, koja je napravljena na krajnje nekonvencionalan način na koji bi se vjerojatno odlučio mali broj programera. Razlog tome je što se za implementaciju prvenstveno koristilo nekoliko različitih tehnologija od kojih su prethodno opisani, programski jezik C++, Shell skriptiranje, OpenCV biblioteka, WebAssembly, Emscripten i CMake. Srž svega leži u Shell skriptama koje su povezane sa CMake sustavom koji na njihovo pokretanje ovisno o okolini, da li je Emscripten ili desktop, generira izvorni kod za izgradnju, te je bilo potrebno utrošiti dosta vremena kako bi se postigao željeni način izgradnje.

U radu je prikazano kako se implementacija detekcije strojno čitljive zone uspješno izvršava i na web preglednicima i na desktopu, gdje je isto tako vidljiva ogromna razlika u brzini između ove dvije okoline. Iz rezultata mjerenja neočekivano jest ustvrđeno da je aplikacija brža na web preglednicima nego na desktopu. Uz sve prednosti koje donosi ovakav način izgradnje, naravno da postoje i neki nedostaci. Neke od prednosti ovakvog načina izgradnje aplikacija:

- Izvođenje aplikacije na gotovo svim web preglednicima raznih platformi
- Velika brzina izvođenja koda
- Znatno manja veličina aplikacije, u konkretnom slučaju zbog ne korištenja nepotrebnih modula OpenCV biblioteke
- Velika mogućnost optimizacije

Neki od nedostataka ovakvog načina izgradnje:

- Kompleksnost tijekom podešavanja CMake sustava izgradnje
- Poteškoće kod otkrivanja i ispravljanja pogreški koje mogu biti na obje strane, JavaScript i C++
- Zbog nepravilnog postavljanja optimizacijskih zastavica performanse znatno lošije nego očekivano

Moguće je bilo i pojednostaviti ovakav način izgradnje tako da se potpuno izbaci CMake iz upotrebe. Isto gledano dugoročno i nije najbolja opcija jer se onda svaki izvorišni kod mora prevesti s Emscripten razvojnim okvirom ručno, što je za velike projekte neodrživo. Isto tako je bilo moguće implementirati detekciju strojno čitljive zone samo s JavaScriptom umjesto C++, gdje OpenCV biblioteka nudi mogućnost izgradnje koda za odabrane module, ali ovakav način je dosta spor. Ovakav način izgradnje koji je predstavljen u ovom radu omogućava da se napravi bilo koja biblioteka koja se želi koristiti na više platformi. To ne mora nužno biti OpenCV biblioteka, već se može koristiti kao nekakav predložak za druge projekte što je bez obzira na neke od spomenutih nedostataka još uvijek značajno poboljšanje po pitanju i stabilnosti i brzine.

Literatura

Battagline, R. (2021) *The Art of WebAssembly*.

Cmake (2000) *Cmake org*. Available at: https://cmake.org (Accessed: March 15, 2022).

developer79433 (2016) *Passport mrz detector*. Available at: https://github.com/developer79433/passport_mrz_detector_cpp (Accessed: June 2, 2022).

Doubango (2011) *MRZ formats — ultimateMRZ 2.6.0 documentation*. Available at: https://www.doubango.org/SDKs/mrz/docs/MRZ_formats.html (Accessed: September 29, 2021).

Emscripten Contributors (2015) *Emscripten*. Available at: https://emscripten.org/index.html (Accessed: February 21, 2022).

Escriva, D., Laganiere, R. and Safari, an O. M. Company. (2019) *OpenCV 4 Computer Vision Application Programming Cookbook - Fourth Edition*.

Fernández Villán, A. (2019) *Mastering OpenCV 4 with Python*.

Fileformat (2001) *SH scripting language*. Available at: https://docs.fileformat.com/programming/sh/ (Accessed: March 26, 2022).

Gallant, G. (2019) *WebAssembly in Action*.

iDenfy (2022) *Understanding: Machine Readable Zone - iDenfy*. Available at: https://www.idenfy.com/blog/machine-readable-zone/ (Accessed: September 29, 2021).

