Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

* Detekcija ivica unutar slike -

Student: Džudžar Nataša, SV14/2021

Predmet: Paralelno programiranje

# Sadržaj

1. Uvod i analiza projektnog zadatka ......................................................................................... 3
2. Koncept rešenja .................................................................................................................... 4
   1. Koncept algoritma sa pretragom okoline pojedinačnog piksela ................................... 4
   2. Koncept algoritma sa implementacijom Prewitt operatera ......................................... 5
3. Programsko rešenje .............................................................................................................. 6

3.1. Implementacija algoritma sa pretragom okoline pojedičnog piksela ............................ 6

3.2. Implementacija Prewitt algoritma ................................................................................. 7

1. Ispitivanje dobijenih rezultata ............................................................................................... 11
   1. Fizičke karakteristike računara ....................................................................................... 11
   2. Testni slučajevi ................................................................................................................ 11
   3. Rezultati vrednosti merenja ............................................................................................ 13
2. Analiza dobijenih rezultata .................................................................................................... 16

# Uvod i analiza projektnog zadatka

Detekcija ivica je osnovna tehnika koja se koristi u digitalnoj obradi slike za identifikaciju i isticanje granica ili prelaza između različitih regiona ili objekata unutar slike. Prvenstveno igra ulogu u izdvajanju važnih karakteristika i razumevanju strukture slike. Proces detekcije ivica bi se ukratko mogao opisati na sledeći način:

1. ***Konverzija u nijanse sive***: Prvenstveno se vrši konverzija fotografije u boji u eng. *grayscale* ako to već prethodno nije urađeno. Ovo pojednostavljuje proces detekcije ivica svođenjem slike na jedan kanal intenziteta.
2. ***Izglađivanje (opciono)***: Primenjuje se filter za izglađivanje ili zamućivanje da bi se smanjio šum i uklonili sitni detalje koji mogu da ometaju detekciju ivica. Uobičajeni filteri uključuju Gausovo zamućenje ili *median* filter.
3. ***Izračunavanje gradijenta***: Izračunavanje gradijenta slike primenom operatora, kao što je Sobel, Prewitt ili Robertsov operator. Ovi operatori izračunavaju veličinu i smer gradijenta za svaki piksel, ukazujući na promenu intenziteta na slici.
4. ***Poboljšanje ivica***: Naglašavanje ivica kako bi bile izraženije. Ovaj korak može uključivati operacije kao što su postavljanje praga, ne-maksimalno potiskivanje ili normalizacija veličine gradijenta. Prag konvertuje magnitude gradijenta u binarne vrednosti, naglašavajući piksele sa značajnim vrednostima gradijenta.
5. ***Lokalizacija ivice***: Preciziranje otkrivenih ivice izvođenjem tehnika lokalizacije ivica. Ovaj korak pomaže u preciznoj lokalizaciji ivičnih piksela potiskivanjem nemaksimalnih ivica i stanjivanjem ivica.
6. ***Naknadna obrada (opciono)***: Izvršite dodatne operacije naknadne obrade da biste poboljšali kvalitet otkrivenih ivica. Ovo može uključivati tehnike kao što su povezivanje ivica ili izglađivanje konture za povezivanje i preciziranje otkrivenih ivica.

Različiti algoritmi i varijacije za detekciju ivica mogu imati dodatne korake ili različite pristupe, ali gornji koraci pružaju sažet pregled opšteg procesa detekcije ivica.

Vredi napomenuti da je detekcija ivica aktivna oblast istraživanja i da postoje razni napredni algoritmi i tehnike za rešavanje specifičnih izazova i zahteva u različitim aplikacijama.

Konkretno na projektnom zadatku su implementirana dva algoritma:

* ***Algoritam sa Prewitt operatorom I reda***
* ***Algoritam sa pretragom okoline svakog piksela***

Za implementaciju algoritama potrebno je da sve slike budu u BMP formatu i RGB prostorom boja, jer se parsiranje vrši pomoću EasyBMP biblioteke.

