УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Божица Калинић

ОПТИМИЗАЦИЈА ИЗДВАЈАЊА И ОТКЛАЊАЊА ПЕРСПЕКТИВЕ СА ФОТОГРАФИЈА САОБРАЋАЈНИХ ЗНАКОВА ПОМОЋУ ВИШЕПРОЦЕСОРСКЕ ПЛОЧЕ

мастер рад

Београд, 2021.

Ментор:

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

др Саша Малков, ванредни професор
Универзитет у Београду, Математички факултет

др Богдан Павковић, доцент
Универзитет у Новом Саду, Факултет техничких наука

Датум одбране: септембар 2021.

Свим драгим особама које су ми помогле у изради овог рада

Наслов мастер рада: Оптимизација издвајања и отклањања перспективе са фотографија саобраћајних знакова помоћу вишепроцесорске плоче

Резиме: Рад се састоји из три дела. У првом делу се помоћу клизајућих прозора димезије 32×32 и класификатора детектује саобраћајни знак на фотографији. У другом делу се помоћу неуронске мреже проналазе четири тачке, које одређују пројективно пресликавање којим се добија саобраћајни знак са отклањеном перспективом. У трећем делу се примењује пројективно пресликавање на улазну слику, чиме се добија цилјни излаз, тј. саобраћајни знак са отклоњеном перспективом.

Кључне речи: рачунарски вид, неуронске мреже, машинско учење, оптимизација, пројективна пресликавања

Садржај

Глава 1

Увод

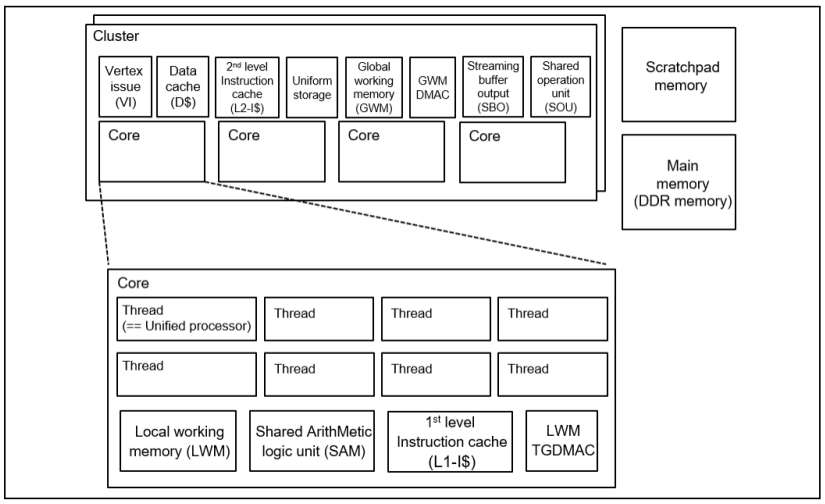
Већина саобраћајних несрећа настаје због људске грешке, непоштовања прописа, вожње под дејством опојних супстанци, непоштовања осталих учесника саобраћаја и сл. Аутономна вожња би умањила утицај људске грешке у саобраћају. Тиме би се смањио број несрећних случајева због саобраћајних несрећа, као и медицински и трошкови поправки који су последица самих удеса. Уз помоћ алгоритама рачунарског вида, можемо омогућити аутономним возилима да препознају саобраћајне знакове, пешаке, семафоре, ивице пута и сл. што би допринело сигурнијем и ефикаснијем саобраћају.

Циљ овог рада је да поједностави слику саобраћајног знака, уклањањем перспективе, и тиме да помогне у њеном даљем тумачењу. Рад се састоји из три дела. Први део рада приказује оптимизацију поступка издвајања саобраћајног знака из унапред задате слике, тако што формира минимални ограничавајући четвороугао, којим ће уоквирити део слике на којој се налази саобраћајни знак. Други део рада помоћу претходно одређеног четвороугла, који издваја саобраћајни знак, проналази барем четири пара тачака, преко којих израчунава матрицу пројективног пресликавања за уклањање перспективе. Трећи део рада помоћу матрице пројективног пресликавања оптимизује поступак трансформисања улазне слике у изазну слику са отклањеном перспективом. Сви делови рада ће бити оптимизовани уз помоћ вишепроцесорке плоче.

Глава 2

Архитектура вишепроцесорке плоче

Плоча се састоји из два процесора, који се зову *Computer Vision engine 1(CVe1)* и *Computer Vision engine 2(CVe2)*. Сваки процесор се састоји од четири језгра, а свако језгро од осам нити. Свака нит може да има функционалност или господара или слуге. Нит која има улогу господара покреће и координише рад нити које имају улогу слуге. Свако језгро мора да има барем јенду нит чија је улога господар. На слици (2.1) се налази визуални приказ архитектуре плоче коју ћемо даље продискутовати.



Слика 2.1: Архитектура плоче [1]

[1] слика је преузета из документације за Renesas плочу

Постоји више врста меморије у оквиру плоче. Свака врста меморије се може користити за читање и писање података. Њихов садржај се дели на улазни и излазни део. У улазном делу се налази слика или неки њен део који је потребно обрадити, а на излазном делу се налази резултујућа слика или неки њен део. Плоча има *DMA* јединицу, која обавља трансфер података из једног типа меморије у други. У оквиру плоче постоје наредни типови меморије:

- Први тип меморије са највећим капацитетом, али најспоријом брзином приступа читања и писања је спољна меморија (на слици означена као *DDR* меморија). Она је дељена међу свим језгрима оба процесора.

- Други тип меморије је привремена меморија (на слици (2.1) означена као *Scratchpad* меморија). Она је исто дељена међу свим језгрима оба процесора. Има мало брже време приступа читању и писању у односу на спољну меморију, али има мањи капацитет (896kB). Спољна и привремена меморија су организоване дводимензионално и могу приступити пикселима слике помоћу x и y координата.

- Трећи тип меморије је глобална радна меморија које је дељена међу свим језгрима једног процесора. Њен капацитет је $4kB$.

- Четврти тип меморије је униформна меморија. Она се користи за чување константи и дељена је међу свим језгрима једног процесора.

- Пети тип меморије је локална радна меорија (*Local Working Memory - LWM*). Свако језгро садржи своју засебну локалну радну меморију. Њен капацитет је $16kB$. С обзиром да једно језгро има осам нити, у недељеном режиму рада, свака нит располаже са $2kB$ локалне меморије.

- Последњи тип меморије су регистри. Један CVe садржи 32 тридестдвобитна регистра. Регистри су подељени према функцији на регистре за улаз, излаз, константе, адресе, статус и контролу. Регистри могу да чувају целовите вредности, као и вредности у покретном зарезу.

Нити једног језгра, осим што деле локалну радну меморију, деле и аритметичко-логичку јединицу. Међу језгрима једног процесора деле се инструкцијски кеш, кеш података, дељена операцијска јединица, стрим бафер (секвенцијално уписује податке у спољну меморију) и глобална радна меморија.

Нити сваког језгра су нумерисане бројевима од нула до седам и тим редом имају приоритет. У оквиру *Local Working Memory Thread Group Data Management and Communications* ($LWMTGDMAC$) се налазе контролни регистри. Преко њих се контролише понашање нити током процеса читања и писања у LWM.

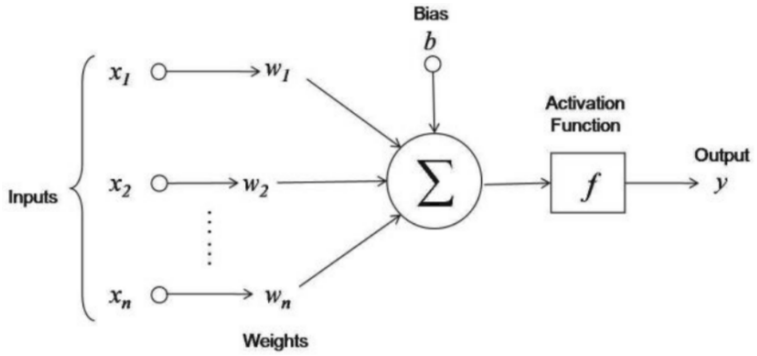
Глава 3

Конволутивне неуронске мреже

Да бих објаснила појам конвулутивне неуронске мреже, потребно је прво објаснити целине из којих се она састоји. На крају поглавља биће објашњен дизајн конвулутивне неуронске мреже преко њених појединачних слојева.

3.1 Потпуно повезана неуронска мрежа

Неуронска мрежа је састављена од основних рачунских јединица, које називамо неуронима. Један неурон формира линерану комбинацију улазних променљивих и над њима примењује нелинерану функцију. Параметри линеарне комбинације се најчешће означавају са w (називају се и тежине), а нелинерна функција са σ (назива се и активационом функцијом). На слици (3.1) се налази визуални приказ неурона.



Слика 3.1: Неурон[2]

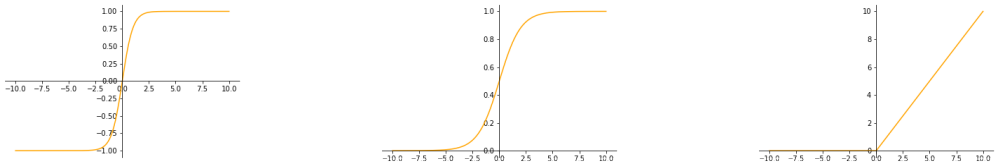
[2] https://dimensionless.in/artificial-intelligence-accelerate-the-power-with-neural-networks/

Нека су улазне вредности неурона $x_1, x_2, ..., x_n$, а придружене тежине неурону $w = (w_0, w_1, w_2, ..., w_n)$, онда неурон можемо формално записати као функцију $f_{\sigma w}(x_1, x_2, ..., x_n) = \sigma(w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n)$. Најчешћи избори активационе функције су сигмоидна функција, тангенс хиперболички и исправљачка функција *relu*.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$relu(x) = max(0, x)$$

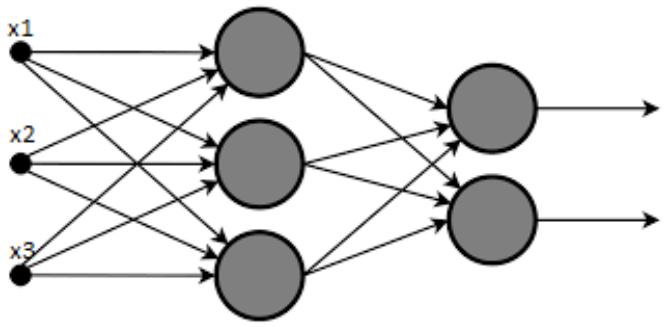


(a) тангенс хиперболички (b) сигмоидна функција (c) исправљачка функција *relu*

Слика 3.2: Графици активационих функција

Неурони потпуно повезане мреже су организовани у слојеве. Један слој чини неколико неурона, при чему сваки неурон тренутног слоја као улаз добија све излазе претходног слоја неурона. Сваки неурон тренутног слоја прослеђује свој илаз сваком неурону у наредном слоју. Међу неуронима једног слоја нема размена улаза и излаза. Назив потпуно повезан је настао због овог начина прослеђивања излаза. Сваки неурон почетног слоја добија исти улаз, уређену n-торку улазних вредности. Последњи слој потпуно повезане мреже дефинише димензију и вредност илаза мреже. Димензија је одређена бројем неурона у последњем слоју, а вредност излаза неуронске мреже је уређена m-торка излазних вредности неурона последњег слоја.

На слици (3.3) је приказана потпуно повезана неуронска мрежа са улазним слојем величине 3 и излазим слојем величине два. Улазни слој прослеђује сваку од променљивих x_1, x_2, x_3, другом слоју неуронске мреже. Други слој



Слика 3.3: Потпуно повезана мрежа[3]

садржи три неурона. Сваки неурон прослеђује свој излаз наредном слоју неурона. Последњи, трећи слој садржи два неурона чији излаз чини уређени пар који се третира као резултат неуронске мреже.

Формално се потпуно повезана неуронска мрежа f са L слојева дефише као:

$$h_0(x) = x$$

$$h_i(x) = g(W_i h_{i-1}(x) + w_{i0}), i = 1, 2, 3, .., L;$$

при чему је x вектор улазних вредности, W_i матрица, чија j-та врста представља вредности парамтера j-тог неурона у i-том слоју неурона, w_{i0} представља вектор вредности слободног параметара у i-том слоју неурона, а g нелинеарну активациону функцију.

Потпуно повезана неуронска мрежа може да се користи за регресију и за класификацију. За регресију резултујућа n-торка неуронске мреже представља предвиђену вредност тражених атрибута. За класификацију се на неуронску мрежу додаје још један слој који користи функцију

$$softmax(x_1, x_2, ..., x_n) = (\frac{e^{x_1}}{\sum_{i=1}^{n} x_i}, \frac{e^{x_2}}{\sum_{i=1}^{n} x_i}, ..., \frac{e^{x_n}}{\sum_{i=1}^{n} x_i}),$$

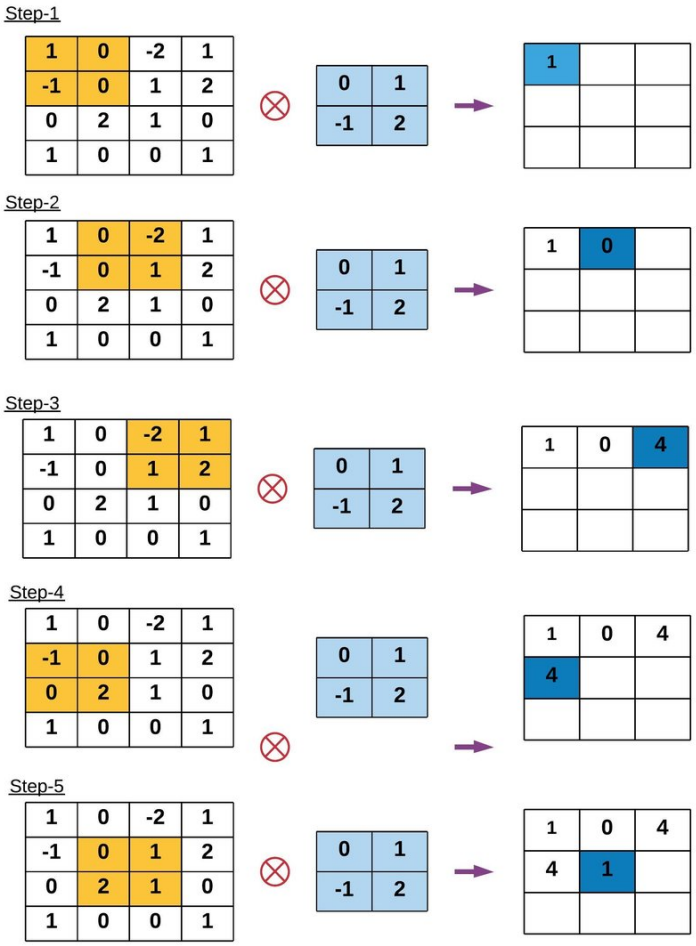
а резултујући вектор представља вектор вероватноћа, где i-та координата предстваља вероватноћу да улаз припада i-тој класи. Коначна процена је да улаз припада класи i, где је i координата чија је вредност највећа.

[3] https://en.wikipedia.org/wiki/Artificial_neural_network

3.2 Конволуција

Дефиниција 3.2.1 (Конволуција) *Конволуција у дводимензином и дискретном случају је операција која се примењује на две матрице $M(m \times n)$ и $F(p \times q)$, где за резултујућу матрицу $N((m-p+1) \times (n-q+1))$ важи*

$$N_{ij} = \sum_{k=1}^{p} \sum_{l=1}^{q} M_{(i+k)(j+l)} F_{kl};\ i = 1, .., (m-p+1), j = 1, 2, ..., (n-q+1)$$



Слика 3.4: Пример конволуције[4]

Матрица M се третира као улаз на којој се примењује матрица F, коју зовемо филтер (кернел).

[4] https://www.researchgate.net/figure/Illustrating-the-first-5-steps-of-convolution-operation_fig5_337401161

У неким случајевима желимо да улазна и резултујућа матрица конволуције буду истих димензија. У ту сврху се улазна матрица проширује *padding*-ом, где се на ободу матрице додају нуле или неки други бројеви.

Конволуције се користе у обради дигиталних слика, да би извукле неке корисне податке из слике или да би над сликом извршиле неку трансформацију. Следеће матрице представљају примере неких познатих филтера.

$$G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \qquad \text{(Собелов филтер)}$$

$$G = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \qquad \text{(Гаусов филтер)}$$

$$L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad \text{(Лапласов филтер)}$$

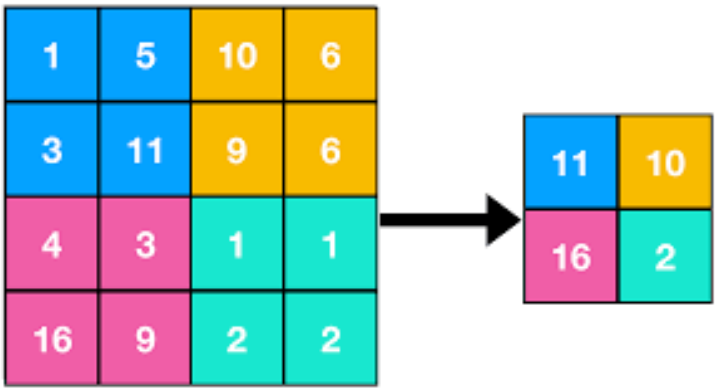
Собелов и Лапласов филтер се користе за детектовање ивица слике. Гаусов филтер се користи за замагљивање слике.