Leo Gamboa Uribe (2020) *Computer Vision VS Human Vision: This is how computers see*. Available at: https://santanderglobaltech.com/en/computer-vision-vs-human-vision-this-is-how-computers-see/ (Accessed: February 8, 2022).

LLVM (2000) *LLVM Project*. Available at: https://llvm.org/ (Accessed: February 28, 2022).

McAnlis, C. *et al.* (2014) “HTML5 Game Development Insights.”

mdn web_docs (2022) *Web Workers API*. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (Accessed: April 14, 2022).

Microblink (2022) *MRTD MRZ FAQ – Microblink Help Center*. Available at: https://help.microblink.com/hc/en-us/articles/360007487833-MRTD-MRZ-FAQ (Accessed: September 29, 2021).
Mozilla (2005) *MDN Web Docs*. Available at: https://developer.mozilla.org/en-US/docs/Games/Tools/asm.js?source=post_page (Accessed: February 13, 2022).
OpenCV (2022a) *cv::Mat Class Reference*. Available at: https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#details (Accessed: April 5, 2022).
OpenCV (2022b) *cv::Size_< _Tp > Class Template Reference*. Available at: https://docs.opencv.org/3.4/d6/d50/classcv_1_1Size__.html#details (Accessed: April 5, 2022).
OpenCV (2022c) “OpenCV Image Filtering.” Available at: https://docs.opencv.org/4.5.2/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f (Accessed: April 6, 2022).
OpenCV (2022d) *OpenCV Introduction*. Available at: https://docs.opencv.org/4.x/d1/dfb/intro.html (Accessed: February 10, 2022).
Rosebrock, A. (2021) *OpenCV Smoothing and Blurring*. Available at: https://pyimagesearch.com/2021/04/28/opencv-smoothing-and-blurring/ (Accessed: April 6, 2022).
Scott, C. (2018) *A PRACTICAL GUIDE Professional CMake: A Practical Guide*. Available at: https://crascit.com.
Soft float library routines (1988) *Soft float library routines*. Available at: https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html (Accessed: March 1, 2022).
Web IDL (2022) *Web IDL - Living Standard*. Available at: https://webidl.spec.whatwg.org/#introduction (Accessed: April 14, 2022).
WebAssembly (2022) *WebAssembly*. Available at: https://webassembly.org/ (Accessed: June 1, 2022).
Wikipedia (2022a) *Computer vision*. Available at: https://en.wikipedia.org/wiki/Computer_vision (Accessed: July 3, 2022).
Wikipedia (2022b) *LLVM Wikipedia*. Available at: https://en.wikipedia.org/wiki/LLVM (Accessed: May 3, 2022).
Wikipedia (2022c) *Sobel operator*. Available at: https://en.wikipedia.org/wiki/Sobel_operator (Accessed: April 7, 2022).

Popis slika

Slika 1: Stražnjeg dijela dokumenta sa strojno čitljivom zonom (Doubango, 2011).... 4
Slika 2: Primjer dokumenta tipa 2 sa strojno čitljivom zonom (Doubango, 2011) 6
Slika 3: Primjer putovnice sa strojno čitljivom zonom (Doubango, 2011) 8
Slika 4: Primjer vize sa strojno čitljivom zonom (Doubango, 2011) 9
Slika 5: Raspored abecednih znakova od A do Z u odnosu na vrijednosti (Microblink, 2022) 11
Slika 6: Ljudski vid (Leo Gamboa Uribe, 2020)..... 13
Slika 7: Binarni prikaz boje slike (Leo Gamboa Uribe, 2020)..... 14
Slika 8: Reprezentacija u 256 razina intenziteta (Leo Gamboa Uribe, 2020)..... 14
Slika 9: Reprezentacija u boji (Leo Gamboa Uribe, 2020)..... 14
Slika 10: Asm modul (Gallant, 2019) 18
Slika 11: Osnovna struktura WebAssembly binarnog koda (Gallant, 2019) 21
Slika 12: Veza između *Type*, *Function* i *Code* sekcija (Gallant, 2019)..... 22
Slika 13: Tijek rada Emscripten prevodioca (McAnlis *et al.*, 2014) 26
Slika 14: Primjer lerp funkcije napisane u C 26
Slika 15: lerp u LLVM IR (McAnlis *et al.*, 2014) 27
Slika 16: lerp prevedena u JavaScript (McAnlis *et al.*, 2014)..... 27
Slika 17: Prevođenje programa sa emcc (McAnlis *et al.*, 2014)..... 28
Slika 18: Proces izgradnje projekta sa CMake (Scott, 2018)..... 30
Slika 19: Out of source struktura (Scott, 2018)..... 31
Slika 20: Izvršavanje cmake komande u komandnoj liniji (Scott, 2018)..... 32
Slika 21: Ispis nakon izvršenja cmake komande (Scott, 2018)..... 32
Slika 22: Izvršavanje komande za generiranje projekta (Scott, 2018) 33
Slika 23: Generiranje projekta sa dodatnim parametrima (Scott, 2018) 34
Slika 24: Struktura projekta DocumentExtractor..... 35
Slika 25: Primjer build-emscripten.sh skripte..... 38
Slika 26: Primjer dodatnih opcija build-emscripten.sh skripte..... 38
Slika 27: OpenCV moduli za Emscripten okolinu..... 39
Slika 28: Dodatne opcije za generiranje wasm..... 39
Slika 29: OpenCV moduli za desktop okolinu..... 40
Slika 30: Primjer CMakeLists.txt unutar glavnog direktorija..... 41
Slika 31: Primjer DocumentExtractor.h datoteke 43