# Koncept rešenja

U osnovi digitalne obrade slike nalazi se operacija dvodimenzionalne diskretne konvolucije. Uprošćeno, ova operacija se može svesti na množenje odgovarajućih elemenata dveju matrica i sumiranje rezultata.

Prva matrica u operaciji konvolucije je jezgro digialnog filtra (kvadratna matrica dimenzija 3x3, 5x5, 7x7 itd.).

Druga matrica je podmatrica slike koja sadrži podatke o vrednostima boja pojedinih tačaka (piksela) slike, i istih je dimenzija kao odabrani filter.

Drugim rečima, cela operacija se može posmatrati kao klizanje matrice filtra preko svih podmatrica slike i obavljanje aritmetičkih operacija nad svakim ovako stvorenim parom.

## Algoritam sa pretragom okoline svakog piksela

Okolinu tačke predstavljaju njoj susedne tačke: *levo, desno, iznad, ispod i dijagonalno na sve strane*. Proces obrade funkcioniše prvenstveno tako što se vrednosti skaliraju na 0 i 1, kako bi se uklonila siva komponenta. Nakon uklanjanja sive komponente se vrši pretraga okoline, na sledeći način:

P(i,j) = 1, ako u okolini tačke postoji tačka sa vrednošću 1

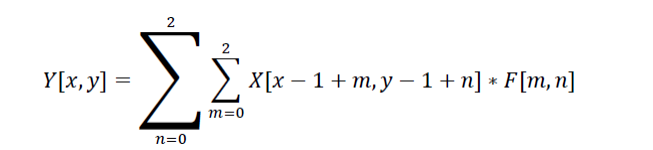
P(i,j) = 0, ako u okolini tačke ne postoji tačka sa vrednošću 1

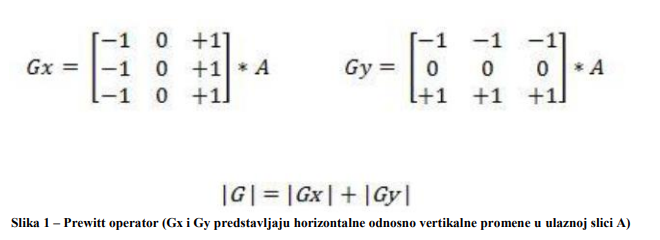
O(i,j) = 0, ako u okolini tačke postoji tačka sa vrednošću 0

O(i,j) = 1, ako u okolini tačke ne postoji tačka sa vrednošću 0

Potom se upisuju vrednosti u izlaznu matricu, gde se rezultujuće tačke dobijaju razlikom apsolutnih vrednosti P(i,j) i O(i,j) sa rezultatom vraćenim u opseg 0-255.

## Agoritam sa implementacijom Prewitt operatera



Prewitt perator se sastoji od dva konvolucijska jezgra dimenzija 3x3. Jednog za detekciju vertikalnih ivica i drugog za detekciju horizontalnih ivica.

Da bismo primenili Prewitt operator, jezgra se konvolutiraju sa slikom, tj. intenzitet svakog piksela se zamenjuje vrednošću koja predstavlja jačinu ivice na toj lokaciji. Vertikalno jezgro detektuje vertikalne ivice, dok horizontalno jezgro detektuje horizontalne ivice. Kombinacijom rezultata oba jezgra mogu se identifikovati ivice u različitim pravcima.

Prewitt operator je jednostavna i računski efikasna tehnika za detekciju ivica, iako može proizvesti deblje ivice i bitno je napomenuti da je poprilično osetljiv na šum.

# 3. Programsko rešenje

Da bi uopšte mogli pokrenuti naše rešenje potrebno je za početak zadovoljiti par uslova:

* Ulazne fotografije su u BMP formatu sa RGB prostorom boja
* parametri komandne linije su nazivi fajlova:
  + ulazni fajl
  + izlazni fajlovi za serijsku i paralelnu verziju prewitt algoritma+algoritma sa pretragom okoline svakog filtera.
* Takođe, u zaglavlju programa su definisane konstante **FILTER\_SIZE** (dimenzije filtera), **NEIGHBOURHOOD\_SIZE** (dimenzija posmatranog okruženja) i **CUTOFF** (granični slučaj dimenzije matrice).