3.3 Агрегација

Агрегација је функција која се примењује над матрицом, да би се смањила њена димензија и извукли само најбитнији подаци. Примењује се редом по блоковима матрице одређених димензија и са одређеним кораком преласка на следећи блок. Сваком таквом блоку агрегација додељује један број. Најчећи избори додељивања броја су просек или максимум блока матрице.

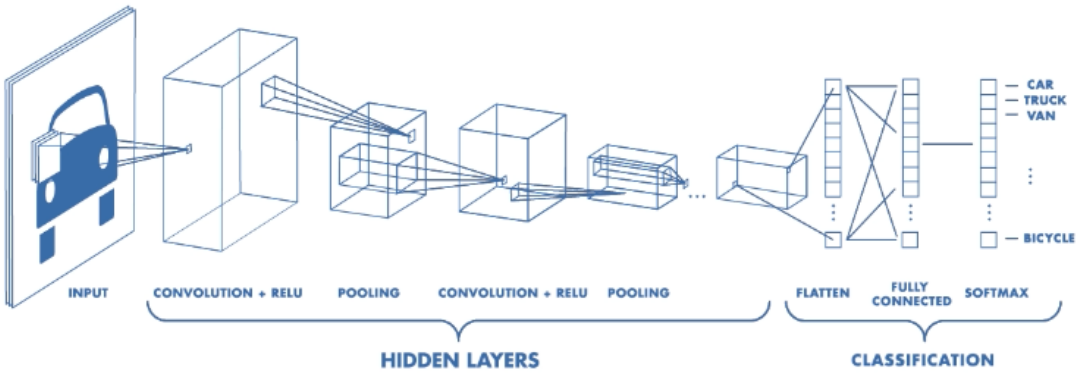
3.4 Архитектура конволутивне неуронске мреже

Конволутивне неуронске мреже се састоје из неколико смењиваних слојева конволуције и агрегације, при чему може да се понови иста врста слоја више



Слика 3.5: Пример агрегације по 2x2 блоковима са функцијом максимума и помарејем 2 по врсти и по колони [5]

пута за редом, на које се на крају додаје потпуно повезана мрежа. На излазе конволутивних слојева се примењује нелинеарна активациона функција. Конволутивне мреже третирају филтере, који се корсите у конволуцији, као параметре које треба да науче. Служе за обраду дигиталних сигнала и у тој области су се показале јако успешне. На слици (3.6) је приказана архитектура конволутивне неуронске мреже.



Слика 3.6: Визуална репрезентација архитектуре конволутивне неуронске мреже[6]

[5] https://iq.opengenus.org/pooling-layers/
[6] https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html

3.5 Евалуација модела класификације и регресије

Током обучавања модела располажемо лабелираним подацима за тренирање и тестирање. За све податке знамо тражене циљне вредности. Модулу дајемо само податке за тренирање и њихове циљне вредности. Минимизацијом функције грешке модел одређује вредности својих параметара. Квалитет модела се гледа на основу података за тестирање, којима модел током тренирања није имао приступ. У зависности од тога колико добро модел процењује циљне вредности података за тестирање, можемо да дамо процену квалитета модела.

У случају класификације, мера квалитета се најчешће одређује на основу матрице конфузије. Нека постоји n класа. Матрица конфузије је квадратна матрица C димензије $n \times n$, где је C_{ij} предствља број инстанци класе i које су класификаоване да припадају класи j, $i = 1, ..n; j = 1, ..n$. Тачност модела класификације је:

$$Acc = \frac{\sum_{i=1}^{n} C_{ii}}{\sum_{j=1}^{n} \sum_{k=1}^{n} C_{jk}}.$$

У случају бинарне класификације, једну класу зовемо позитивном, а другу негативном. Уводе се следеће ознаке: $TP = C_{11}$ (стварно позитиве), $FN = C_{12}$ (лажно негативне), $FP = C_{21}$ (лажно позитивне) и $TN = C_{22}$ (стварно негативне). Тада тачност у случају бинарне класификације можемо да изразимо као: $Acc = \frac{TP+TN}{TP+TN+FP+FN}$. Тачност можемо да схватимо као вероватноћу да у просечном случају добро класфиикујемо инстанце. Циљ је да тачност буде што ближа јединици. Међутим, испоставља се да тачност није довољна добра мера квалитета модела у случају неравномерно распоређених инстанци класа. На пример, ако од 10 000 инстанци је само једна негативне класе, а све остале инстанце су позитивне класе, довољно тачан модел се постиже тако што за сваку инстанцу кажемо да је позитивна. Тиме постижемо тачност од 0.9999. Из тог разлога се додатно разматра F_1 мера која се дефинише преко прецизности $Prec = \frac{TP}{TP+FP}$, и одзива $Rec = \frac{TP}{TP+TN}$, као $F_1 = 2\frac{RecPrec}{Prec+Rec}$. Циљ је да тачност, одзив, прецизнист и F_1 мера буду близу вредности један. Такође, треба посматрати обрнуту варијанту одзива и прецизности, где се посматра негативна класа.

У случају регресије за функцију грешке се најчешће користе средње ква-

дратна грешка: $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$, и средња апсолутна грешка: $MAE = \frac{1}{N}\sum_{i=1}^{N} abs(y_i - f(x_i))$, при чему је N укупан број инстанци, y_i стварна вредност циљне променљиве инстанце i, а $f(x_i)$ предвиђена вредност модела за инстанцу i.

Глава 4

Оптимизација издвајања саобраћајног знака

За издвајање саобраћајног знака користила сам метод клизајућих прозора уз класификатор.

Метод клизајућих прозора као улаз користи пирамидално скалирање. Пирамидално скалирање формира низ слика. Прва слика низа је оригинал, а следећа настаје скалирањем тренутне слике на дупло мању димензију. Овај поступак се понавља, све док не достигнемо унапред одређен минимум за димензију слике. Одабрала сам да тај минимум буде 32×32.

Метод клизајућих прозора користи прозор фиксне димензије, у мом случају димензије 32×32, који помера кроз сваку слику улазног низа пирамидално скалираних слика. Кретање прозора по сликама се врши редом кретањем са одређеним кораком по врсти, а затим преласком на наредну врсту, при чему се обично врши и неки корак преласка на следећу врсту, где се прескаче одређен број врсти. Чести избори за корак су 4 или 8. За сваку позицију прозора на слици се позива класификатор. Улога класификатора је да да одговор на питање: „Да ли се овде налази тражени објекат?". У случају да је одговор потврдан, тренутна позиција клизајућег прозора нам даје процену локације траженог објекта на слици.

Класификатор који сам користила је била неуронска мрежа. Податке за тренирање и тестирање класификатора сам добила коришћењем базе слика саобраћајних знакова *The German Traffic Sign Recognition Benchmark (GTSRB)*. База података је организована у 42 фолдера, где сваки фолдер садржи више слика једне врсте саобраћајног знака, као и *csv* фајл у ком се

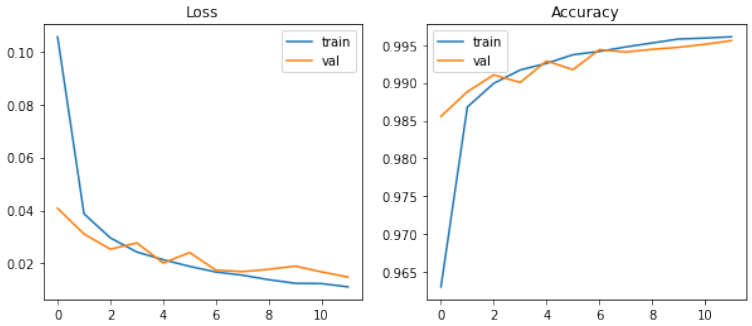
налазе подаци везани за слике тог фолдера. За сваку слику одговарајућег фолдера *csv* фајл садржи одговарајућу класу којој тај саобраћајни знак припада, вредности координата горњег левог темена и доњег десног темена, као и вредности ширине и висине ограничавајућег правоугаоника саобраћајног знака те слике. На основу тих података, за сваки положај клизајућег прозора сваке слике, могу да одредим да ли тај прозор садржи саобраћајни знак или не. Нека је (i, j) позиција горњег левог темена клизајућег прозора, а $(x_1, y_1), (x_2, y_2)$ координате горњег левог и доњег десног темена ограничавајућег правоугаоника, које су забележене у табели. Тада можемо да тврдимо да се на тренутној позицији клизајућег прозора налази саобраћајни знак, ако важи: $i <= x_1$ *and* $x_2 < i + 32$ *and* $j <= y_1$ *and* $y_1 < j + 32$. Ако тај услов не важи, можемо рећи да се на тренутној позицији клизајућег прозора не налази саобраћајни знак. Овде смо ограничени тачношћу података забележених у *GTSRB* бази. С обзиром да користимо статистички модел, не тврдимо са 100% тачношћу да ће класификатор увек прецизно дати одговор. У прилогу А.1 се налази кôд за генерисање података за тренирање и тестирање.

На основу претходно генерисаних података сам обучила класификатор за препознавање саобраћајног знака. У прилогу А.2 се налази кôд за обучавање и евалуацију класификатора саобраћајног знака.

За оптимизацију неуронске мреже, која се користи као класификатор у методи клизајућих прозора, је коришћен *CNN Frontend*. *CNN Frontend* је софтвер који аутоматизује поступак конверзије неуронске мреже у кôд, који се може извршити на *Renesas RCar* плочи. Као улаз *CNN Frontend* захтева *onnx* фајл, формат који описује модел машинског учења, као и *config* фајл, који описује улазни, као и излазни тип података мреже. Софтвер оптимизује операције које се јављају у конволутивним неуронским мрежама, попут конволуције и агргације, преко нити. Има способност да врши више различитих конволуција у истом временском периоду. Такође се убрзање постиже и аритметичко-логичком јединицом, као и кеш меморијом.

Време потребно да би неоптимизована мрежа обрадила један улаз износи $0.0365s$, док је време потребно да би оптимизована мрежа обрадила један улаз $842 \cdot 10^{-6}s$.

У прилогу А.3 се налази кôд, који приказује начин коришћења методе клизајућих прозора за детекцију саобраћајног знака уз оптимизовану неуронску мрежу која се кортисти као класификатор.



(а) Етапе тренирања

	precision	recall	f1-score	support
0	1.00	1.00	1.00	104885
1	0.98	0.99	0.98	17626
accuracy			1.00	122511
macro avg	0.99	0.99	0.99	122511
weighted avg	1.00	1.00	1.00	122511

(b) Евалуација модела

Model: "sequential_2"

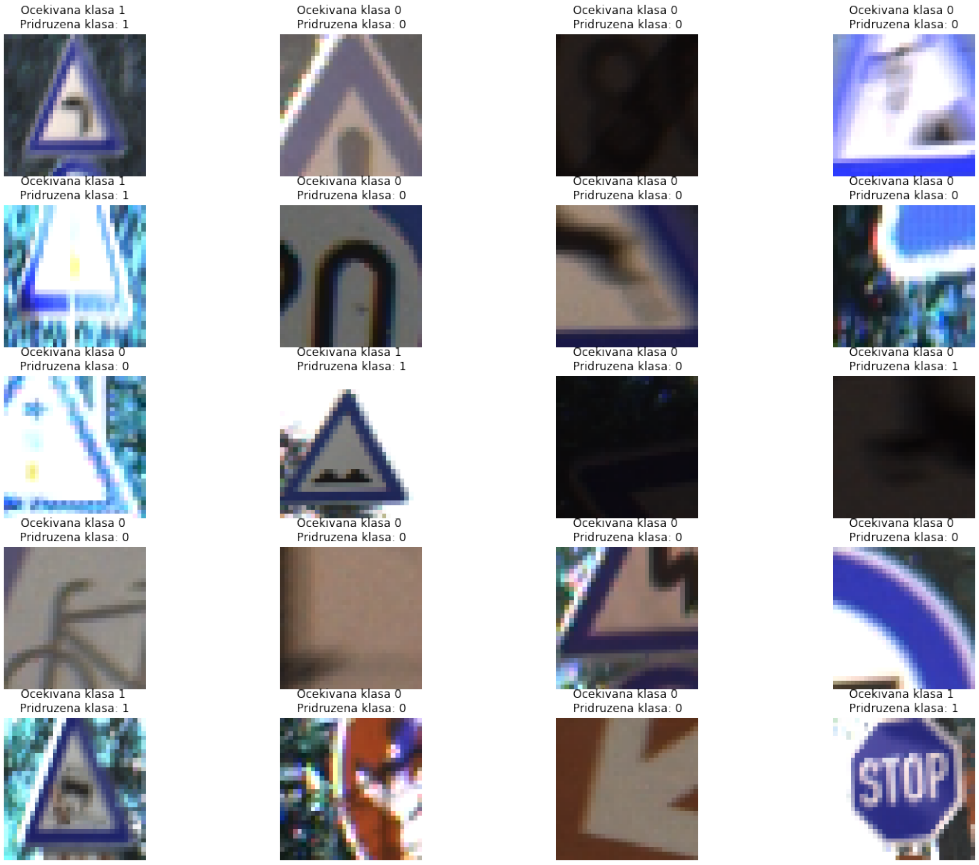
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 32, 32, 8)	608
max_pooling2d_6 (MaxPooling2	(None, 16, 16, 8)	0
conv2d_7 (Conv2D)	(None, 16, 16, 16)	3216
max_pooling2d_7 (MaxPooling2	(None, 8, 8, 16)	0
conv2d_8 (Conv2D)	(None, 8, 8, 32)	12832
max_pooling2d_8 (MaxPooling2	(None, 4, 4, 32)	0
flatten_4 (Flatten)	(None, 512)	0
dense_6 (Dense)	(None, 128)	65664
dropout_4 (Dropout)	(None, 128)	0
flatten_5 (Flatten)	(None, 128)	0
dense_7 (Dense)	(None, 128)	16512
dropout_5 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 2)	258

Total params: 99,090
Trainable params: 99,090
Non-trainable params: 0

(c) Архитектура неуронске мреже

Слика 4.1: Модел класификатора

Очекивано време извршавања оптимизације детекције саобраћајног знака зависи од величине улазне слике, као и брзине пребацивања и скидања података са плоче. Ако је улазна слика димензије $n \times m$, број различитих положаја прозора за које се позива класификатор је $\frac{n \cdot m}{16}$. Пошто за сваку позицију прозора позивамо класификатор, множимо то време са временом потребним да класификатор обради једну слику, тј. $\frac{n \cdot m}{16} 842 \cdot 10^{-6} s$. Да би добили коначну процену времена извршавања, за сваки позив класификатора, морамо да урачунамо и време потребно да се слика димензије 32×32 пребаци на плочу, као и да се скине фајл са плоче у коме је забележена резултујућа вредност модела. С обзиром да је улаз у мрежу константне величине, као и да излазни фајл садржи само две вредности, можемо третирати време преноса података као неку константу. Множењем $\frac{n \cdot m}{16} 842 \cdot 10^{-6} s$ са том константом добијамо коначно време извршавања. Проблем оптимизације првог дела рада је у томе



Слика 4.2: Примери класификације

што је морао да се обради прозор по прозор улазне слике. Није постојала могућност да се за сваки прозор пусти нит, која би позвала извршавање мреже на плочи. Тиме би природније решење за први део рада, била стандардна оптимизација, где се за сваки положај прозора пушта засебна нит која ће да позивана неоптимизован модел. У тој варијанти оптимизације не зависимо од брзине преноса података, што је предност.

У прилоу А.4 се кôд, који садржи пример оптимизације клизајућих прозора без коришћења вишепроцесорске плоче.

Глава 5

Одређивање пројективног пресликавања

5.1 Основни појмови

Дефиниција 5.1.1 (Хомогене координате) *Хомогене координате тачке $M(x, y)$ афине равни $\mathbb{R}^2$ су било која тројка $(x_1 : x_2 : x_3)$, x_1, x_2, $x_3 \in \mathbb{R}$, за коју важи да је $\frac{x_1}{x_3} = x$, $\frac{x_2}{x_3} = y$ и $x_3 \neq 0$.*

Дефиниција 5.1.2 (Бесконачно далека права) *Права $p_\infty : x_3 = 0$ се назива бесконачно далеком правом. Свака тачка хомогених координата $(x : y : 0)$, $x, y \in \mathbb{R}$ јој припада и назива се бесконачно далеком тачком.*

Дефиниција 5.1.3 (Реална пројективна раван) *Реална пројективна раван у ознаци $\mathbb{R}P^2 = \{(x_1 : x_2 : x_3) |\ x_1, x_2, x_3 \in \mathbb{R};\ x_1^2 + x_2^2 + x_3^2 \neq 0\}$*

Дефиниција 5.1.4 (Пројективно пресликавање) *Пројективно пресликавање $f : \mathbb{R}P^2 \to \mathbb{R}P^2$ које пресликава тачку $M(x_1 : x_2 : x_3)$ у тачку $N(y_1 : y_2 : y_3)$ је оно пресликавање одређено формулом*

$$\lambda \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \exists \lambda \in \mathbb{R}, \lambda \neq 0, det(P_{i,j}) \neq 0.$$

Матрицу P_{ij} називамо матрицом пројективног пресликавања f.