Slika 32: Postavke u CMakeLists.txt od Emscripten projekta 45
Slika 33: Datoteka interfaces.idl 46
Slika 34: Datoteka interfaces.idl 46
Slika 35: Sadržaj datoteke Extractor.cc 48
Slika 36: Sadržaj datoteke interfaces_glue.cpp 49
Slika 37: Dio datoteke interfaces_glue.js 50
Slika 38: Implementacija getter i setter funkcija 51
Slika 39: Prikaz implementacije promjene veličine i pretvaranje u sivi kanal slike 51
Slika 40: Promjena veličine i pretvaranje slike u sivu boju 52
Slika 41: Zamagljivanje Gaussovim filterom 53
Slika 42: Zamagljivanje i pronalaženje tamnih područja na svijetloj pozadini (developer79433, 2016) 53
Slika 43: Rezultat nakon korištenja morfološkog operatora 54
Slika 44: Izračunavanje Scharrovog gradijenta slike blackhat i skaliranje (developer79433, 2016) 54
Slika 45: Detekcija rubova Sobel i Scharr 54
Slika 46: Zatvaranje praznina među znakovima – Otsu binarizacija (developer79433, 2016) 55
Slika 47: Horizontalno zatvaranje praznina 55
Slika 48: Operacija zatvaranja četvrtastom jezgrom (developer79433, 2016) 55
Slika 49: Vertikalno zatvaranje praznina 56
Slika 50: Uklanjanje 5% rubova slike (developer79433, 2016) 56
Slika 51: Rezultat uklanjanja rubova slike 56
Slika 52: Izračunavanje kontura strojno čitljive zone (developer79433, 2016) 58
Slika 53: Detekcija strojno čitljive zone 58
Slika 54: Prikaz toka od izgradnje biblioteke do razvoja za desktop 59
Slika 55: Prikaz toka od izgradnje biblioteke do razvoja za desktop i web 59
Slika 56: Učitavanje datoteka u wasm modul 61
Slika 57: Sadržaj datoteke WasmHelper.js 62
Slika 58: Primjer slušača događaja u main.js datoteci 63