## 3.1. Implementacija algoritma sa pretragom pojedinačnog piksela

Razlika u postupku između algoritma sa Prewitt operaterom i metode koja se služi pretragom okoline pojedinačnog

void filter\_serial\_edge\_detection(int\* inBuffer, int\* outBuffer, int width, int height, int col = 0, int row = 0)

{

int lowerX, lowerY, upperX, upperY;

lowerX = (col < SKIP\_NEIGHBOURS) ? SKIP\_NEIGHBOURS : col;

lowerY = (row < SKIP\_NEIGHBOURS) ? SKIP\_NEIGHBOURS : row;

upperX = (col + width > totalWidth - (SKIP\_NEIGHBOURS)) ? totalWidth - (SKIP\_NEIGHBOURS) : col + width;

upperY = (row + height > totalHeight - (SKIP\_NEIGHBOURS)) ? totalHeight - (SKIP\_NEIGHBOURS) : row + height;

for (int x = lowerX; x < upperX; x++) {

for (int y = lowerY; y < upperY; y++) {

outBuffer[x + y \* totalWidth] = checkNeighbours(inBuffer, x, y);

}

}

}

void filter\_parallel\_edge\_detection(int\* inBuffer, int\* outBuffer, int width, int height, int col = 0, int row = 0)

{

if (min(height, width) <= CUTOFF) {

filter\_serial\_edge\_detection(inBuffer, outBuffer, width, height, col, row);

}

else {

task\_group t;

int taskHeight = height / 2;

int restHeight = height - taskHeight;

int taskWidth = width / 2;

int restWidth = width - taskWidth;

t.run([&] {filter\_parallel\_edge\_detection(inBuffer, outBuffer, taskWidth, taskHeight, col, row); });

t.run([&] {filter\_parallel\_edge\_detection(inBuffer, outBuffer, restWidth, taskHeight, col + taskWidth, row); });

t.run([&] {filter\_parallel\_edge\_detection(inBuffer, outBuffer, taskWidth, restHeight, col, row + taskHeight); });

t.run([&] {filter\_parallel\_edge\_detection(inBuffer, outBuffer, restWidth, restHeight, col + taskWidth, row + taskHeight); });

t.wait();

}

}

## 3.2. Implementacija Prewitt algoritma

***Funkcija filter***()

int filter(int\* inBuffer, int x, int y) {

int save = SKIP\_PREWITT - 1;

int G = 0, Gx = 0, Gy = 0, raw;

for (int n = 0; n < FILTER\_SIZE; n++) {

for (int m = 0; m < FILTER\_SIZE; m++) {

raw = inBuffer[(x - save + m) + (y - save + n) \* totalWidth];

Gx += raw \* filterHor[m + n \* FILTER\_SIZE];

Gy += raw \* filterVer[m + n \* FILTER\_SIZE];

}

}

G = sqrt(Gx \* Gx + Gy \* Gy);

return 255 \* scale(G);

}

Funkcija uzima bafer ulazne slike (***inBuffer***), koordinate piksela od interesa (***x i y***) i ukupnu širinu slike (***totalWidth***). Potom primenjuje Prewitt operator na podmatricu koja je centrirana oko zadatih koordinata piksela.

Prewitt operator se sastoji od dva filtera: ***filterHor*** za horizontalne ivice i ***filterVer*** za vertikalne ivice. Filteri se konvoluiraju sa podmatricom slike kako bi se izračunali ***horizontalni gradijent*** (***Gx***) i ***vertikalni gradijent*** (***Gy***) na toj poziciji piksela.