Пројективно пресликавање је индуковано линераним пресликавањем векторског простора $\mathbb{R}^3$. Инверзно пресликавање f^{-1} пројективног преслика-

вања f је одређено инверзом матрице P_{ij}. Композиција пројективних пресликавања је одређена производом матрица тих пројективних пресликавања. За пројективна пресликавања важи да могу да пресликају произвољан четвороугао у произвољан четвороугао. Тиме пројективна пресликавања не разликују ромб од квадрата или од делтоида или од неког неконвексног четвороугла. Ради циља отклањања перспективе, можемо пронаћи пројективно пресликавање које ће произвољан четвороугао у коме се налази саобраћајни знак пресликати у квадрат са теменима хомогених координата $(0:0:1), (1:0:1), (1:1:1), (0:1:1)$.

Теорема 5.1.1 (Основна теорема Пројективне геометрије) *Постоји јединствено пројективно пресликавање које четири тачке A_1, B_1, C_1, D_1 у општем положају слика редом у четири тачке A_2, B_2, C_2, D_2, у општем положају.*

5.2 Одређивање произвољног пројективног пресликавања

Пројективно пресликавање h које тачке A_1, B_1, C_1, D_1, у општем положају слика редом у тачке A_2, B_2, C_2, D_2, у општем положају, је одређено као последица теореме 5.1.1 са $h = f^{-1} \circ g$, где је f пројективно пресликавање које тачке A_2, B_2, C_2, D_2 редом слика у $(1:0:0), (0:1:0), (0:0:1), (1:1:1)$, а g пројективно пресликавање које тачке A_1, B_1, C_1, D_1 редом слика у $(1:0:0), (0:1:0), (0:0:1), (1:1:1)$, тј. ако су F, G редом матрице пројективних пресликавања f, g, онда је матрица траженог пројективног пресликавања $H = F^{-1} \cdot G$.

На основу претходног, да бисмо одредили матрицу произвољног пројективног пресликавања, довољно је да решимо систем једначина дефинисан пресликавањем произвољне четири тачке у општем положају у неке четири фиксне тачке у општем полажају, нпр. $(0:0:1), (0:1:1), (1:1:1), (1:0:1)$.

Означимо матрицу пројективног пресликавања са

$$P = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix},$$

а проивољне четири тачке у општем положају као $A(x_0, y_0)$, $B(x_1, y_1)$, $C(x_2, y_2)$, $D(x_3, y_3)$. Њихове хомогене координате можемо добити додавањем јединице као треће координате.

Тада је систем линеарних једначина дефинисан преко следећих једнакости:

$$t_0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$$

$$t_1 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

$$t_2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix}$$

$$t_3 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_3 \\ y_3 \\ 1 \end{pmatrix},$$

при чему важи $t_0, t_1, t_2, t_3 \neq 0$ и пошто су тачке A, B, C и D у општем положају, важи да ни које три тачке нису колинеарне, тј. да је површина троуглова које свака тројка тачака одређује различита од 0,

$$\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} \neq 0$$

$$\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \neq 0$$

$$\begin{vmatrix} x_0 & y_0 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \neq 0$$

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \neq 0.$$

Извођење решења система линеарних једначина се налази у прилогу В. На основу тог извођења, добијамо решење:

1. $x_1 - x_0 = 0$

$$t_0 = 1$$

$$t_1 = t_0 \frac{(x_2 - x_0)(y_1 - y_3) + (y_2 - y_1)(x_3 - x_0)}{(y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0)}$$

$$t_2 = t_1(x_2 - x_0)(\frac{y_2 - y_0}{(y_1 - y_0)(x_2 - x_0)} - \frac{y_3 - y_0}{(y_1 - y_0)(x_3 - x_0)})$$

$$t_3 = \frac{t_2(x_3 - x_0)}{x_2 - x_0}$$

$$a = \frac{t_2}{x_2 - x_0}$$

$$d = \frac{t_2}{x_2 - x_0} - \frac{t_1(y_2 - y_0)}{(y_1 - y_0)(x_2 - x_0)}$$

$$g = \frac{t_2 - t_0}{x_2 - x_0} - \frac{(t_1 - t_0)(y_2 - y_0)}{(y_1 - y_0)(x_2 - x_0)}$$

$$b = 0$$

$$e = \frac{t_1}{y_1 - y_0}$$

$$h = \frac{t_1 - t_0}{y_1 - y_0}$$

$$c = -ax_0 - by_0$$

$$f = -dx_0 - ey_0$$

$$i = t_0 - gx_0 - hy_0.$$

2. $x_1 - x_0 \neq 0$

$$
\begin{aligned}
t_1 &= 1\\
t_2 = t_1(&-(x_3-x_0)(y_2-y_0)(x_1-x_0)+(x_3-x_0)(y_1-y_0)(x_2-x_0)+\\
&(x_2-x_0)(y_3-y_0)(x_1-x_0)-(x_2-x_0)(y_1-y_0)(x_3-x_0))/\\
&((x_1-x_0)((y_3-y_0)(x_1-x_0)-(y_1-y_0)(x_3-x_0)))\\
t_3 &= t_2((y_3-y_0)(x_1-x_0)-(x_3-x_0)(y_1-y_0))/\\
&(-(x_2-x_0)(y_1-y_0)+(y_2-y_0)(x_1-x_0))\\
t_0 = (&-t_2(x_1-x_0)((y_3-y_0)(x_1-x_0)-(y_1-y_0)(x_3-x_0))\\
&+t_3(x_1-x_0)((y_2-y_0)(x_1-x_0)-(y_1-y_0)(x_2-x_0))+\\
&t_1((y_3-y_0)(x_1-x_0)(x_2-x_0)-(y_1-y_0)(x_3-x_0)(x_2-x_0)-\\
&(y_2-y_0)(x_1-x_0)(x_3-x_0)+(y_1-y_0)(x_2-x_0)(x_3-x_0)))\\
&/((y_3-y_0)(x_1-x_0)(x_2-x_1)-(y_1-y_0)(x_3-x_0)(x_2-x_1)-\\
&(y_2-y_0)(x_1-x_0)(x_3-x_1)+(y_1-y_0)(x_2-x_0)(x_3-x_1))\\
b &= \frac{t_2(x_1-x_0)}{-(x_2-x_0)(y_1-y_0)+(y_2-y_0)(x_1-x_0)}\\
e &= \frac{t_2(x_1-x_0)-(x_2-x_0)t_1}{(y_2-y_0)(x_1-x_0)-(y_1-y_0)(x_2-x_0)}\\
h &= \frac{(t_2-t_0)(x_1-x_0)-(x_2-x_0)(t_1-t_0)}{(y_2-y_0)(x_1-x_0)-(y_1-y_0)(x_2-x_0)}\\
a &= \frac{-b(y_1-y_0)}{x_1-x_0}\\
d &= \frac{t_1-e(y_1-y_0)}{x_1-x_0}\\
g &= \frac{t_1-t_0-h(y_1-y_0)}{x_1-x_0}.
\end{aligned}
$$

5.3 Одабир улазних и излазних тачака пројективног пресликавања

Сада имамо потребан математички алат преко ког можемо да отклонимо перспективу саобраћајног знака. На нама је да одаберемо четири тачке улазне слике и четири излазне тачке за пресликавање.

Користила сам базу података *The German Traffic Sign Recognition Benchmark(GTSRB)* за примере улазних слика за које желимо да уклонимо пер-

спективу. База је намењена за класификацију саобраћајних знакова. Садржи 42 фолдера. Сваки фолдер садржи слике једног типа саобраћајног знака и *csv* фајл у коме се за сваку слику налазе координате ограничавајућег четвороугла и тип саобраћајног знака. У мом раду нису били битни подаци о конкретном типу саобраћајног знака, већ ме је занимао његов геометријски облик до на пројекцију. Међу саобраћајним занковима се јављају облици: круг, квадрат, троугао и осмоугао. Улазне тачке бирамо на основу типа облика саобраћајног знака.

Продискутујмо одабир тачака на основу облика:

1. *круг*: Одабраћемо тачке које настају као пресек тангенти на најсевернијој, најјужнијој, најзападнијој и најисточнијој тачки;

2. *квадрат*: Темена квадрата одређују тражене тачке;

3. *осмоугао*: Само стоп знак има осмоугли облик. Пресеком страна осмоугла које су паралелне или нормалне реду слова на знаку добијамо четири тражене тачке;

4. *троугао*: Троугао је представљао најтежи случај у избору тачака. Троугао има три темена, али нама су потребне четири тачке за пројетивно пресликавање. Идеја је била да искористимо једну основицу и њену тетиву да би фомирали четвороугао у коме се налази троугао. Постоје саобраћајни знакови где је једна од основица троугла на дну слике, а други тип има једну основицу на врху слике. У зависности да ли једна основица на врху/дну слике, тражену основицу сам бирала тако да њу одређују два најсевернија/најјужнија темена, а тетиву добијамо спајањем средине те основице са преосталим теменом. Тражени четвороугао добијамо транслирањем основице низ тетиву и транслирањем тетиве до темена одабране основице.

Тачке излаза бирамо тако да се саобраћајни знак простире по целој излазној слици. Тражене тачке ће баш бити ћошкови излазне слике, тј. $(0, 0)$, $(w, 0)$, (w, h), $(0, h)$, где је w дужина, а h висина излазне слике.

(a) круг (b) квадрат (c) доњи троугао (d) осмоугао (e) горњи троугао

Слика 5.1: Примери одабира улазних тачака

5.4 Алгоритам одабира четири улазне тачке

Иако смо претходно описали тачан поступак одабира тачака, на основу познатих ивица саобраћајног знака, са реалним подацима ситуација није била толико једноставна. Током израде кôда, прво што сам пробала је било да директно алгоритамски решим овај проблем. Било је потребно на слици означити само ивице саобраћајног знака. Тиме бисмо преко претходно описаног одабира тачки лако решили проблем. У ту сврху сам користила функције библиотеке *OpenCV*, намењене за обраду дигиталних слика, да бих детектовала све ивице слике. На основу њим сам пронашла затворене контуре и издвојила само оне контуре које ималу тражени број темена, у зависности од облика соабраћајног знака. Међутим, због пристуства позадине, дешавало се да постоји више оваквих контура. На сликама осим саобраћајног знака се налазе и дрвећа, зграде, небо, бандере и сл. Покушала сам да у ситуацији где има више таквих контура, извучем ону са највећом површином. Сви ови покушаји су давали јако непрецизне резултате. Квалитет слика базе није био иделан за директно решавање проблема.

Из ових разлога, одабрала сам да проблем решим пробабилистички. Одлучила сам да направим неуронску мрежу, која на основу улазне слике, одређује координате четири тачака, које би представљале улаз за пројективно пресликавање.

У ту сврху, прво што сам пробала је било да пронађем готове податке за тренирање неуроснке мреже. Била сам јако оптимистична. Користила сам *Google dataset search* и *Kaggle*, тражећи било какву базу података намењену пројективним пресликавањима. Међутим, нисам имала среће. Нисам успела да нађем потребну базу података. Схватила сам да ћу морати сама да је направим.

Иницицијално сам ручно за 1050 слика базе *GTSRB* одредила координате

четири тражених тачака (по 210 слика за сваки од базних облика саобраћајног знака), записивајући вредности у *csv* табелу. Како у бази података има око 50 000 слика, не би било практично ручно проћи кроз све слике базе, већ има више смисла аутоматизовати поступак припреме података за тренирање.

За неуронску мрежу сам одабрала да све улазне слике буду димензија 32×32. У бази су димензије слика варирале од 28×28, до 250×250. Њиховим скалирањем до тражене димензије сам омогућила да све могу бити третиране као улаз за неуронску мрежу.

Ради генерисања података, искористила сам иницијалних 1050 обрађених слика, тако што сам сваку слику покушала да пројективно пресликам на различите начине. За сваку слику сам изгенерисала 200 рандом пројективних пресликавања, у које не спадају скалирања, јер ће њихово дејство бити неутрализовано током предпроцесирања слика за улаз у неуронску мрежу, где ће све слике бити склиране на 32×32. Пошто пројективна пресликавања сликају произвољан четвороугао у произвољан четвороугао, оригинална слика која је правоугаоног облика може постати неки произвољан четвороугао. Због тога било је потребно од резултујуће слике направити праоугаони исечак који садржи саобраћајни знак и његове тражене четири тачке. Означимо темена произвољног четвороугла са A, B, C, D, при чему је теме A, тачка са најмањом y координатом, а ако има две такве тачке, онда она са најмањом x координатом. Остала темена су означена кретањем по граници четвороугла у позитивном смеру. Његов правоугаони исечак је $A_1B_1C_1D_1$, при чему важи:

$$A_1.x = max(A.x, D.x)$$

$$A_1.y = max(A.y, B.y)$$

$$C_1.x = min(B.x, C.x)$$

$$C_1.y = min(D.y, C.y)$$

$$B_1.x = C_1.x$$

$$B_1.y = A_1.y$$

$$D_1.x = A_1.x$$

$$D_1.x = C_1.y.$$

Преко горе наведеног начина сам за сваку изгенерисану слику пронашла правоугаони исечак. Икористићемо пројективно пресликавање P за генерисање

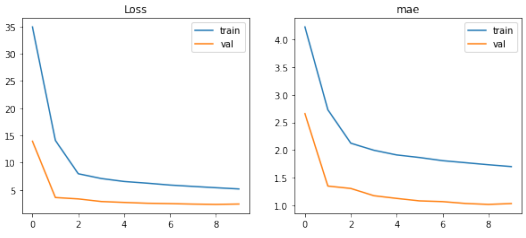
нове слике, ако за координате тражених тачака важи да се њихове слике налазе у унтрашњости или граници правоугаоног исечка излане слике. Одабрала сам да изгенеришем нове слике на основу фолдера базе *GTSRB*: 00000 (садржи кружни знак), 00011 (доњи троугао), 00012 (квадрат), 00013 (горњи троугао), 00014 (осмоугао). У прилогу А.5 се налази кôд за генерисање нових слика.

Овако изгенерисане податке сам искористила за обучавање неуронске мреже. Модел неуронске мреже је описан сликом 5.2:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 64)        4864
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        102464
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 64)          102464
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               131200
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
flatten_3 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 8)                 1032
=================================================================
Total params: 358,536
Trainable params: 358,536
Non-trainable params: 0
_________________________________________________________________
```

Слика 5.2: Модел

У прилогу А.6 се налази кôд за тренирање неуронске мреже.



Слика 5.3: Етапе тренирања мрежа

За дату улазну слику тражено пројективно пресликавање P добијамо као композицију $P = H \cdot F$, где је F пројективно пресликавање које тачке одре-

Слика 5.4: Примери проналажења улазних тачки

ђене науронском мрежом *FourPointFinder* за дату улазну слику пресликава у тачке $(0,0),(1,0),(1,1),(0,1)$, а H пројективно пресликавање које тачке $(0,0),(1,0),(1,1),(0,1)$ прелсикава у $(0,0),(32,0),(32,32),(0,32)$.

Неуронску мреже за одређивање кооридната тачака сам оптимизовала уз помоћ софтвера *CNN Frontend*. Време потребно да би неоптимизована мрежа обрадила једну улазну слику је $0.2288s$. Време потребно да би оптимизована мрежа обрадила једну улазну слику је $1405 \cdot 10^{-6}s$. С обзиром да се за целу слику позива неуронска мрежа, у односу на оптимизацију клизајућих прозора, оптимизација мреже за одређивање координата тачака је представљала доста природнији избор за оптимизацију уз помоћ *CNN Frontend-a*.

У прилогу А.7 се налази кôд који покреће извршавање оптимизоване неуронске мреже на вишепроцесорској плочи.

Глава 6

Оптимизација отклањања перспективе

На основу дате фотографије саобраћајног знака, коришћењем неуронске мреже *FourPointFinder*, коју сам описала у глави 5, можемо пронаћи пројективно пресликавање којим отклањамо перспективу.

6.1 Неоптимизован кôд

Кôд, приказан у наставку, ћемо оптимизовати уз помоћ вишепроцесорке плоче, описане у глави 2. Наредни кôдови за све оптимизације обрађују исту слику димензије 612×488, где је излаз димензије 300×300, ради поређења времена извршавања.

Да бисмо у потпуности одредили вредност сваког пиксела излазне слике, редом пролазимо кроз сваку координату илазне слике и на њој примењујемо иверзно пројективно прелсикавање, да бисмо сазнали која тачка улазне слике се пресликава у тражену тачку излазне слике. Применом инверзног пројективног пресликавања на целобројне координате изназне слике можемо добити реалне коориднате улазне слике. У тој ситуацији можемо да изаберемо различите методе апроксимације вредности пиксела излазне слике. Изабрала сам да разматрам билинеарну интерполацију и метод најближег суседа.

Опишимо билинеарну интерполацију преко функције $BilinearInterpolation$, при чему је *image* улазна слика, а (x, y) реалне координате улазне слике на основу којих вршимо интеполацију:

```
def BilinearInterpolation(image, x, y):
```

```
xf = floor(x)
yf = floor(y)


c00 = image(xf, yf)
c10 = image(xf + 1, yf)
c01 = image(xf, yf + 1)
c11 = image(xf + 1, yf + 1)

dx = x - xf
dy = y - yf

move_x_top = dx*c00 + (1-dx)*c10
move_x_bottom = dx*c01 + (1-dx)*c11

return dy*move_x_top + (1-dy)*move_x_bottom
```

Апроксимација најближим суседом узима вредност пиксела са координатама $(round(x), round(y))$, при чему су (x, y) одговарајуће реалне вредности координата на основу којих вршимо апроксимацију. Неоптимизован кôд за отклањање перспективе се налази у прилогу А.8.

У случају примене билинеране интеропације за одређивање вредности пиксела, време извршавање неоптимизованог кôда је $29ms$, а у случају одређивања вредности пиксела излазне слике уз помоћ најближег суседа време извршавања је $9ms$.

6.2 Оптимизација $1CVe$

Оптимизација се врши у више корака, при чему се кораци међусобно надограђују. Први корак је да се подаци за обраду запишу у спољној *DDR* меморији и да се користи само један процесор. Ову оптимизацију називамо $1CVe$. Уместо двоструке петље, која обрађује пиксел по пиксел слике, желимо да поделимо посао обрађивања пиксела нитима, тј. да за своје додељене пикселе врше билинеарну интерполацију или апроксимацију најближим суседом. Низ корака које нити извршавају је описан у засебном фајлу *cve_projection.c* за билинеарну интерполацију, а у *cve_projection_neighbor.c* је описан низ корака у случају апроксимације најближим суседом. Распоређивање података нитима и њихово покретање је кодирано у *r_test_cve.c*.

Подаци се преносе у *DDR* меморију преко *dataplane*-ова. У *dataplane*-у један се налазе елементи матрице пројективног пресликавања. У *dataplane*-у

нула се налазе пиксели улазне слике, а *dataplane* 2 треба да сместимо пикселе резултујуће слике. У глави 2 је наведено да пикселе матрице можемо добити преко координата. Слику посматрамо као низ rgb координата њених пиксела. Тиме је дужина слике утростручена, а висина остаје иста. Један члан тог низа има вредност између нуле и 255, што одговара вредности одговарајуће координате једног пиксела. У *r_ test_ cve.c* је преко *rect* команде описан начин поделе података за обраду по нитима. *Rect* команда описује начин кретања по слици, где током кретања формира сегменте слике, које прослеђује нитима као податке за обраду. *Rect* команда врши одређен померај по врсти слике, као и по колони, чиме формира сегмент за обраду. Пошто нам је био циљ да свака нит обради по један пиксел по сесији, *rect* команда се креће тако што се по реду помера за 3 док не дође до краја реда слике, а након обрађеног реда се спусти на следећи ред. Почиње обраду од позиције $(0, 0)$, и креће се на претходно описан начин, док не прође све врсте и колоне. За сваку тачку помераја *rect* команде се помера координатни систем тако да координатни почетак буде баш у тачки слике докле је стигла *rect* команда. То значи да је координатни систем релативан у односу на тачку убода *rect* команде. Функције *__getX()* и *__getY()* враћају апсолутне координате тренутне тачке убода *rect* команде у односу на стварни координатни почетак. На основу тих функција за сваку тачку слике можемо да сазнамо њене апсолутне координате транслирањем њених релативних кооридната за $(-\_\_getX(), -\_\_getY())$. Вредност пиксела са вредношћу релативних координата (x, y) у односу на тренутну тачку убода *rect* команде, добијамо прееко функције *__getSrc(x, y)*. Вредност пиксела на резултујућој слици са релативним координатама (x, y) у односу на тренутну тачку убода *rect* команде, постављамо на вредност p преко функције *__setDst(x, y, p)*. Кôд за оптимизацију $1CVe$ се налази у прилогу А.9.

Након *1CVe* оптимизације време извршавања програма за билинеарну интерполацију је $22ms$, а у случају најближег суседа $13ms$.

6.3 Оптимизација $2CVe$

У овој оптимизацији користимо опет само спољну *DDR* меморију, али оба процесора. Идеја је да слику поделимо на горњу и доњу половину, тако да један процесор обрађује горњу половину слике, а други доњу. Нити оба проце-

сора ће и даље радити исти низ корака, који су описани у фајлу *cve_ projection.c* за билинеарну интерполацију, а за најближег суседа у *cve_projection_ neighbor.c*. Разлика је што сад имамо два процесора, па ћемо имати и две *rect* команде. У *rect* командама треба да опишемо како ћемо поделити податке нитима оба процесора. *Rect* команда за оба процеора имају заједничко да се померају за по три елемената у реду, тј. за једна пиксел, узимајући његове *rgb* координате као податке, а кад заврше ред, спустиће се на следећи, и кренути од почетка новог реда. Међутим, први процесор креће од апсолутне координате $(0, 0)$ и стаје кад заврши *height_ out/2* редова. Док други процесор почиње од позиције $(0, height\_\ out/2)$ и треба да обради *height_ out - height_ out/2* редова. Кôд за оптимизацију $2CVe$ се налази у прилогу А.10.

Након *2CVe* оптимизације време извршавања програма за билинеарну интерполацију се спустило са $22ms$ на $13ms$, а у случају најближег суседа време извршавања спустило са $13ms$ на $9ms$.

6.4 Оптимизација *Scratchpad*

У овој оптимизацији користимо привремену и спољну меморију, као и оба процесора. Привремена меморија има брже време приступа читању и писању, али има мањи капацитет. Пошто привремена меморија има мањи капацитет од спољне меморије, идеја је да се обрађује хоризонтални исечци слике. Хоризонтални исечак је део слике који се простире целом њеном дужином, али да је сужен по висини. Поступак је следећи:

1. Пребаци хоризонтални исечак улазне слике из спољне меморије у привремену меморију.

2. Део слике који је у привременој меморији обради преко нити и резултат смести у излазни део привремене меморије.

3. Обрађен део слике, који се налази у излазном делу привремене меморије, пребаци у спољну меморију на одговарајућу позицију тако да у спољној меморији добијамо надовезане исечке који формирају целокупну излазну слику.

Да би нити могле да обраде хоризонталне исечке слике, морају да знају матрицу пројективног пресликавања. Тако да осим хоризонталнног исечка

у привремену меморију преносимо и матрицу пројективног пресликавања. На основу капацитета излазног дела привремене меморије, израчунала сам број итерација, тј. број хоризонталних исечака који морамо да обрадимо. $height\_out$ је висина излазне слике, $width\_out\_64\_rgb$ је ширина излазне слике помножена са 3, због rgb координата и проширена до броја који је дељив са 64(ово је услов који се намеће због организације привремене меморије). Први ред привремене меморије је резервисан за елементе матрице пројективног пресликавања, тако да од укупаног капацитета одузимамо величину једног реда. Тиме добијамо да је број итерација:

$$unsigned\ output\_size\_scp = 896 * 1024 * 4;$$
$$unsigned\ num\_of\_iter = height\_out * width\_out\_64\_rgb/(output\_size\_scp - width\_out\_64\_rgb) +$$
$$(height\_out * width\_out\_64\_rgb\%(output\_size\_scp - width\_out\_64\_rgb)! = 0);$$

На основу броја итерација, можемо прецизно да одредимо хоризонталне исечке који се простиру по редовима:

$$[i * (height\_out/num\_of\_iter), (i+1) * (height\_out/num\_of\_iter) - 1], i \in \{0, 1, .., num\_of\_iter - 2\},$$

а последњи исечак садржи редове:

$$[(num\_of\_iter - 1) * (height\_out/num\_of\_iter), height\_out).$$

Rect команде су слично написане као у оптимизацији $2CVe$. Једина разлика је у томе што уместо да делимо целу слику на два дела, сада сваки хоризонтални исечак излазе слике који обрађујемо делимо на горњу и доњу половину, где ће $1CVe$ да обрађује горњу половину, а $2CVe$ доњу.

Треба да опишемо који тачно део улазне слике из спољне меморије желимо да пребацимо у привремену, као и где ћемо пребацити резултујући хоризонатлни исечак излазне слике у спољну меморију. Ово се постиже преко низа наредбни које прослеђујемо преносној јединици $iDMA$. Да бих одредила део улазне слике, који се слика у хоризонтални исечак излазне слике, користила сам функцију $find\_boundaries$. У тој функцији одређујем тражени четвороугао $ABCD$, који се слика у хоризонтални исечак са познатим пројективним преслиаквањем. Одлучила сам да на основу тог четвороугла пронађем минимални хоризонтални исечак улазне слике, који садржи четвороугао $ABCD$

и да тај део слике пребацим у привремену меморију. Кôд за оптимизацију *Scratchpad* се налази у прилогу А.11.

Нити изваршавају кôд написан у фајлу *cve_ projection_ idma.c* за билинеарну интерполацију, а за најближег суседа у *cve_projection_ neighbor_ idma.c*. Разлика у односу на претходне оптимизације је да су координате за приступ подацима релативне у односу на привремену меморију. То значи да осим што се координатни почетак помера на основу *rect* команде, да ће се координати почетак померати и по горњим левим угловима хоризонталних исечака излазне слике, тј. по $(0, i * (height\_out/num\_of\_iter))$. Да бисмо израчунали апсолутне координате, преко униформне меморије смо слали тренутну вредност бројача i, тј. редни број хоризонталног исечка, ког тренутно обрађујемо. Кôд које извршавају нити за оптимизацију *Scratchpad* се налази у прилогу А.12.

Након *Scratchpad* оптимизације време извршавања програма за билинеарну интерполацију је $6ms$, а у случају најближег суседа $2ms$.

6.5 Оптимизација $TGDMAC + LWM$

У овој оптимизацији користимо спољну, привремену и локалну радну меморију нити, као и оба процесора. Локална радна меморија има најмањи капацитет у односу на спољну и привремену меморију, са само $16kB$. На основу базе података $GTSRB$, у којој највећа слика има димензије 250×250, у најгороем случају највише можемо да обрадимо по један ред излазне слике по сесији. У свим претходним оптимизацијама, нити су радиле независно, тј. све су биле у режиму господара. У овој оптимизацији, имаћемо једну нит која ће бити господар, а осталих седам нити ће бити слуге. Поступак је следећи:

1. Пребаци хоризонтални исечак улазне слике из спољне меморије у привремену меморију.

2. На основу траженог реда излазне слике који желимо да израчунамо, пронађи податке у хоризонталном делу улазне слике који су ти потребни и пребаци их из привремене меморије у локалну радну меморију.

3. У локалној радној меморији одреди вредности пиксела реда излазне слике.

4. Израчунат ред илазне слике, пребаци у привремену меморију на одговарајућу позицију.

5. Након обрађених свих редова хоризонталног исечка излазног дела слике, пребаци резултујући исечак из излазног дела привремене меморије у спољну меморију на одговарајућу позицију, тако да у спољној меморији добијамо надовезане исечке који формирају целокупну излазну слику.

У *r_test_cve.c* постоји неколико малих разлика у односу на претходну оптимизацију. Назначен је другачији режим рада нити и *rect* команда се више не креће унутар врсте слике. Пошто ће пиксели бити израчунати унутар локалне радне меморији, где ће цео ред слике бити пребачен, кретаћемо се само по верикали. Кôд за оптимизацију $TGDMAC + LWM$ се налази у прилогу А.13.

У фајлу *cve_ projection_lwm.c* се налази кôд који изврашавају нити за билинеарну интерполацију, а кôд за најближег суседа се налази у *cve_projection_neighbor_lwm.c*. Ту су описане функције преноса података из привремене у локалну ранду меморију. Оне су написане у асемблеру и траже горњу леву тачку сегмента слике који желимо да пребацимо, као и његову ширину и висину. Те податке користи преносна јединица $iDMA$, која ће извршити трансфер података. Нит, која има улогу господара, ради следеће:

1. Узима вредности пројективног пресликавања из привремене меморије, које чува у помоћном низу;

2. За тренутни ред излазне слике, чије пикселе желимо да израчунамо, одређује потребне пикселе улазне слике;

3. Тражи пренос потребних података из привремене у локалну радну меморију;

4. Позива нити које имају улогу слуге да израчунају пикселе. Такође, да не би нит улоге господара непотребно чекала да се овај поступак заврши, укључује и себе у одбраду пиксела;

5. Након завршеног поступка израчунавања пиксела реда излазне слике, нит улоге господара позива пренос података из локалне у привремену меморију.

Кôд који извршавају нити за оптимизацију $TGDMAC + LWM$ се налази у прилогу А.14.

Након *TGDMAC+LWM* оптимизације време извршавања програма за билинеарну интерполацију је $1ms$, а у случају најближег суседа $1ms$.

6.6 Компајлерске оптимизације

Последња оптимизација користи одвијање петље ради бољих перформанси, по цену мало дужег кôда и потенцијано већег броја коришћених регистара. Свака петља која има константан број корака се мења одговрајућим низом корака тако да се добије линеаран кôд, без скокова. Предност овога је да унутрашње наредбе петље могу да бити паралелизоване, ако су независне од итерације до итерације петље и смањује се број грањања програма, јер не постављамо за сваку итерацију упит да ли треба да изађемо из петље. Кôд за ову оптимизацију је практино исти, осим што је додат један $flag$ у $make$ фајлу.

Након ове оптимизације време извршавања програма за билинеарну интерполацију је $1ms$, а у случају најближег суседа $1ms$. Разлика у односу на претходни корак се мери у нано секундама.

На табели (6.1) се налазе времена извршавања свих оптимизација, као и време извршавања неоптимизованог кôда.

Врста оптимизације	билинеарна интерполација	најближи сусед
ARM	29	9
1CVe	22	13
2CVe	13	9
iDMA+Scratchapad	6	2
TGDMC+LWM	1	1
компајлерске оптимизације	1	1

Табела 6.1: Времена извршавања

На основу другог дела мастер рада, део слике на којој се налази саобраћајни знак се налази у *test.ppm*, а координате тачака пројективног пресликавања су записане у фајлу *koordinate.txt*. У прилогу А.15 се налази кôд за покретање трећег дела рада.

Глава 7

Закључак

Решење отклањања перспективе са фотографија саобраћајних знакова, смо поделили на три дела. У првом делу смо преко методе клизајућих прозора и класификатора саобраћајног знака, изоловали део слике на којој се налази тражени објекат. У другом делу смо на основу резултата првог дела, пронашли четири тачке за пројективно пресликавање помоћу неуронске мреже. У трећем делу смо решавајући систем једначина, пронашли пројективно пресликавање и пресликали дату слику, тако да је перспектива отклоњена. Коришћем вишепроцесорке плоче смо убрзали време извршавања програма.

Мане програма се јављају у првом и другом делу рада. Пошто су коришћене неуронске мреже, решење неће бити егзактно, већ са одређеном вероватноћом тврдимо да је решење тачно. Резултати издвајања саобраћајног знака и детектовања координата тачки пројективног пресликавања се могу побољшати прецизнијим класификатором и прецизнијом неуронском мрежом за одређивање четири тачке пројективног пресликавања.

Допринос рада је у начину оптимизације кôда, у којој су постигнута знатно бржа времена извршавања. У случају да се у будућности развију прецизнији статистички модели за први и други део рада, и даље би могао да се примени исти начин оптимизације и тиме постогну побољшања перформанси.

Библиографија

1. *Машинско учење*, Младен Николић, Ађелка Зечевић, Београд 2019.
2. *Fundamentals of Digital Image Processing*, Anil K. Jain
3. Предавања професора Срђана Вукмировића из Геометрије 4

Додатак А

Кôд

А.1 Припрема података за класификацију

```
from csv import writer
import pandas as pd
import cv2 as cv
import matplotlib.pyplot as plt
import random
import numpy as np

path = 'C:\\Users\\bozica\\Desktop\\deljeni_folder\\GTSRB\\Final_Training\\Images\\'
csvFiles = []
folders = []
for i in range(0,43):
    if i<10:
        folders.append('0000' + str(i))
    else:
        folders.append('000' + str(i))

for folder in folders:
    subpath = path+folder+'\\GT-'+folder+'.csv'
    csvFiles.append(subpath)

for file in csvFiles:
    data = pd.read_csv(file)
    writePath = 'C:\\Users\\bozica\\Desktop\\SlikeZaKlasifikaciju3\\'+file[-9:-4]
    imagePathPrefix = path+file[-9:-4]+'\\'
    rowsToWrite = []
    print(file)
    num_of_images = 0

    for row in data.iterrows():
        if num_of_images == 90:
            break

        values =
```

```
row[1]['Filename;Width;Height;Roi.X1;Roi.Y1;Roi.X2;Roi.Y2;ClassId'].split(';')
imagePath = imagePathPrefix+values[0]

x1 = int(values[3])
y1 = int(values[4])
x2 = int(values[5])
y2 = int(values[6])

image = cv.imread(imagePath)
h,w = image.shape[0:2]

pyramid = [[image,[h,w], [x1,y1,x2,y2]]]
scale = 0.5

h_new = round(h*scale)
w_new = round(w*scale)
x1_new = round(x1*scale)
y1_new = round(y1*scale)
x2_new = round(x2*scale)
y2_new = round(y2*scale)

while h_new>=32 and w_new>=32:
    new_image = cv.resize(image, (h_new, w_new))
    pyramid.append([new_image, [h_new, w_new], [x1_new, y1_new, x2_new, y2_new]])
    h_new = round(h_new*scale)
    w_new = round(w_new*scale)
    x1_new = round(x1_new*scale)
    y1_new = round(y1_new*scale)
    x2_new = round(x2_new*scale)
    y2_new = round(y2_new*scale)

if not(w_new==32 and h_new==32):
    new_image = cv.resize(image, (32, 32))
    x1_new =round( x1/w*32)
    y1_new = round(y1/h*32)
    x2_new = round(x2/w*32)
    y2_new = round(y2/h*32)
    pyramid.append([new_image, [32, 32], [x1_new, y1_new, x2_new, y2_new]])

for k in range(0, len(pyramid)):
    elem = pyramid[k]
    img = elem[0]
    h_new = elem[1][0]
    w_new = elem[1][1]
    x1_new = elem[2][0]
    y1_new = elem[2][1]
    x2_new = elem[2][2]
    y2_new = elem[2][3]

    for j in range(0, h_new-32,4):
        for i in range(0, w_new-32,4):
            new_image = img[i:(i+32), j:(j+32)]
            value = "0"
```