Popis tablica

Tablica 1: Prvi redak dokument tip 1 (Doubango, 2011) (vlastiti prijevod) 5

Tablica 2: Drugi redak dokument tip 1 (Doubango, 2011) (vlastiti prijevod)..... 5
Tablica 3: Treći redak dokument tip 1 (Doubango, 2011) (vlastiti prijevod) 5
Tablica 4: Prvi redak dokument tip 2 (Doubango, 2011) (vlastiti prijevod) 6
Tablica 5: Drugi redak dokument tip 2 (Doubango, 2011) (vlastiti prijevod)..... 7
Tablica 6: Prvi redak dokument tip 3 (Doubango, 2011) (vlastiti prijevod) 8
Tablica 7: Drugi redak dokument tip 3 (Doubango, 2011) (vlastiti prijevod)..... 9
Tablica 8: Prvi redak strojno čitljive vize (Doubango, 2011) (vlastiti prijevod)..... 10
Tablica 9: Drugi redak strojno čitljive vize (Doubango, 2011) (vlastiti prijevod) 10
Tablica 10: Primjer izračuna kontrolne znamenke 12
Tablica 11: Primjer izračuna složene kontrolne znamenke..... 12
Tablica 12: Primjer izračuna složene kontrolne znamenke (nastavak od prethodne) 12
Tablica 13: CMake generatori (Scott, 2018) 32
Tablica 14: Rezultati mjerenja u FPS 64

Sažetak

Tijekom posljednjih nekoliko godina sve više se teži izradi aplikacija koje imaju jedan izvorišni kod, te isto tako po mogućnosti, taj isti kod da se optimalno izvršava na što više platformi. Razlog tome su troškovi koji mogu uslijediti kada bi za svaku platformu postojao jedan ili više programera. Da bi se to izbjeglo, danas se sve više aplikacija kreira koje rade na svakom ili gotovo svakoj vrsti web preglednika, uz to da na aplikaciji radi jedan do nekoliko programera. Dakle za ovakav način izrade je potreban jedan tim stručnjaka, dok je nekad za svaku platformu bio potreban jedan stručnjak. U tom slučaju riječ multiplatforma dolazi do izražaja.

Fokus ovog rada je na implementaciji jedne takve multiplatformske aplikacije čiji je izvorišni kod pisan u C++ koristeći OpenCV biblioteku računalnog vida, te se odnosi na multiplatformsku detekciju strojno čitljive zone na osobnim dokumentima. Osim C++ i OpenCV, za sustav izgradnje se koristi Shell skripte i Cmake sustav koji je odgovoran za automatsku izgradnju koda pokretanjem Shell skripti, od kojih je svaka skripta za izgradnju napisana za željenu platformu, u konkretnom slučaju desktop i web platformu, gdje za web platformu glavnu ulogu igra Emscripten razvojni okvir, prevodeći izvorišni kod u WebAssembly, predstavljajući sam pojam multiplatforme. Pored nabrojanih tehnologija, završnu riječ ima JavaScript preko kojeg se WebAssembly generiran kod koristi na web preglednicima. Rezultati rada objašnjavaju

razlog zbog kojeg je ovakav način razvoja odličan, te uz neke sitne nedostatke prilikom razvoja, u konačnici prevladavaju pozitivne strane ovakve vrste razvoja od kojih je dokazana brzina izvođenja i stabilnost koja se može dovesti do zavidne razine.

Ključne riječi: multiplatforma, strojno čitljiva zona, C++, opencv, emscripten, wasm, webassembly, cmake, JavaScript.

Abstract

Over the last few years, there has been an increasing effort to build applications that have a single source code and, if possible, the same code to run optimally on as many platforms as possible. The reason for this is the cost that could follow if there were one or more developers for each platform. To avoid this, today more and more applications are being made that work on each or almost every type of web browser, with one to several developers working on the application. So this way of making requires one team of experts, while once one platform needed one expert. In this case, the word multiplatform comes to the fore.

This paper focuses on implementing one such multiplatform application whose source code is written in C ++ using the OpenCV computer vision library and refers to the multiplatform detection of the machine-readable zone on personal documents. In addition to C ++ and OpenCv, the construction system uses Shell scripts and the Cmake system, which is responsible for automatically building code by running Shell scripts, each of which is written for the desired platform, in this case, desktop and web platform, where for the web platform, the Emscripten development framework plays a major role, translating the source code into WebAssembly, representing the very concept of multiplatform. In addition to the listed technologies, the final word is JavaScript, through which WebAssembly generated code is used on web browsers. The results explain the reason why this method of development is excellent, and with some minor shortcomings in development, ultimately the positive aspects of this type of development prevail, of which proven speed and stability can be brought to an enviable level.

Keywords: multiplatform, machine readable zone, C++, opencv, emscripten, wasm, webassembly, cmake, JavaScript.