***Magnituda gradijenta*** (G) se izračunava uzimanjem korena kvadratnog zbirka kvadrata horizontalnog i vertikalnog gradijenta. Na kraju, rezultat se skalira sa 255 koristeći funkciju ***scale***.

***Funkcija scale()***

int scale(int value) {

if (value <= THRESHOLD) {

return 0;

}

return 1;

}

Funkcija "***scale***" prima ulaznu vrednost (u ovom slučaju, procenjenu vrednost piksela slike) i vraća skaliranu vrednost između 0 i 1.

Ako je ulazna vrednost manja ili jednaka definisanom pragu ("***THRESHOLD***"), funkcija vraća 0. To znači da se sve vrednosti koje su manje ili jednake pragu smatraju kao 0 nakon skaliranja.

Ako je ulazna vrednost veća od praga, funkcija vraća 1. To znači da se sve vrednosti koje su veće od praga smatraju kao 1 nakon skaliranja.

Svrha ove funkcije je mapiranje ulaznih vrednosti na binarnu skalu, gde se vrednosti ispod praga smatraju kao 0, a vrednosti iznad praga smatraju kao 1.

***Serijska implementacija Prewitt algoritma***

void filter\_serial\_prewitt(int\* inBuffer, int\* outBuffer, int width, int height, int col = 0, int row = 0)

{

int lowerX, lowerY, upperX, upperY;

lowerX = (col < SKIP\_PREWITT) ? SKIP\_PREWITT : col;

lowerY = (row < SKIP\_PREWITT) ? SKIP\_PREWITT : row;

upperX = (col + width > totalWidth - (SKIP\_PREWITT)) ? totalWidth - (SKIP\_PREWITT) : col + width;

upperY = (row + height > totalHeight - (SKIP\_PREWITT)) ? totalHeight - (SKIP\_PREWITT) : row + height;

for (int currentX = lowerX; currentX < upperX; currentX++) {

for (int currentY = lowerY; currentY < upperY; currentY++) {

outBuffer[currentX + currentY \* totalWidth] = filter(inBuffer, currentX, currentY); }

}

}

Funkcija filter\_serial\_prewitt uzima ulazni bafer slike (***inBuffer***), bafer za izlaznu sliku (***outBuffer***), širinu slike (***width***) i visinu slike (***height***).

Funkcija prvo određuje granice podmatrice koju treba obraditi na osnovu trenutne kolone, reda i dimenzija slike. Donje granice (***lowerX i*** l***owerY***) se određuju tako da budu ili vrednost ***SKIP\_PREWITT*** ili vrednost trenutne kolone/red, zavisno od toga šta je veće. Gornje granice (***upperX i upperY***) se određuju tako da budu ili vrednost totalWidth - ***SKIP\_PREWITT*** ili vrednost trenutne kolone/red + širina/visina, zavisno od toga šta je manje.

Zatim se prolazi kroz svaki piksel unutar određenog opsega podmatrice. Za svaki piksel, poziva se funkcija filter koja primenjuje ***Prewitt*** operator na ulaznu sliku i dobijenu vrednost smešta u odgovarajući piksel izlazne slike u baferu ***outBuffer***.

Ova funkcija obrađuje sliku redom, piksel po pikselu, primenjujući ***Prewitt*** operator na svaki piksel unutar određenog područja, i rezultat čuva u izlaznom baferu.

***Paralelizacija***

Prolasci kroz matricu mogu biti ***paralelizovani*** putem particionisanja matrice na manje segmente na kojima se vrši obrada.