```
                    if i<=x1_new and x2_new<i+32 and j<=y1_new and y2_new<j+32:
                        value="1"
                    try:
                        cv.imwrite(writePath+values[0][0:-4]+"_"+str(k)+"_"+str(i)+"_"
                        +str(j) +'.ppm', new_image)
                        rowsToWrite.append([file[-9:-4]+values[0][0:-4]+"_"+str(k)+"_"+
                        str(i)+"_"+str(j) +'.ppm'+';'+value])
                    except:
                        continue

        value="1"
        for i in range(0, x1, 4):
            for j in range(x2+1, w, 4):
                for z in range(0, y1, 4):
                    for l in range(y2+1, h, 4):
                        new_image = img[i : j, z:l]
                        try:
                            cv.imwrite(writePath+values[0][0:-4]+"_"+ str(i)+'_'+str(j) +
                            "_"+ str(z)+'_'+str(l)+'.ppm', new_image)
                            rowsToWrite.append([file[-9:-4]+values[0][0:-4]+"_"+str(i)
                            +"_"+str(j) + "_"+ str(z)+'_'+str(l)+
                            '.ppm'+';'+value])
                        except:
                            continue
        num_of_images+=1

    with open('C:\\Users\\bozica\\Desktop\\SlikeZaKlasifikaciju3\\data.csv', 'a')
        as f_object:
        # Pass this file object to csv.writer()
        # and get a writer object
        writer_object = writer(f_object)

        # Pass the list as an argument into
        # the writerow()
        for elem in rowsToWrite:
            writer_object.writerow(elem)
```

A.2 Модел класификације

```
import numpy as np
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import utils
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, classification_report
```

```
import cv2 as cv
import csv
import os
import copy
import pandas as pd
import random
np.random.seed(7)

data = {'Pixels':[], 'isTrafficSign':[]}

#lets copy the image pixels from the folders and their target values
path = 'C:\\Users\\bozica\\Desktop\\SlikeZaKlasifikaciju3\\'
filename = path + 'data.csv'

path, dirs, files = next(os.walk(path))
n = len(files) - 1
even = 0

isTrafficSign = np.zeros(shape=(n,1),dtype=np.uint8)
pixels = np.zeros(shape=(n,32,32,3),dtype=np.uint8)

i=0
with open(filename, 'r') as csvfile:
    datareader = csv.reader(csvfile)
    for row in datareader:
        if even%2==1:
            name = row[0].split(';')[0]
            file = path + name
            value = int(row[0].split(';')[1])
            isTrafficSign[i] = value
            image = cv.imread(file)
            image = cv.resize(image,(32,32))
            pixels[i] = image
            i+=1
        even+=1

isTrafficSign = np.array(isTrafficSign)
pixels = np.array(pixels)
print(isTrafficSign.shape)
print(pixels.shape)

jedn=0
nula=0
for elem in isTrafficSign:
    if elem==1:
        jedn+=1
    else:
        nula+=1
print(jedn,nula)

data['Pixels'] = pixels
data['isTrafficSign'] = isTrafficSign
```

```
X = data['Pixels']
print(X.shape)
X = X.astype('float32')
X /= 255
Y = isTrafficSign
number_of_classes = 2
Y = utils.to_categorical(Y, number_of_classes)

X_train, X_test, Y_train, Y_test =
model_selection.train_test_split(X, Y, test_size = 0.33, random_state = 7)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

#creatining the neural network model
classes = 2
input_shape = (32,32,3)
model = Sequential([
    Input(shape=input_shape),
    Conv2D(filters=8, kernel_size=(5, 5), strides=(1, 1), padding='same',
    activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=16, kernel_size=(5, 5), strides=(1, 1), padding='same',
    activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=32, kernel_size=(5, 5), strides=(1, 1), padding='same',
    activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(classes, activation='softmax')
])

model.compile(loss=CategoricalCrossentropy(), optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

batch_size = 128
epochs = 12

history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)

plt.figure(figsize=(10, 4))
```

```
plt.subplot(1, 2, 1)
plt.title('Loss')
plt.plot(np.arange(0, epochs), history.history['loss'], label='train')
plt.plot(np.arange(0, epochs), history.history['val_loss'], label='val')
plt.legend(loc='best')

plt.subplot(1, 2, 2)
plt.title('Accuracy')
plt.plot(np.arange(0, epochs), history.history['accuracy'], label='train')
plt.plot(np.arange(0, epochs), history.history['val_accuracy'], label='val')
plt.legend(loc='best')

plt.show()

score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

#Test loss: 0.01462860219180584
#Test accuracy: 0.9954534769058228

y_predicted = model.predict(X_test)
y_predicted_classes = np.argmax(y_predicted, axis = 1)
y_test_classes = np.argmax(Y_test, axis = 1)

matrix_of_confusion = confusion_matrix(y_test_classes, y_predicted_classes)
print(matrix_of_confusion)

#[[104539     346]
# [   211   17415]]

report = classification_report(y_test_classes, y_predicted_classes)
print(report)

fig = plt.figure(figsize=(20, 16))
columns = 4
rows = 5
for i in range(1, rows*columns +1):
    fig.add_subplot(rows, columns, i)
    img = X_test[i]

    true_label = np.argmax(Y_test[i])
    predicted_label = np.argmax(y_predicted[i])
    plt.title('Ocekivana klasa {0} \nPridruzena klasa: {1}'
    .format(true_label, predicted_label))
    plt.axis('off')

    plt.imshow(img, cmap='gray')

plt.show()

model.save('C:\\Users\\bozica\\Desktop\\deljeni_folder\\MasterRad\\'+
'TrafficSignModelRecognition3')
```

А.3 Пример детекције саобраћајног знака помоћу оптимизованог класификатора

```
import subprocess
import numpy as np
import cv2 as cv
import os
import copy
from PIL import Image
import sys

def transform(x):
    return x*(2**7)

def preprocessImage(image):

    image = image.astype('float32')
    image/=255
    image = np.reshape(image, (1, 32,32, 3))
    image = transform(image)
    image = np.array(image, dtype=np.uint8)
    image = np.reshape(image, (32,32,3))
    image = Image.fromarray(image)
    return image

print('Unesite putanju do fajla: ', end=None)
path = input()

image = Image.open(path)
image_array = np.array(image)

h_original,w_original = image_array.shape[0:2]
pyramid = []
scale = 0.5
h = h_original
w = w_original

if h>32 and w>32:
    pyramid.append(image_array)
    while True:
        h_old = h
        w_old = w
        h = int(h*scale)
        w = int(w*scale)
        if h<32 or w<32:
            break

        blur = cv.GaussianBlur(image_array,(5,5),0)
        image_array = cv.resize(blur, (h,w))
        pyramid.append(image_array)

    if h_old!=32 and w_old!=32:
```

```
        blur = cv.GaussianBlur(image_array,(5,5),0)
        image_array = cv.resize(blur, (32,32))
        pyramid.append(image_array)
elif h==32 and w==32:
    pyramid = [image_array]
else:
    image_array = cv.resize(image_array, (32,32))
    pyramid = [image_array]

windowSize = (32,32)
step = 4
max_prob = 0
best_image = None
best_i = 0
best_j = 0

hasTrafficSign = 0
brojac = 0
for imag in pyramid:
    h,w = imag.shape[0:2]
    print('Dimenzije slike:', h,w)
    for j in range(0, h-windowSize[0]+1, step):
        for i in range(0, w-windowSize[1]+1, step):
            imageCopy = copy.deepcopy(imag[j:j+32, i:i+32])
            imageCopy = preprocessImage(imageCopy)

            imageCopy.save('C:\\Users\\bozica\\Desktop\\input_3_memory.ppm')

            res = subprocess.run(['scp', '/c/Users/bozica/Desktop/input_3_memory.ppm',
            'root@10.10.96.183:/home/root/bozica'])

            res = subprocess.run(['ssh', 'root@10.10.96.183', 'cd', 'bozica' + ';',
            './final_app_template_HIL_GNU_linux', '--bcl', 'subn0_commandlist.bcl',
            '--commandlist','subn0_cl_cnn0.bin',
            '--core', 'CNN', '--sync-id', '1', '--commandlist','subn0_cl_cve0.bin',
            '--core', 'CVE', '--sync-id', '4', '--commandlist', 'subn0_cl_dma0.bin',
            '--core', 'DMA', '--sync-id', '2', '--setMemory',
            'input_3_memory:0,input_3_memory.ppm,0', '--setMemory',
            'input_3_memory:1,input_3_memory.ppm,1', '--setMemory',
            'input_3_memory:2,input_3_memory.ppm,2'])

            res = subprocess.run(['scp', 'root@10.10.96.183:/home/root/bozica/'+
            'statefulpartitionedcall_sequential_2'+
            '_dense_8_softmax_memory.csv',
            '/c/Users/bozica/Desktop/izlaz_ploce/' +
            str(brojac)+ '_'+str(i)+ '_'+str(j)+'.csv'])


    brojac+=1

directory = 'C:\\Users\\bozica\\Desktop\\izlaz_ploce\\'
for filename in os.listdir(directory):
    f = open(directory + filename, 'r')
```

```
    linije = f.read().split('\n')
    f.close()
    a = int(linije[1][:-2], 16)
    b = int(linije[2][:-2], 16)
    if b>a and b>max_prob:
        max_prob = b
        best_image = filename

if max_prob == 0:
    print('na slici nema trazenog objekta')
else:
    reci = filename.split('_')
    best_image_index = int(reci[0])
    best_i = int(reci[1])
    best_j = int(reci[2][0:-4])
    best_image = pyramid[best_image_index]
    result = best_image[best_j:best_j+32, best_i:best_i+32]
    image = Image.fromarray(result)
    image.save('C:\\Users\\bozica\\Desktop\\test.ppm')
    result = preprocessImage(result)
    result.save('C:\\Users\\bozica\\Desktop\\input_2_memory.ppm')
```

A.4 Пример детекције саобраћајног знака помоћу неоптимизованог класификатора

```
#biblioteke
import cv2 as cv
import matplotlib.pyplot as plt
import copy
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
import time
import threading
from concurrent.futures import ThreadPoolExecutor
from PIL import Image
import subprocess

model = keras.models.load_model('C:\\Users\\bozica\\Desktop\\deljeni_folder\\'+
'MasterRad\\TrafficSignModelRecognition3')

lock = threading.Lock()
value = 0
best_name = None

def transform(x):
    return x*(2**7)

def preprocessImage(image):
```

```
    image = image.astype('float32')
    image/=255
    image = np.reshape(image, (1, 32,32, 3))
    image = transform(image)
    image = np.array(image, dtype=np.uint8)
    image = np.reshape(image, (32,32,3))
    image = Image.fromarray(image)
    return image

class myThread (threading.Thread):
    def __init__(self, i, j, model, image, image_id):
        threading.Thread.__init__(self)
        self.name = str(image_id)+' '+str(i)+ ' '+ str(j)
        self.i = i
        self.j = j
        self.model = model
        self.isecak = copy.deepcopy(image[j:j+32, i:i+32])
    def run(self):
        global value
        global best_name
        img = self.isecak.astype('float32')
        img /=255
        img = np.reshape(img,[1,32,32,3])
        prediction = self.model.predict(img)
        klasa = np.argmax(prediction[0])
        if klasa==1:
            lock.acquire()
            if prediction[0][1] > value:
                value = prediction[0][1]
                best_name = self.name
            lock.release()

path = input('unesite putanju do slike: ')
image = Image.open(path)
image_array = np.array(image)

h_original,w_original = image_array.shape[0:2]
pyramid = []
scale = 0.5
h = h_original
w = w_original

if h>32 and w>32:
    pyramid.append(image_array)
    while True:
        h_old = h
        w_old = w
        h = int(h*scale)
        w = int(w*scale)
        if h<32 or w<32:
            break

        blur = cv.GaussianBlur(image_array,(5,5),0)
```

```
        image_array = cv.resize(blur, (h,w))
        pyramid.append(image_array)

    if h_old!=32 and w_old!=32:
        blur = cv.GaussianBlur(image_array,(5,5),0)
        image_array = cv.resize(blur, (32,32))
        pyramid.append(image_array)
elif h==32 and w==32:
    pyramid = [image_array]
else:
    image_array = cv.resize(image_array, (32,32))
    pyramid = [image_array]

threads = []
for k in range(0, len(pyramid)):
    h,w = pyramid[k].shape[0:2]
    for i in range(0, w-32+1, 4):
        for j in range(0, h-32+1, 4):
            thread = myThread(i, j, model, pyramid[k], k)
            thread.start()
            threads.append(thread)
for thread in threads:
    thread.join()

if best_name is None:
    print('Na datoj slici nema saobracajnog znaka')
else:
    names = best_name.split(' ')
    k = int(names[0])
    i = int(names[1])
    j = int(names[2])
    best_image = pyramid[k][j:j+32, i:i+32]
    best_image = preprocessImage(best_image)
    best_image.save('C:\\Users\\bozica\\Desktop\\input_2_memory.ppm')
```

A.5 Генерисање слика за обучавање неуронске мреже за детекцију координата четири улазне тачке пројективног пресликавања

```
import numpy as np
import cv2 as cv
from csv import writer
import pandas as pd
import random


#given the 4 input and output points find projection P such that:
```

```
#inputput->(0,0), (0,1), (1,1), (1,0)
def find_projection_to_base(input_points):
    x0 = float(input_points[0][0])
    y0 = float(input_points[0][1])
    x1 = float(input_points[1][0])
    y1 = float(input_points[1][1])
    x2 = float(input_points[2][0])
    y2 = float(input_points[2][1])
    x3 = float(input_points[3][0])
    y3 = float(input_points[3][1])

    d1 = np.linalg.det(np.vstack(([x0, y0, 1], [x1, y1, 1], [x2, y2, 1])))
    d2 =  np.linalg.det(np.vstack(([x3, y3, 1], [x1, y1, 1], [x2, y2, 1])))
    d3 = np.linalg.det(np.vstack(([x0, y0, 1], [x1, y1, 1], [x3, y3, 1])))
    d4 = np.linalg.det(np.vstack(([x0, y0, 1], [x2, y2, 1], [x3, y3, 1])))

    if d1==0 or d2==0 or d3==0 or d4==0:
        return None

    m = 1
    n = 1

    t0 = 1
    t1 = 1
    t2 = 1
    t3 = 1
    a = 1
    b = 1
    c = 1
    d = 1
    e = 1
    f = 1
    g = 1
    h = 1
    i = 1
    if x1-x0==0:
        t1 = t0*((x2 - x0)*(y1 - y3) + (y2 - y1)*(x3 - x0)) /
                ((y2 - y0)*(x3 - x0) - (y3 - y0)*(x2 - x0))
        t2 = t1 * ((y2 - y0) - (y3 - y0) * (x2 - x0) / (x3 - x0)) / (y1 - y0)
        t3 = t2 * (x3 - x0) / (x2 - x0)

        b = 0
        e = m*t1 / (y1 - y0)
        h = (t1 - t0) / (y1 - y0)

        a = n*t2 / (x2 - x0)
        d = m*t2 / (x2 - x0) - m*t1*(y2 - y0) / ((y1 - y0)*(x2 - x0))
        g = (t2 - t0) / (x2 - x0) - (y2 - y0)*(t1 - t0) / ((y1 - y0)*(x2 - x0))

        c = - a*x0 - b*y0
        f = - d*x0 - e*y0
        i = t0 - g*x0 - h*y0
    else:
```

```
        t2 = t1*(- (x3 - x0)*(y2 - y0)*(x1 - x0) + (x3 - x0)*(y1 - y0)*(x2 - x0) +
            (x2 - x0)*(y3 - y0)*(x1 - x0) - (x2 - x0)*(y1 - y0)*(x3 - x0)) /
            ((x1 - x0)*((y3 - y0)*(x1 - x0) - (y1 - y0)*(x3 - x0)))
        t3 = t2 * ((y3 - y0) * (x1 - x0) - (x3 - x0) * (y1 - y0)) /
            (-(x2 - x0) * (y1 - y0) + (y2 - y0) * (x1 - x0))


        t0 = (-t2*(x1 - x0)*((y3 - y0)*(x1 - x0) - (y1 - y0)*(x3 - x0))
                + t3*(x1 - x0)*((y2 - y0)*(x1 - x0) - (y1 - y0)*(x2 - x0))
                + t1*((y3 - y0)*(x1 - x0)*(x2 - x0) - (y1 - y0)*(x3 - x0)*(x2 - x0) -
                (y2 - y0)*(x1 - x0)*(x3 - x0) + (y1 - y0)*(x2 - x0)*(x3 - x0))) /
                ((y3 - y0)*(x1 - x0)*(x2 - x1) - (y1 - y0)*(x3 - x0)*(x2 - x1) -
                (y2 - y0)*(x1 - x0)*(x3 - x1) + (y1 - y0)*(x2 - x0)*(x3 - x1))

        b = n*t2*(x1 - x0)/(-(x2 - x0)*(y1 - y0)+(y2 - y0)*(x1 - x0))
        e = (m*t2*(x1 - x0) - (x2 - x0)*m*t1) /
            ((y2 - y0)*(x1 - x0) - (y1 - y0)*(x2 - x0))
        h = ((t2 - t0)*(x1 - x0) - (x2 - x0)*(t1 - t0)) /
            ((y2 - y0)*(x1 - x0) - (y1 - y0)*(x2 - x0))

        a = - b * (y1 - y0) / (x1 - x0)
        d = (m * t1 - e * (y1 - y0)) / (x1 - x0)
        g = (t1 - t0 - h * (y1 - y0)) / (x1 - x0)

        c = - a * x0 - b * y0
        f = - d * x0 - e * y0
        i = t0 - g * x0 - h * y0

    return [[a,b,c], [d,e,f], [g,h,i]]

def find_projection(input_points, output_points):
    projection1 = find_projection_to_base(input_points)
    projection2 = find_projection_to_base(output_points)

    if projection1 == None or projection2 == None:
        return None

    p1 = np.array(projection1)
    p2 = np.array(projection2)
    p_2 = np.linalg.inv(p2)
    p = p_2.dot(p1)

    return p

#chages x to be between [0,w)
def relu(x, w):

    if x<0:
        return 0
    elif x>=w:
        return w-1
    else:
        return int(x)
```