Dodatno, svaki segment može biti dalje particionisan do određene granice (definisane kao ***CUTOFF***), čime se postiže veći stepen paralelizacije. Svaki segment početne matrice je dodeljen kao jedan zadatak raspoređivaču.

void filter\_parallel\_prewitt(int\* inBuffer, int\* outBuffer, int width, int height, int col = 0, int row = 0)

{

if (min(height, width) <= CUTOFF) {

filter\_serial\_prewitt(inBuffer, outBuffer, width, height, col, row);

}

else {

task\_group t;

int taskHeight = height / 2;

int restHeight = height - taskHeight;

int taskWidth = width / 2;

int restWidth = width - taskWidth;

t.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, taskWidth, taskHeight, col, row); });

t.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, restWidth, taskHeight, col + taskWidth, row); });

t.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, taskWidth, restHeight, col, row + taskHeight); });

t.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, restWidth, restHeight, col + taskWidth, row + taskHeight); });

t.wait();

}

}

Ako dimenzije prelaze vrednost ***CUTOFF***, funkcija kreira četiri zadatka koristeći klasu task\_group kako bi paralelno obradila četiri podmatrice. Ulazni i izlazni baferi slike se dele među svim zadacima.

Svaki zadatak je odgovoran za obradu određene podmatrice, koja se dobija deljenjem trenutne matrice na polovine vertikalno i horizontalno.

Zadaci se kreiraju koristeći lambda izraze i prosleđuju se funkciji ***t.run()*** radi paralelnog izvršavanja.

Nakon kreiranja svih zadataka, poziva se funkcija ***t.wait()*** koja čeka da svi zadaci završe pre nego što se nastavi sa izvršavanjem.

***Deljenjem slike na manje podmatrice i paralelnom obradom, ova implementacija postiže bolje performanse iskorišćavanjem paralelizma na višejezgarnim ili višenitnim sistemima.***

# 4. Ispitivanje dobijenih rezultata

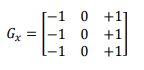
## 4.1 Fizičke karakteristike računara

Ispitivanja su vršena na računaru Lenovo IdeaPad Pro, koji sadrži AMD Ryzen 5 5600U procesor sa baznom frekvencijom od 2301 Mhz, 6 jezgara i 12 logičkih procesora.

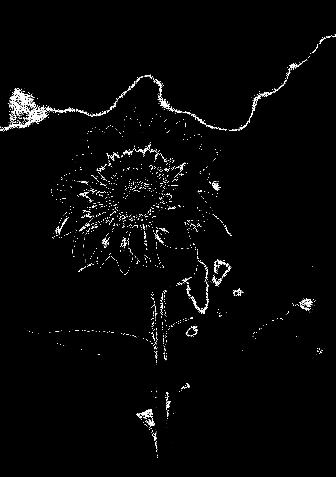
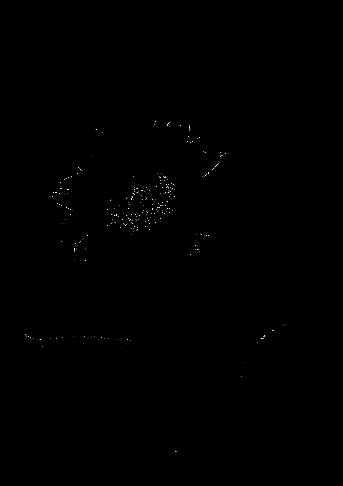
## 4.2 Testni slučajevi

### 4.2.1. Testni slučaj br. 1 – suncokret

***Veličina fotografije***: *4000 x 6000*



***Rezultati za***

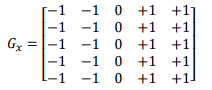
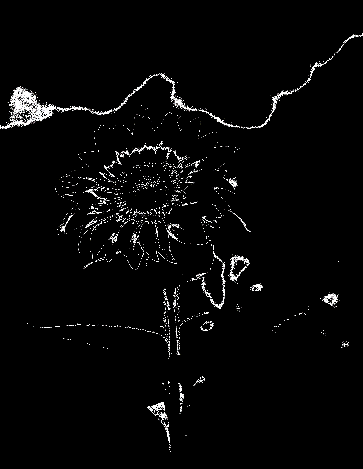
******