```
def applyProjection(projection, point):
    x = projection[0][0] * point[0] + projection[0][1] * point[1] + projection[0][2]
    y = projection[1][0] * point[0] + projection[1][1] * point[1] + projection[1][2]
    t = projection[2][0] * point[0] + projection[2][1] * point[1] + projection[2][2]

    if t==0:
        return None

    return [x/t, y/t]

def checkBound(x, a, b):
    if x<=b and x>=a:
        return True
    else:
        return False

folder_names = ['00000', '00011', '00012', '00013', '00014']
paths = []
for name in folder_names:
    path = "C:\\Users\\bozica\\Desktop\\deljeni_folder\\GTSRBcopy\\"+
           "\\Final_Training\\Images\\" + name + "\\"
    for i in range(0, 7):
        new_path = path + '0000' + str(i)
        for j in range(0, 30):
            image_path = new_path + '_000'
            if j<10:
                image_path += '0' + str(j)
            else:
                image_path += str(j)

            image_path += '.ppm'

            paths.append(image_path)

j = 0
i = 0
#mapa slika sadrzi kao kljuc putanju do fajla,
#a kao vrednost listu koordinata tacaka
map_path_to_coordinates = {}
with open('C:\\Users\\bozica\\Desktop\\MojeSlike\\data.csv', 'w') as f_object:
    writer_object = writer(f_object)

    k=0
    columns = 'Filename;Width;Height;Roi.X1;Roi.Y1;Roi.X2;Roi.Y2;'+
              'ClassId;x1;y1;x2;y2;x3;y3;x4;y4'
    for name in folder_names:
        path = "C:\\Users\\bozica\\Desktop\\deljeni_folder\\GTSRBcopy\\"+
                   "Final_Training\\Images\\" + name + "\\" + "GT-"+ name + ".csv"
        data0 = pd.read_csv(path)
        j+=1

        for row in data0.iterrows():
```

```
            vrednost = ""
            vrednost+=row[1][columns].split(';')[8]
            vrednost+=';'
            vrednost+= row[1][columns].split(';')[9]
            vrednost += ';'
            vrednost+= row[1][columns].split(';')[10]
            vrednost += ';'
            vrednost+= row[1][columns].split(';')[11]
            vrednost += ';'
            vrednost+= row[1][columns].split(';')[12]
            vrednost += ';'
            vrednost+= row[1][columns].split(';')[13]
            vrednost += ';'
            vrednost+= row[1][columns].split(';')[14]
            vrednost += ';'
            vrednost+= row[1][columns].split(';')[15]
            lista = vrednost.split(';')
            new_lista = []
            for elem in lista:
                new_lista.append(int(elem))
            map_path_to_coordinates[paths[k]] = new_lista
            writer_object.writerow([name +"_"+ row[1][columns].split(';')[0] +
                        ';'+vrednost])
            image = cv.imread(paths[i])
            cv.imwrite('C:\\Users\\bozica\\Desktop\\MojeSlike\\'+name+'_' +
                row[1][columns].split(';')[0], image)
            i+=1
            k+=1
            if i==j*210:
                break

map_of_projections = {}
#mapa projekcija kao kljuc ima putanju do vec obradjene slike
#a kao vrednost listu projekcija koje mozemo da primenimo na tu sliku
for path in paths:
    image = cv.imread(path)
    h,w = image.shape[0:2]
    coordinates = map_path_to_coordinates[path]

    input_points = []
    for i in range(0,4):
        input_points.append([coordinates[2*i], coordinates[2*i+1]])

    projections = []

    #samo skaliranje nema smisla jer ce sve slike biti skaliranje na 32x32,
    #ne znace duplikati

    #prosirenje po x koordinatama gornje ivice
    for i in range(0, 20):
        output_points = []
        r = random.random()/2
        for j in range(0, 4):
```

```
            if j==0:
                output_points.append([relu((1 - r)*input_points[j][0], w),
                                                    input_points[j][1]])
            elif j==1:
                output_points.append([relu(r*w + (1-r)*input_points[j][0], w),
                                                        input_points[j][1]])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])
        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)

    #prosirenje po x osi donje ivice
    for i in range(0, 20):
        output_points = []
        r = random.random()/2
        for j in range(0, 4):
            if j==2:
                output_points.append([relu((1-r)*input_points[j][0], w),
                                                    input_points[j][1]])
            elif j==3:
                output_points.append([relu(r*w + (1-r)*input_points[j][0], w),
                                                    input_points[j][1]])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])
        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)
    #prosirenje po x osi leve ivice
    for i in range(0, 20):
        output_points = []
        r = random.random() / 2
        for j in range(0, 4):
            if j == 0:
                output_points.append([relu((1-r)*input_points[j][0], w),
                                                    input_points[j][1]])
            elif j == 2:
                output_points.append([relu(r*w + (1-r)*input_points[j][0], w),
                                                    input_points[j][1]])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])

        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)

    #prosirenje po x osi desne ivice
```

```
for i in range(0, 20):
    output_points = []
    r = random.random() / 2
    for j in range(0, 4):
        if j == 1:
            output_points.append([relu((1-r)*input_points[j][0], w),
                                                    input_points[j][1]])
        elif j == 3:
            output_points.append([relu(r*w + (1-r)*input_points[j][0], w),
                                                      input_points[j][1]])
        else:
            output_points.append([input_points[j][0], input_points[j][1]])

    try:
        projection = find_projection(input_points, output_points)
        projections.append([projection, output_points])
    except:
        projections.append(None)

#prosirenje po y osi gornje ivice
for i in range(0, 20):
    output_points = []
    r = random.random()/2
    for j in range(0, 4):
        if j==0:
            output_points.append([input_points[j][0],
                                relu((1-r)*input_points[j][1], h)])
        elif j==1:
            output_points.append([input_points[j][0],
                        relu(r*h + (1-r)*input_points[j][1], h) ])
        else:
            output_points.append([input_points[j][0], input_points[j][1]])

    try:
        projection = find_projection(input_points, output_points)
        projections.append([projection, output_points])
    except:
        projections.append(None)

#prosirenje po y osi donje ivice
for i in range(0, 20):
    output_points = []
    r = random.random()/2
    for j in range(0, 4):
        if j==2:
            output_points.append([input_points[j][0],
                                relu((1-r)*input_points[j][1], h)])
        elif j==3:
            output_points.append([input_points[j][0],
                                relu(r*h + (1-r)*input_points[j][1], h)])
        else:
            output_points.append([input_points[j][0], input_points[j][1]])
```

```
        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)

    #prosirenje po y osi leve ivice
    for i in range(0, 20):
        output_points = []
        r = random.random() / 2
        for j in range(0, 4):
            if j == 0:
                output_points.append([input_points[j][0],
                                      relu((1-r)*input_points[j][1], h)])
            elif j == 2:
                output_points.append([input_points[j][0],
                                      relu(r*h + (1-r)*input_points[j][1], h)])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])

        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)

    #prosirenje po y osi desne ivice
    for i in range(0, 20):
        output_points = []
        r = random.random() / 2
        for j in range(0, 4):
            if j == 1:
                output_points.append([input_points[j][0],
                                      relu((1-r)*input_points[j][1], h)])
            elif j == 3:
                output_points.append([input_points[j][0],
                                      relu(r*h + (1-r)*input_points[j][1], h)])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])

        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)
    #izduzenje po glavnoj diagonali
    for i in range(0, 20):
        output_points = []
        r = random.random() / 2
        for j in range(0, 4):
            if j == 0:
                output_points.append([relu((1-r)*input_points[j][0], w),
                                      relu((1-r)*input_points[j][1], h)])
```

```
            elif j == 3:
                output_points.append([relu(r*w + (1-r)*input_points[j][0],w),
                                      relu(r*h + (1-r)*input_points[j][1], h)])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])

        try:
            projection = find_projection(input_points, output_points)
            projections.append([projection, output_points])
        except:
            projections.append(None)

    # izduzenje po sporednoj diagonali
    for i in range(0, 20):
        output_points = []
        r = random.random() / 2
        for j in range(0, 4):
            if j == 1:
                output_points.append([relu((1-r)*input_points[j][0], w),
                                      relu(r*h + (1-r)*input_points[j][1], h)])
            elif j == 2:
                output_points.append(
                    [relu(r*w + (1-r)*input_points[j][0], w),
                                      relu((1-r)*input_points[j][1], h)])
            else:
                output_points.append([input_points[j][0], input_points[j][1]])

        try:
            projection = find_projection(input_points, output_points)
            if projection is None:
                projections.append(None)
            else:
                projections.append([projection, output_points])
        except:
            projections.append(None)
    map_of_projections[path] = projections

for path in paths:
    projections = map_of_projections[path]
    image = cv.imread(path)
    h, w = image.shape[0:2]
    coordinates = map_path_to_coordinates[path]
    input_points = []
    for i in range(0, 4):
        input_points.append([coordinates[2 * i], coordinates[2 * i + 1]])

    iter = 0
    for pair in projections:
        if pair is None:
            iter+=1
            continue
        projection = pair[0]
        if projection is None:
```

```
        iter+=1
        continue
    output_points = pair[1]

    #zanima me koja tacka se slika u (0,0)
    #p(input)=output
    #input = p^(-1)(output)
    top_left = applyProjection(projection, [0,0])
    top_right = applyProjection(projection, [w-1, 0])
    bottom_left = applyProjection(projection, [0, h-1])
    bottom_right = applyProjection(projection, [w-1, h-1])

    if (top_left is None) or (top_right is None) or
        (bottom_left is None) or (bottom_right is None):
        iter+=1
        continue

    start = [0,0]
    start[0] = max(top_left[0], bottom_left[0], 0)
    start[1] = max(top_left[1], top_right[1], 0)

    end = [0,0]

    end[0] = min(top_right[0], bottom_right[0], w-1)
    end[1] = min(bottom_left[1], bottom_right[1], h-1)

    uslov = True
    for output_point in output_points:
        if not(checkBound(output_point[0], start[0], end[0])) or
            not(checkBound(output_point[1], start[1], end[1])):
                uslov=False

    if not(uslov):
        iter+=1
        continue

    h_new = int(end[1] - start[1] + 1)
    w_new = int(end[0] - start[0] + 1)
    new_image = np.zeros(shape=(h_new, w_new, 3), dtype=np.uint8)

    projection_np = np.array(projection)
    projection_inv = np.linalg.inv(projection_np)

    break_ex = False
    for j in range(0, h_new):
        for i in range(0, w_new):

            xt = projection_inv[0][0] * (i+start[0]) + projection_inv[0][1] *
                                        (j+start[1]) + projection_inv[0][2]
            yt = projection_inv[1][0] *  (i+start[0])  + projection_inv[1][1] *
                                        (j+start[1]) + projection_inv[1][2]
            t = projection_inv[2][0] *  (i+start[0])  + projection_inv[2][1] *
                                        (j+start[1]) + projection_inv[2][2]
```

```
            if t==0:
                break_ex=True
            else:
                x = int(xt / t)
                y = int(yt / t)

                dx = xt / t - x
                dy = yt / t - y

                x = relu(x, w-1)
                y = relu(y, h-1)

                c00 = image[y, x]
                c10 = image[y, x + 1]
                c11 = image[y + 1, x + 1]
                c01 = image[y + 1, x]
                pix1 = c00 * dx + (1 - dx) * c10
                pix2 = c01 * dx + (1 - dx) * c11
                new_image[j, i] = pix1 * dy + (1 - dy) * pix2

    if break_ex:
        iter+=1
        continue

    image_name = path.split("\\")[-2]+"_"+path.split("\\")[-1][0:-4]+"_"+str(iter)
    +".ppm"
    cv.imwrite("C:\\Users\\bozica\\Desktop\\MojeSlike\\"+image_name, new_image)

    # List
    value = image_name
    points = map_path_to_coordinates[path]
    for i in range(0, 4):
        value += ";" + str(output_points[i][0]) + ";" + str(output_points[i][1])



    List = [value]
    # Open our existing CSV file in append mode
    # Create a file object for this file
    with open('C:\\Users\\bozica\\Desktop\\MojeSlike\\data.csv', 'a') as f_object:
    # Pass this file object to csv.writer()
    # and get a writer object
        writer_object = writer(f_object)

    # Pass the list as an argument into
    # the writerow()
        writer_object.writerow(List)
    iter+=1
```

А.6 Обучавање неуронске мреже за детекцију координата четири улазне тачке пројективног пресликавања

```
import numpy as np
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
import cv2 as cv
import csv
import copy
import pandas as pd
import random
import os
np.random.seed(7)

#the data is in the subpath C:\Users\bozica\Desktop\MojeSlike
#I have around 97 000 pictures. The target values are in data.csv

#I want to make a new data Frame that will have everything nicely
#prepared for the neural network.
#It should have the following colomns:
#    *pixels = image pixels in rgb(that are prolonged with black pixels so that
#                all the images have the same size)
#    *x1     = x coordinate of the top left corner of the bounding quadrangle
#    *y1     = y coordinate of the top left corner of the bounding quadrangle
#    *x2     = x coordinate of the top right corner of the bounding quadrangle
#    *y2     = y coordinate of the top right corner of the bounding quadrangle
#    *x3     = x coordinate of the bottom left corner of the bounding quadrangle
#    *y3     = y coordinate of the bottom left corner of the bounding quadrangle
#    *x4     = x coordinate of the bottom right corner of the bounding quadrangle
#    *y4     = y coordinate of the bottom right corner of the bounding quadrangle

data = {'Pixels':[], 'x1':[], 'y1':[], 'x2':[], 'y2':[],'x3':[], 'y3':[], 'x4':[], 'y4':[]}

path = 'C:\\Users\\bozica\\Desktop\\MojeSlike\\data.csv'
data0 = pd.read_csv(path)
n = len(data0.index)

#now lets add the targets x1,y1,x2,y2,x3,y3,x4,y4
x1 = np.zeros(shape=(n, 1), dtype=np.uint8)
y1 = np.zeros(shape=(n, 1), dtype=np.uint8)
x2 = np.zeros(shape=(n, 1), dtype=np.uint8)
```

```
y2 = np.zeros(shape=(n, 1), dtype=np.uint8)
x3 = np.zeros(shape=(n, 1), dtype=np.uint8)
y3 = np.zeros(shape=(n, 1), dtype=np.uint8)
x4 = np.zeros(shape=(n, 1), dtype=np.uint8)
y4 = np.zeros(shape=(n, 1), dtype=np.uint8)

i = 0
list_of_image_paths = []
for j in range(0,n):
    list_of_image_paths.append("")

for row in data0.iterrows():
    list_of_image_paths[i] = 'C:\\Users\\bozica\\Desktop\\MojeSlike\\'+
                row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[0]
    image = cv.imread(list_of_image_paths[i])
    h, w = image.shape[0:2]
    x1[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[1])/w)
    y1[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[2])/h)
    x2[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[3])/w)
    y2[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[4])/h)
    x3[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[5])/w)
    y3[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[6])/h)
    x4[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[7])/w)
    y4[i] = int(32*float(row[1]['Filename;x1;y1;x2;y2;x3;y3;x4;y4'].split(';')[8])/h)
    i+=1

n = len(list_of_image_paths)
print("Number of images:", n)

pixels = np.zeros(shape = (n, 32, 32, 3), dtype=np.uint8)
image_shapes = []
for i in range(0, n):
    image = cv.imread(list_of_image_paths[i])
    image_shapes.append(image.shape)
    image = cv.resize(image, (32, 32))
    pixels[i] = image

#setting the pixel values for all images in the dataset
data['Pixels'] = pixels

data['x1'] = x1
data['y1'] = y1
data['x2'] = x2
data['y2'] = y2
data['x3'] = x3
data['y3'] = y3
data['x4'] = x4
data['y4'] = y4

#final dataframe
df = pd.Series(data).to_frame()

#we need to set what is the input data(X) and what is the target(output data, Y)
```

```
X = data['Pixels']
print(X.shape)
Y = np.column_stack((x1, y1, x2, y2, x3, y3, x4, y4))
print(Y.shape)
X = X.astype('float32')
X /= 255

#let's divide the training and testing data
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
                                        test_size = 0.33, random_state = 7)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

#(65418, 32, 32, 3)
#(32222, 32, 32, 3)
#(65418, 8)
#(32222, 8)

input_shape = (32, 32, 3)

#creatining the neural network model
model = Sequential([
    Input(shape=input_shape),
    Conv2D(filters=64, kernel_size=(5, 5), strides=(1, 1), padding='same',
    activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=64, kernel_size=(5, 5), strides=(1, 1), padding='same',
    activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=64, kernel_size=(5, 5), strides=(1, 1), padding='same',
    activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(8, activation='linear')
])

model.compile(loss='mse', optimizer=Adam(learning_rate=0.001), metrics=['mae'])

batch_size = 32
epochs = 10

history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
```

```
                    validation_split=0.2)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title('Loss')
plt.plot(np.arange(0, epochs), history.history['loss'], label='train')
plt.plot(np.arange(0, epochs), history.history['val_loss'], label='val')
plt.legend(loc='best')

plt.subplot(1, 2, 2)
plt.title('mae')
plt.plot(np.arange(0, epochs), history.history['mae'], label='train')
plt.plot(np.arange(0, epochs), history.history['val_mae'], label='val')
plt.legend(loc='best')

plt.show()

score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test mae:', score[1])

#Test loss: 1.9290950298309326
#Test mae: 1.0383319854736328

Y_predicted = model.predict(X_test)

metrics.mean_squared_error(Y_predicted, Y_test)

#1.9290897

fig = plt.figure(figsize=(20, 16))
columns = 4
rows = 5
for i in range(1, rows*columns +1):
    r = random.randint(0,X_test.shape[0])
    image = X_test[r]
    result = Y_predicted[r]


    h = image_shapes[r][0]
    w = image_shapes[r][1]

    x1 = int(w*result[0]/32)
    y1 = int(h*result[1]/32)
    x2 = int(w*result[2]/32)
    y2 = int(h*result[3]/32)
    x3 = int(w*result[4]/32)
    y3 = int(h*result[5]/32)
    x4 = int(w*result[6]/32)
    y4 = int(h*result[7]/32)

    image2 = copy.deepcopy(X_test[r])
```

```
    image2 = cv.resize(image2, (h, w))
    cv.line(image2, (x1, y1), (x2,y2), (0,255,0))
    cv.line(image2, (x1, y1), (x3, y3), (0,255,0))
    cv.line(image2, (x3, y3), (x4, y4), (0,255,0))
    cv.line(image2, (x2, y2), (x4,y4), (0,255,0))
    fig.add_subplot(rows, columns, i)
    plt.imshow(image2)
plt.show()

model.save('C:\\Users\\bozica\\Desktop\\deljeni_folder\\MasterRad\\FourPointFinder')
```

A.7 Покретање неуронске мреже за проналажење координата четири улазне тачке пројективног пресликавања на вишепроцесоркој плочи

```
res = subprocess.run(['scp', '/c/Users/bozica/Desktop/input_2_memory.ppm',
'root@10.10.96.182:/home/root/bozica'])
res = subprocess.run(['ssh', 'root@10.10.96.182', 'cd', 'bozica' + ';',
'./final_app_template_HIL_GNU_linux', '--bcl', 'subn0_commandlist.bcl', '--commandlist',
'subn0_cl_cnn0.bin', '--core', 'CNN', '--sync-id', '1',
'--commandlist', 'subn0_cl_cve0.bin', '--core', 'CVE',
'--sync-id', '4', '--commandlist', 'subn0_cl_dma0.bin',
'--core', 'DMA', '--sync-id', '2', '--setMemory',
'input_2_memory:0,input_2_memory.ppm,0', '--setMemory',
'input_2_memory:1,input_2_memory.ppm,1', '--setMemory',
'input_2_memory:2,input_2_memory.ppm,2'])
res = subprocess.run(['scp', 'root@10.10.96.182:/home/root/bozica/'+
'statefulpartitionedcall_sequential_1_dense_5_matmul_memory.csv',
'/c/Users/bozica/Desktop/koordinate.csv'])
f = open('C:\\Users\\bozica\\Desktop\\koordinate.csv', 'r')
linije = f.read().split('\n')
f.close()
tacke = []
for i in range(1,9):
    tacke.append(int(linije[i][:-2], base=16) * (2**(-10)))

f = open('C:\\Users\\bozica\\Desktop\\koordinate.txt', 'w')
for tacka in tacke:
    f.write(str(tacka)+' ')

f.close()
```

A.8 Неоптимизован кôд отклањања перспективе

```
//ova funkcija racuna inverz matrice A, koji smesta u matricu B
//povratna vrednost je 0 ako je sve ok, -1 ako je det(A) = 0
int invers_3(float A[][3], float B[][3]){

    float a = A[0][0];
    float b = A[0][1];
    float c = A[0][2];

    float d = A[1][0];
    float e = A[1][1];
    float f = A[1][2];

    float g = A[2][0];
    float h = A[2][1];
    float i = A[2][2];

    float det = a*(e*i - h*f) - b*(d*i - g*f) + c*(d*h - g*e);

    if(det == 0)
        return -1;

    B[0][0] = (e*i - h*f)/det;
    B[1][0] = -(d*i - g*f)/det;
    B[2][0] = (d*h - g*e)/det;

    B[0][1] = -(b*i - h*c)/det;
    B[1][1] = (a*i - g*c)/det;
    B[2][1] = -(a*h - g*b)/det;

    B[0][2] = (b*f - e*c)/det;
    B[1][2] = -(a*f - d*c)/det;
    B[2][2] = (a*e - d*b)/det;

    return 0;

}


void bilinear_interpolation(unsigned width_in, unsigned width_out, unsigned height_out,
float projection_inv[][3], uint8_t pixels_rgb_in[], uint8_t pixels_rgb_out[]){
    /*transformacija: ulaz->izlaz*/
    unsigned x,y,t;
    //homogene koordinate piksela izlazne slike
    float pinv_tx, pinv_ty, pinv_t;
    //=pinv*(x,y,t)^T tj. homogene koordinate ulazne slike -> (x,y,t)^T
    unsigned pinv_x, pinv_y;
    // pinv_x = pinv_tx/pinv_t, slicno za pinv_y
    float dx, dy;
    // greske pri zaokruzivanju za pinv_x, pinv_y
    unsigned c00, c01, c10, c11;
    //okolni pikseli za bilineranu interpolaciju
    unsigned width_out_rgb = width_out*3;
    unsigned width_in_rgb = width_in*3;
```

```
    int i,j,k;


    for(i=0; i<height_out; i++){ //za svaki red
        for(j=0; j<width_out; j++){ //idi redom po pikselima
            x = j; //kolona
            y = i; //vrsta
            t = 1; //prosirenje na 3d

            //homogene koordinate tacke koja se slika u (j, i, 1)
            pinv_tx = projection_inv[0][0]*x +
            projection_inv[0][1]*y + projection_inv[0][2]*t;
            pinv_ty = projection_inv[1][0]*x +
            projection_inv[1][1]*y + projection_inv[1][2]*t;
            pinv_t  = projection_inv[2][0]*x +
            projection_inv[2][1]*y + projection_inv[2][2]*t;

            //prave koordinate
            pinv_x = pinv_tx / pinv_t;
            pinv_y = pinv_ty / pinv_t;

            //koliko smo omasili pri zaokruzivanju
            dx = pinv_tx / pinv_t - pinv_x;
            dy = pinv_ty / pinv_t - pinv_y;


            //bilinearnom interpolacijom aproksimiramo vrednost
            for(k=0; k<3; k++){ //za rgb
                c00 = pixels_rgb_in[pinv_y*width_in_rgb + 3*pinv_x + k];
                c10 = pixels_rgb_in[pinv_y*width_in_rgb + 3*(pinv_x+1) + k];
                c01 = pixels_rgb_in[(pinv_y+1)*width_in_rgb + 3*pinv_x + k];
                c11 = pixels_rgb_in[(pinv_y+1)*width_in_rgb + 3*(pinv_x+1) + k];

                pixels_rgb_out[y*width_out_rgb + 3*x + k] =
                lerp(lerp(c00, c10, dx), lerp(c01, c11, dx), dy);

            }
        }
    }
}


void nearest_neighbor(unsigned width_in, unsigned width_out, unsigned height_out,
float projection_inv[][3], uint8_t pixels_rgb_in[], uint8_t pixels_rgb_out[]){
        /*transformacija: ulaz->izlaz*/
    unsigned x,y,t;
    //homogene koordinate piksela izlazne slike
    float pinv_tx, pinv_ty, pinv_t;
    //=pinv*(x,y,t)^T tj. homogene koordinate ulazne slike -> (x,y,t)^T
    unsigned pinv_x, pinv_y;
    // pinv_x = pinv_tx/pinv_t, slicno za pinv_y
    float dx, dy;
    // greske pri zaokruzivanju za pinv_x, pinv_y
```

```
    unsigned width_out_rgb = width_out*3;
    unsigned width_in_rgb = width_in*3;
    int i,j,k;


    for(i=0; i<height_out; i++){ //za svaki red
        for(j=0; j<width_out; j++){ //idi redom po pikselima
            x = j; //kolona
            y = i; //vrsta
            t = 1; //prosirenje na 3d

            //homogene koordinate tacke koja se slika u (j, i, 1)
            pinv_tx = projection_inv[0][0]*x + projection_inv[0][1]*y +
            projection_inv[0][2]*t;
            pinv_ty = projection_inv[1][0]*x + projection_inv[1][1]*y +
            projection_inv[1][2]*t;
            pinv_t  = projection_inv[2][0]*x + projection_inv[2][1]*y +
            projection_inv[2][2]*t;

            //prave koordinate
            pinv_x = pinv_tx / pinv_t;
            pinv_y = pinv_ty / pinv_t;

            //koliko smo omasili pri zaokruzivanju
            dx = pinv_tx / pinv_t - pinv_x;
            dy = pinv_ty / pinv_t - pinv_y;

            if(dx>0.5){
                pinv_x++;
            }

            if(dy>0.5){
                pinv_y++;
            }

            for(k=0; k<3; k++){ //za rgb
                pixels_rgb_out[y*width_out_rgb + 3*x + k] =
                pixels_rgb_in[pinv_y*width_in_rgb + 3*pinv_x + k];
            }
        }
    }
}


TEST(C_Group, TEST_ARM_PROJECTION_TRANSFORMATION){


    void    * dataBuffer;
    uint8_t  ppm;
    size_t   nChannels;
    size_t   width_in, height_in, width_in_rgb;
    size_t   width_out, height_out, width_out_rgb;
    unsigned char * input_img  = NULL;
```

```
unsigned char * output_img = NULL;
uint8_t  * pixels_rgb_in = NULL;
uint8_t  * pixels_rgb_out = NULL;
struct timespec time_s, time_e;

//input_img se alocira tokom funkcije read_ppm
input_img = read_ppm("parking.ppm");
if(input_img==NULL) exit(EXIT_FAILURE);

/* Read PPM image header */
//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(input_img, &dataBuffer, &width_in, &height_in, &ppm,
&nChannels))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

if (nChannels != 3)
{
    printf("Invalid number of input channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

width_in_rgb = width_in * 3;
pixels_rgb_in = gf_Memalign(64, sizeof(uint8_t)*width_in_rgb *height_in);
TEST_ASSERT_NOT_NULL(pixels_rgb_in);

seperatePPMChannelRGB8(dataBuffer, width_in, height_in, pixels_rgb_in);


width_out = 300;
width_out_rgb = width_out*3;
height_out = 300;

unsigned input_points[4][2];
unsigned output_points[4][2];

input_points[0][0] = 280; input_points[0][1] = 125;
input_points[1][0] = 270; input_points[1][1] = 410;
input_points[2][0] = 580; input_points[2][1] = 355;
input_points[3][0] = 520; input_points[3][1] = 40;

output_points[0][0] = 0; output_points[0][1] = 0;
output_points[1][0] = 0; output_points[1][1] = height_out;
output_points[2][0] = width_out; output_points[2][1] = height_out;
output_points[3][0] = width_out; output_points[3][1] = 0;

/*pronalazenje odgovarajuceg projektivnog preslikavanja*/
int i,j,k;
```

```
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne
        ispunjavaju uslov da ni jedna trojka tacaka nije
        kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne
        poklapaju se ulazne i izlazne tacke u projektivnom
        preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }
    }

    /*pronalazenje inverza nadjenog projektivnog preslikavanja*/
    float projection_inv[3][3];
    result = invers_3(projection, projection_inv);
    if(result == -1){
         printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
         return;
    }

    printf("\nInverz projektivnog preslikavanja:\n");
    for(i=0; i<3; i++){
           for(j=0; j<3; j++){
               printf("%lf ", projection_inv[i][j]);
           }
        printf("\n");
    }

    pixels_rgb_out = gf_Memalign(64, sizeof(uint8_t)*width_out_rgb*height_out);
    TEST_ASSERT_NOT_NULL(pixels_rgb_out);

    clock_gettime(CLOCK_MONOTONIC, &time_s);
    bilinear_interpolation(width_in, width_out, height_out,
    projection_inv, pixels_rgb_in, pixels_rgb_out);
    clock_gettime(CLOCK_MONOTONIC, &time_e);

    write_ppm("./outputARM.ppm", pixels_rgb_out, width_out, height_out);
    printf("\nTEST_ARM_PROJECTION_TRANSFORMATION [ms]:%f\n\n",
    (double)(time_e.tv_nsec - time_s.tv_nsec) / 1000000.0 +
    (double)(time_e.tv_sec - time_s.tv_sec) * 1000.0);

    gf_Free(pixels_rgb_in);
    gf_Free(pixels_rgb_out);
```

```
}

TEST(C_Group, TEST_ARM_PROJECTION_TRANSFORMATION_NEIGHBOR){

    void   * dataBuffer;
    uint8_t  ppm;
    size_t   nChannels;
    size_t   width_in, height_in, width_in_rgb;
    size_t   width_out, height_out, width_out_rgb;
    unsigned char * input_img  = NULL;
    unsigned char * output_img = NULL;
    uint8_t  * pixels_rgb_in = NULL;
    uint8_t  * pixels_rgb_out = NULL;
    struct timespec time_s, time_e;

    //input_img se alocira tokom funkcije read_ppm
    input_img = read_ppm("parking.ppm");
    if(input_img==NULL) exit(EXIT_FAILURE);

    /* Read PPM image header */
    //dataBuffer ce pokazivati na neki deo input_img
    //tj. cuvace samo adresu
    if (!getPPMPGMImageInfo(input_img, &dataBuffer, &width_in, &height_in, &ppm,
    &nChannels))
    {
        printf("Failed to decode ppm image header\n");
        printf("ABORT\n");
        exit(EXIT_FAILURE);
    }

    if (nChannels != 3)
    {
        printf("Invalid number of input channels\n");
        printf("ABORT\n");
        exit(EXIT_FAILURE);
    }

    width_in_rgb = width_in * 3;

    pixels_rgb_in = gf_Memalign(64, sizeof(uint8_t)*width_in_rgb *height_in);
    TEST_ASSERT_NOT_NULL(pixels_rgb_in);

    seperatePPMChannelRGB8(dataBuffer, width_in, height_in, pixels_rgb_in);

    width_out = 300;
    width_out_rgb = width_out*3;
    height_out = 300;

    unsigned input_points[4][2];
    unsigned output_points[4][2];

    input_points[0][0] = 280; input_points[0][1] = 125;
```

```
    input_points[1][0] = 270; input_points[1][1] = 410;
    input_points[2][0] = 580; input_points[2][1] = 355;
    input_points[3][0] = 520; input_points[3][1] = 40;

    output_points[0][0] = 0; output_points[0][1] = 0;
    output_points[1][0] = 0; output_points[1][1] = height_out;
    output_points[2][0] = width_out; output_points[2][1] = height_out;
    output_points[3][0] = width_out; output_points[3][1] = 0;

    /*pronalazenje odgovarajuceg projektivnog preslikavanja*/
    int i,j,k;
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne
ispunjavaju uslov da ni jedna trojka tacaka nije
kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne
poklapaju se ulazne i izlazne tacke u projektivnom
preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }
    }

    /*pronalazenje inverza nadjenog projektivnog preslikavanja*/
    float projection_inv[3][3];
    result = invers_3(projection, projection_inv);
    if(result == -1){
         printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
         return;
    }

    printf("\nInverz projektivnog preslikavanja:\n");
    for(i=0; i<3; i++){
          for(j=0; j<3; j++){
              printf("%lf ", projection_inv[i][j]);
          }
        printf("\n");
    }

    pixels_rgb_out = gf_Memalign(64, sizeof(uint8_t)*width_out_rgb*height_out);
    TEST_ASSERT_NOT_NULL(pixels_rgb_out);

    clock_gettime(CLOCK_MONOTONIC, &time_s);
```

```
    nearest_neighbor(width_in, width_out, height_out, projection_inv, pixels_rgb_in,
    pixels_rgb_out);
    clock_gettime(CLOCK_MONOTONIC, &time_e);

    write_ppm("./output_ARM_neighbor.ppm", pixels_rgb_out, width_out, height_out);
    printf("\nTEST_ARM_PROJECTION_TRANSFORMATION_NEIGHBOR_[ms]:%f\n\n",
    (double)(time_e.tv_nsec - time_s.tv_nsec) / 1000000.0 +
    (double)(time_e.tv_sec - time_s.tv_sec) * 1000.0);

    gf_Free(pixels_rgb_in);
    gf_Free(pixels_rgb_out);