Ilustracija 1: Ilustracija suncokreta

Ilustracija 1: suncokret

Ilustracija 1: algoritam pretrage okoline piksela

Ilustracija 1: Prewitt algoritam



Ilustracija 1: algoritam pretrage okoline piksela

***Rezultati za***

***Zaključak:*** Prewitt operator pokazao

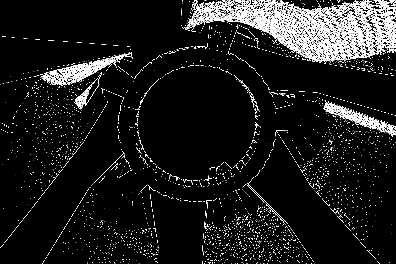
detaljnijim, nakon povećanja dimenzija

operatora. Daljim povećavanjem

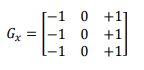
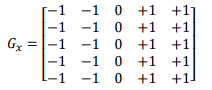
slika postaje previše detaljna.

Ilustracija 1: Prewitt algoritam

FILTER\_SIZE=3

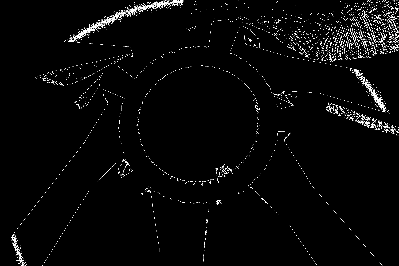


### ***C:\SIIT-druga godina\Letnji semestar\Paralelno programiranje\projekat\PP_Projekat\PP_Projekat\Windows\Test slike\color\color.bmp***4.2.3. Testni slučaj broj 2 – građevina

***Rezultati za:***

Ilustracija 2: Prewitt operater

Ilustracija 2: color.bmp

***Zaključak:***

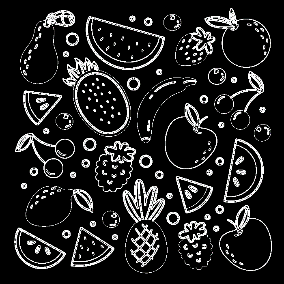
Ilustracija 2: algoritam pretrage okoline tačaka

Slika nije previše detaljna, samim time nema neke prevelike razlike između algoritma sa Prewitt operaterom i algoritma koji implementira metodu detekcije ivica.

FILTER\_SIZE=5

***Veličina fotografije:*** 3888x2592

#### 4.2.4. Testni slučaj broj 3 – vector art fotografija



Ilustracija 4:algoritam pretrage okoline tačaka

***Veličina fotografije:*** 700x700

***Zaključak:*** Prewitt algoritam preciznije

detektuje detalje, kada je u pitanju

doodle.png. Povećanjem parametra

Ilustracija 3: Prewitt operater

FILTER\_SIZE dolazi do zadebljanja

Ilustracija 3:doodle.png

Ivica, tako da je korišćen FILTER\_SIZE=5. Matrica 5x5.

## 4.3. Rezultati i vrednosti merenja

U tabeli su dati rezultati i merenja koja su vršena nad ***testnim slučajem br.2***, tj. slici dimenzija 3888 x 2592, gde je vrednost CUTOFF parametara menjana. Generalno gledano, optimalni rezultati su dobijeni kada vrednost CUTOFF parametra varira oko broja 130. Relativno slični rezultati se dobijaju kada je vrednost CUTOFF parametra u opsegu:

* 50 - 160 za algoritam sa Prewitt operaterom
* 30 – 50 i 100-120 za algoritam sa pretragom okoline svakog piksela

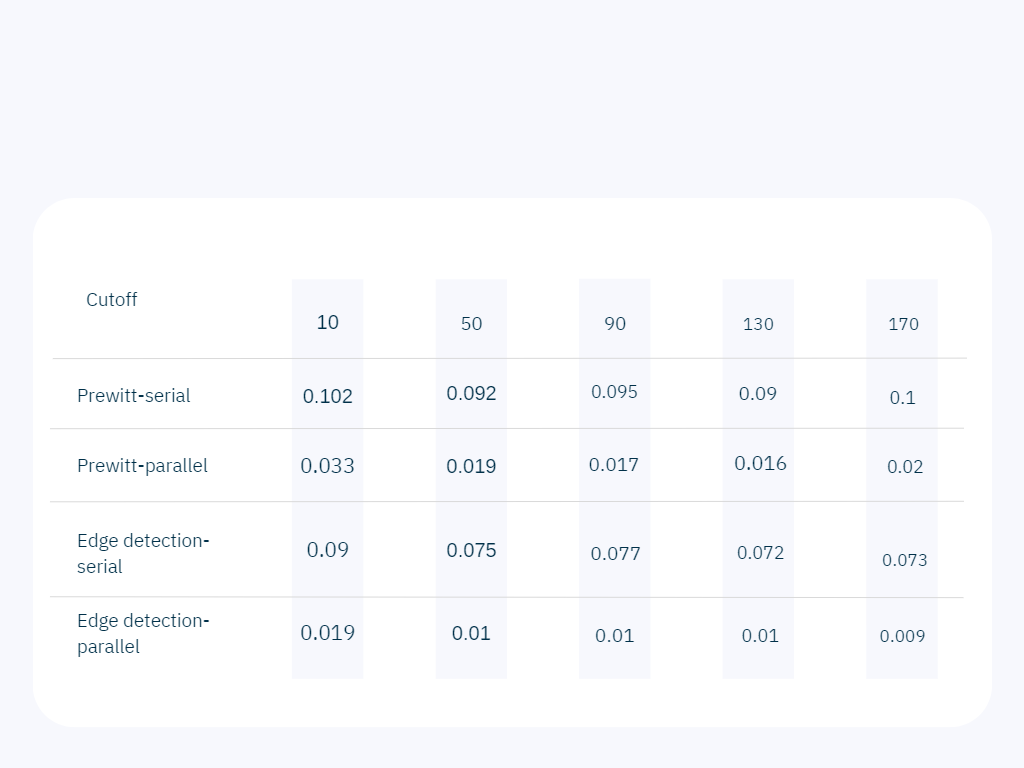
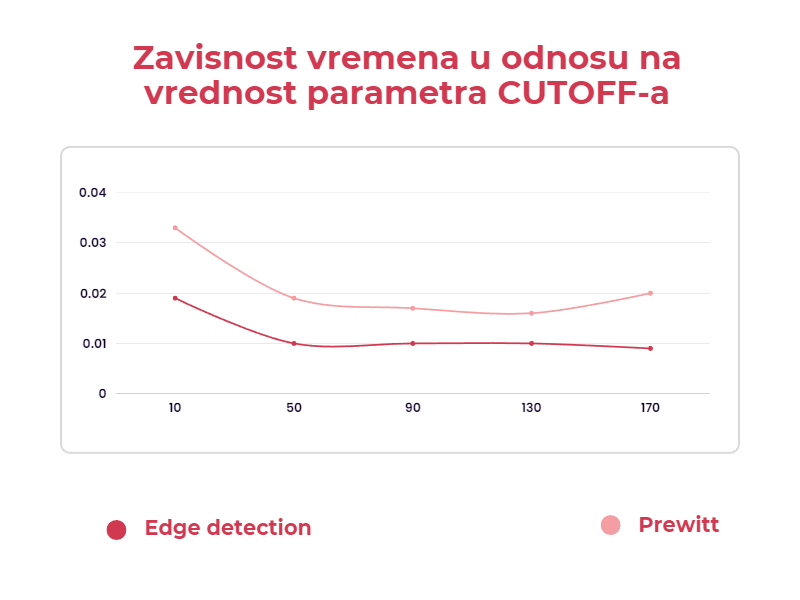
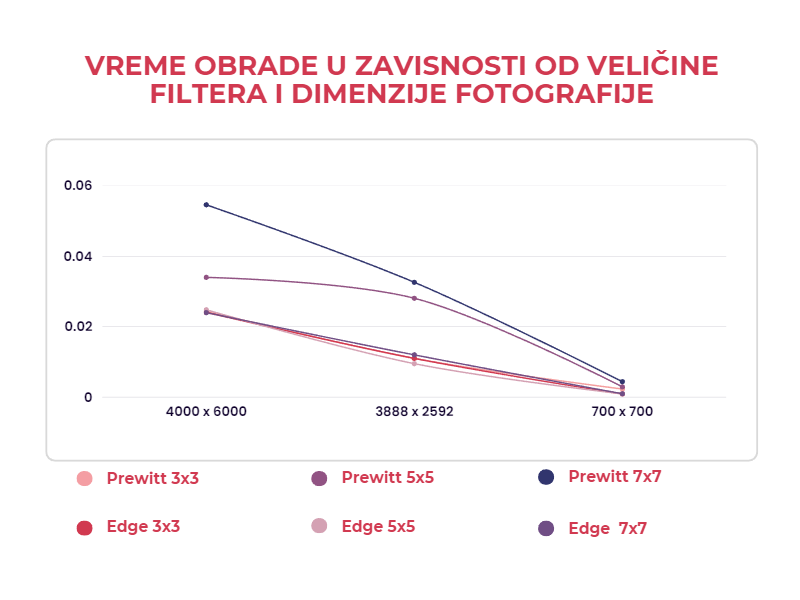
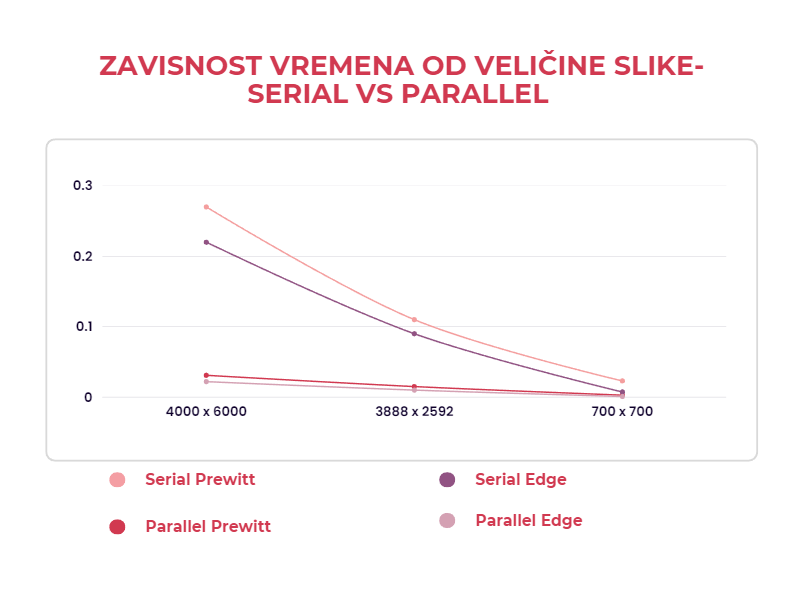


Tabela 1:Vremena obrade u zavisnosti od vrednosti CUTOFF parametra







# 5. Analiza dobijenih rezultata

Dati rezultati dobijeni su uzimanjem proseka više merenja-uglavnom su većina merenja imala manje-više približne vrednosti, dok su uglavnom uvek postojala po oko dva ekstrema gde su vrednosti značajno odstupale od prethodne medijane.

Čak štaviše, najveće ubrzanje je postignuto prilikom korišćenja malih vrednosti CUTOFF-a, paralelizacijom više paralelnih zadataka, max 12 puta brže. Međtim, gde naša implementacija ne bi bila zadovoljavajuća-pa, prilikom obrade fotografija ogromnih dimenzija, sa mnoštvom detalja.

Istraživanje je zadovoljilo postavljenje i očekivane norme-spram fizičkih karakteristika računara i veličina ulaznih parametara, njihovom varijabilnošću se u svakom slučaju došlo do optimalnog paralelnog rešenja.