}
```

A.9 Оптимизација $1CVe$

```
#bilinearni cve kod
#include <stdint.h>

float lerp(float s, float e, float t){return s+(e-s)*t;}

float blerp(float c00, float c10, float c01, float c11, float tx, float ty){
    return lerp(lerp(c00, c10, tx), lerp(c01, c11, tx), ty);
}
const int l = 0;

#pragma section bss lwm
float projection_invers[9];

#pragma entry main master
void main(void)
{
    __setSrcImageID(1);



    int k;
    for(k=0; k<9; k++)
        projection_invers[k] = __getSrcf(k - __getX(), - __getY());


    __setSrcImageID(0);
    __setDstImageID(2);

    unsigned x,y,t; //homogene koordinate piksela izlazne slike
    float pinv_tx, pinv_ty, pinv_t;
    //=pinv*(x,y,t)^T tj. homogene koordinate ulazne slike -> (x,y,t)^T
    unsigned pinv_x, pinv_y; // pinv_x = pinv_tx/pinv_t, slicno za pinv_y
    float dx, dy; // greske pri zaokruzivanju za pinv_x, pinv_y
    unsigned c00, c01, c10, c11; //okolni pikseli za bilineranu interpolaciju
```

```
    //prave koordinate
    x = __getX()/3; //kolona
    y = __getY(); //vrsta
    t = 1; //prosirenje na 3d

    //homogene koordinate tacke koja se slika u (x, y, 1) sa P
    pinv_tx = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
    pinv_ty = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
    pinv_t  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

   //prave koordinate
    pinv_x = pinv_tx / pinv_t;
    pinv_y = pinv_ty / pinv_t;

    //koliko smo omasili pri zaokruzivanju
    dx = pinv_tx / pinv_t - pinv_x;
    dy = pinv_ty / pinv_t - pinv_y;


    //bilinearnom interpolacijom aproksimiramo vrednost
    for(k=0; k<3; k++){ //za rgb
        c00 = __getSrc(3*pinv_x + k - __getX(), pinv_y  - __getY());
        c10 = __getSrc(3*(pinv_x+1) + k - __getX(), pinv_y - __getY());
        c01 = __getSrc(3*pinv_x + k - __getX(), pinv_y + 1 - __getY());
        c11 = __getSrc(3*(pinv_x+1) + k - __getX(), pinv_y + 1 - __getY());

        float rez = lerp(lerp(c00, c10, dx), lerp(c01, c11, dx), dy);
        __setDst(k, 0, (uint32_t)rez);

    }

    __trap();

}


#aproksimacija najblizim susedom u cve kodu
#include <stdint.h>

const int l = 0;

#pragma section bss lwm
float projection_invers[9];

#pragma entry main master
void main(void)
{
    __setSrcImageID(1);



    int k;
    for(k=0; k<9; k++)
```

```
        projection_invers[k] = __getSrcf(k - __getX(), - __getY());


    __setSrcImageID(0);
    __setDstImageID(2);

    unsigned x,y,t; //homogene koordinate piksela izlazne slike
    float pinv_tx, pinv_ty, pinv_t;
    //=pinv*(x,y,t)^T tj. homogene koordinate ulazne slike -> (x,y,t)^T
    unsigned pinv_x, pinv_y; // pinv_x = pinv_tx/pinv_t, slicno za pinv_y
    float dx, dy; // greske pri zaokruzivanju za pinv_x, pinv_y

    //prave koordinate
    x = __getX()/3; //kolona
    y = __getY(); //vrsta
    t = 1; //prosirenje na 3d

    //homogene koordinate tacke koja se slika u (x, y, 1) sa P
    pinv_tx = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
    pinv_ty = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
    pinv_t  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

   //prave koordinate
    pinv_x = pinv_tx / pinv_t;
    pinv_y = pinv_ty / pinv_t;

    //koliko smo omasili pri zaokruzivanju
    dx = pinv_tx / pinv_t - pinv_x;
    dy = pinv_ty / pinv_t - pinv_y;

    if(dx>0.5){
        pinv_x++;
    }

    if(dy>0.5){
        pinv_y++;
    }


    for(k=0; k<3; k++){ //za rgb
        __setDst(k, 0, __getSrc(3*pinv_x + k - __getX(), pinv_y  - __getY()));

    }

    __trap();

}

#r_test_cve.c ya bliniearnu interpolaciju

int compare_pixels(uint8_t *original, uint8_t *copy, int length){

    int i;
```

```
    int num_of_diff_pixels = 0;
    for(i=0; i<length; i++)
        if(abs(copy[i] - original[i])>1)
            num_of_diff_pixels++;

    return num_of_diff_pixels;

}


TEST(C_Group, TEST_1CVe_PROJECTION_TRANSFORMATION){
    R_ATMLIB_CLData clData0;
    R_ATMLIB_OCVRectParam rect0;
    uint32_t data[10];
    uint8_t  program[CVE_size_of_cve_projection_text] = {CVE_cve_projection_text};
    uint8_t  unifirom[CVE_size_of_cve_projection_uniform] = {CVE_cve_projection_uniform};
    uint8_t *progMem      = NULL;
    uint32_t  *memories[3] = {NULL, NULL, NULL};
    struct timespec time_s, time_e;



    void   * dataBuffer;
    uint8_t  ppm;
    size_t   nChannels;
    size_t   width_in, height_in, width_in_rgb;
    size_t   width_out, height_out, width_out_rgb;
    unsigned char * input_img  = NULL;
    unsigned char * output_img = NULL;

    void   * dataBuffer_ARM;
    uint8_t  ppm_ARM;
    size_t   nChannels_ARM;
    size_t   width_ARM, height_ARM;
    unsigned char * output_image_ARM  = NULL;
    uint8_t * output_image_ARM_rgb_pixels = NULL;



    //input_img se alocira tokom funkcije read_ppm
    input_img = read_ppm("parking.ppm");
    if(input_img==NULL) exit(EXIT_FAILURE);

    /* Read PPM image header */
    //dataBuffer ce pokazivati na neki deo input_img
    //tj. cuvace samo adresu
    if (!getPPMPGMImageInfo(input_img, &dataBuffer,
    &width_in, &height_in, &ppm, &nChannels))
    {
        printf("Failed to decode ppm image header\n");
        printf("ABORT\n");
        exit(EXIT_FAILURE);
    }
```

```
    if (nChannels != 3)
    {
        printf("Invalid number of input channels\n");
        printf("ABORT\n");
        exit(EXIT_FAILURE);
    }

    width_in_rgb = width_in * 3;
    width_out = 300;
    width_out_rgb = width_out*3;
    height_out = 300;

    unsigned input_points[4][2];
    unsigned output_points[4][2];


    input_points[0][0] = 280; input_points[0][1] = 125;
    input_points[1][0] = 270; input_points[1][1] = 410;
    input_points[2][0] = 580; input_points[2][1] = 355;
    input_points[3][0] = 520; input_points[3][1] = 40;

    output_points[0][0] = 0; output_points[0][1] = 0;
    output_points[1][0] = 0; output_points[1][1] = height_out;
    output_points[2][0] = width_out; output_points[2][1] = height_out;
    output_points[3][0] = width_out; output_points[3][1] = 0;


    /*pronalazenje odgovarajuceg projektivnog preslikavanja*/
    int i,j,k;
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne
        ispunjavaju uslov da ni jedna trojka tacaka nije
        kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne
        poklapaju se ulazne i izlazne tacke u projektivnom
        preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }
```

```
}


//pronalazenje inverza nadjenog projektivnog preslikavanja
float projection_inv[3][3];
result = invers_3(projection, projection_inv);
if(result == -1){
     printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
     return;
}

printf("\nInverz projektivnog preslikavanja:\n");
for(i=0; i<3; i++){
       for(j=0; j<3; j++){
            printf("%lf ", projection_inv[i][j]);
       }
    printf("\n");
}

uint8_t* input_image_rgb_pixels = NULL;
input_image_rgb_pixels = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)*width_in_rgb
*height_in);
TEST_ASSERT_NOT_NULL(input_image_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer, width_in, height_in, input_image_rgb_pixels);

progMem = gf_Memalign(64, CVE_size_of_cve_projection_text);
memories[0] = (uint32_t *)gf_Memalign(64, sizeof(uint32_t)*width_in_rgb *height_in);
memories[1] = (float *)gf_Memalign(64, sizeof(float)*9);
memories[2] = (uint32_t *)gf_Memalign(64, sizeof(uint32_t)*width_out_rgb*height_out);

TEST_ASSERT_NOT_NULL(progMem);
TEST_ASSERT_NOT_NULL(memories[0]);
TEST_ASSERT_NOT_NULL(memories[1]);
TEST_ASSERT_NOT_NULL(memories[2]);

memset(memories[0], 0x00, width_in_rgb*height_in*sizeof(uint32_t));
memset(memories[1], 0x00, 9 * sizeof(float));
memset(memories[2], 0x00, width_out_rgb*height_out*sizeof(uint32_t));


for(i=0; i<height_in; i++){
    for(j=0; j<width_in_rgb; j++){
        *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j) =
        input_image_rgb_pixels[i*width_in_rgb+j];
    }
}

memcpy(memories[1], projection_inv, 9*sizeof(float));

/* Copy program to allocated memory for the ASM code */
memcpy(progMem, program, sizeof(program));
```

```
/* Prepare CL data structure for CVe0*/
clData0.cltype = R_ATMLIB_CLTYPE_OCV;
clData0.align  = 64;
clData0.size   = R_CL_SIZE;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData0));

data[0] = gf_GetPhysAddr(progMem);
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_VIBAR, 1, data));

data[0]   = 0x000000FF; /* 8 masters */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));

TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_UNR0,
CVE_size_of_cve_projection_uniform >> 2, (uint32_t*)unifirom));


data[0] = gf_GetPhysAddr(memories[0]); /* phy address */
data[1] = width_in_rgb * sizeof(uint32_t); /* stride */
data[2] = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));

data[0] = gf_GetPhysAddr(memories[1]); /* phy address */
data[1] = 9 * sizeof(float); /* stride */
data[2] = 0x00070107; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));

data[0] = gf_GetPhysAddr(memories[2]); /* phy address */
data[1] = width_out_rgb * sizeof(uint32_t); /* stride */
data[2] = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));


data[0]   = CVE_entry_master_cve_projection__main;
data[1]   = 0;
data[2]   = 0;
data[3]   = 0;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_VPCSAR, 4, data));


//RECT for CVe0

rect0.dx1  =     3                                          ;
rect0.dy2  =     0                                          ;
```

```
// vertical step
rect0.dx2  =     0                                          ;
rect0.dy1  =     1                                          ;
// rect size
rect0.xlen =     width_out                                  ;
rect0.ylen =     height_out                                 ;
// target point
rect0.xs   =     0                                          ;
rect0.ys   =     0                                          ;


/*Cv0 setup */


TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_RECT(&clData0, clData0.cur_addr -
clData0.top_addr, 0, 0, 0, &rect0));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_TRAP(&clData0, clData0.cur_addr -
clData0.top_addr, 0 ));



gf_DCacheFlushRange((uintptr_t)clData0.top_addr,gf_MallocUsableSize(clData0.top_addr));
gf_DCacheFlushRange((uintptr_t)progMem,         gf_MallocUsableSize(progMem));
gf_DCacheFlushRange((uintptr_t)memories[0],     gf_MallocUsableSize(memories[0]));
gf_DCacheFlushRange((uintptr_t)memories[1],     gf_MallocUsableSize(memories[1]));
gf_DCacheFlushRange((uintptr_t)memories[2],     gf_MallocUsableSize(memories[2]));

clock_gettime(CLOCK_MONOTONIC, &time_s);
/* Execute the IMP task              */
running_OCV0  = 1;

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
            &(getControl()->rcvdrvctl),
            (RCvUint)gf_GetPhysAddr(clData0.top_addr),
            RCVDRV_CORE_OCV0,
            core_map,
            CallBackFuncCVe0));


while(running_OCV0)
{
    gf_SuspendTask(1);
}

clock_gettime(CLOCK_MONOTONIC, &time_e);

gf_DCacheInvalidateRange((uintptr_t)memories[0], gf_MallocUsableSize(memories[0]));
gf_DCacheInvalidateRange((uintptr_t)memories[1], gf_MallocUsableSize(memories[1]));
gf_DCacheInvalidateRange((uintptr_t)memories[2], gf_MallocUsableSize(memories[2]));
```

```
uint8_t* output_image_rgb_pixels = NULL;
output_image_rgb_pixels = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(output_image_rgb_pixels);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out_rgb; j++)
        output_image_rgb_pixels[i*width_out_rgb+j] =
        *(uint32_t*)((uint32_t*)memories[2]+i*width_out_rgb+j);




output_image_ARM = read_ppm("outputARM.ppm");
if(output_image_ARM==NULL) exit(EXIT_FAILURE);

//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(output_image_ARM, &dataBuffer_ARM,
&width_ARM, &height_ARM, &ppm_ARM, &nChannels_ARM))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels_ARM != 3)
{
    printf("Invalid number of ARM output channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

output_image_ARM_rgb_pixels = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)*
width_ARM* 3 *height_ARM);
TEST_ASSERT_NOT_NULL(output_image_ARM_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer_ARM, width_ARM, height_ARM,
output_image_ARM_rgb_pixels);

printf("Provera korektnosti izlaza: ");
if(width_out != width_ARM || height_out != height_ARM){
    printf("DIMENZIJE SLIKE NISU DOBRE\n");
}else{

    int num_of_diff_pixels = compare_pixels(output_image_ARM_rgb_pixels,
    output_image_rgb_pixels, width_out_rgb*height_out);
    if(num_of_diff_pixels != 0){
        printf("BROJ RAZLICITIH PIKSELA JE %d\n", num_of_diff_pixels);
    }else{
```

```
            printf("IZLAZI SU ISTI\n");
        }
    }

    uint8_t *pixel_differences = NULL;
    pixel_differences = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)*
    width_out_rgb*height_out);
    TEST_ASSERT_NOT_NULL(pixel_differences);


    for(i=0; i<height_out; i++)
        for(j=0; j<width_out; j++){
                if(output_image_rgb_pixels[i*width_out_rgb+3*j]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j] ||
                    output_image_rgb_pixels[i*width_out_rgb+3*j+1]!=
                    output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+1] ||
                    output_image_rgb_pixels[i*width_out_rgb+3*j+2]!=
                    output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+2]
                ){
                    pixel_differences[i*width_out_rgb+3*j] = 255;
                    pixel_differences[i*width_out_rgb+3*j+1] = 0;
                    pixel_differences[i*width_out_rgb+3*j+2] = 0;
                }
                else{
                    pixel_differences[i*width_out_rgb+3*j] = 0;
                    pixel_differences[i*width_out_rgb+3*j+1] = 0;
                    pixel_differences[i*width_out_rgb+3*j+2] = 0;
                }
        }

    write_ppm("./output1CVe.ppm", output_image_rgb_pixels, width_out, height_out);
    write_ppm("./output1CVepixel_differences.ppm", pixel_differences, width_out,
    height_out);
    printf("\nTEST_1CVe_IMAGE_SCALING image [ms]:%f\n\n",
    (double)(time_e.tv_nsec - time_s.tv_nsec) / 1000000.0 +
    (double)(time_e.tv_sec - time_s.tv_sec) * 1000.0);
    r_atmlib_ReleaseCL(&clData0);
    gf_Free(progMem);
    gf_Free(memories[0]);
    gf_Free(memories[1]);
    gf_Free(memories[2]);
    gf_Free(output_image_rgb_pixels);
    gf_Free(input_image_rgb_pixels);
    gf_Free(output_image_ARM_rgb_pixels);
    gf_Free(pixel_differences);
    free_resources();
}


#r_test_cve.c za najblizeg suseda

TEST(C_Group, TEST_1CVe_PROJECTION_TRANSFORMATION_NEIGHBOR){
    R_ATMLIB_CLData clData0;
```

```
R_ATMLIB_OCVRectParam rect0;
uint32_t data[10];
uint8_t  program[CVE_size_of_cve_projection_neighbor_text] =
{CVE_cve_projection_neighbor_text};
uint8_t  unifirom[CVE_size_of_cve_projection_neighbor_uniform] =
{CVE_cve_projection_neighbor_uniform};
uint8_t *progMem      =  NULL;
uint32_t  *memories[3] = {NULL, NULL, NULL};
struct timespec time_s, time_e;



void   * dataBuffer;
uint8_t  ppm;
size_t   nChannels;
size_t   width_in, height_in, width_in_rgb;
size_t   width_out, height_out, width_out_rgb;
unsigned char * input_img  = NULL;
unsigned char * output_img = NULL;

void   * dataBuffer_ARM;
uint8_t  ppm_ARM;
size_t   nChannels_ARM;
size_t   width_ARM, height_ARM;
unsigned char * output_image_ARM  = NULL;
uint8_t * output_image_ARM_rgb_pixels = NULL;



//input_img se alocira tokom funkcije read_ppm
input_img = read_ppm("parking.ppm");
if(input_img==NULL) exit(EXIT_FAILURE);

/* Read PPM image header */
//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(input_img, &dataBuffer, &width_in,
&height_in, &ppm, &nChannels))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels != 3)
{
    printf("Invalid number of input channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

width_in_rgb = width_in * 3;
```

```
    width_out = 300;
    width_out_rgb = width_out*3;
    height_out = 300;

    unsigned input_points[4][2];
    unsigned output_points[4][2];


    input_points[0][0] = 280; input_points[0][1] = 125;
    input_points[1][0] = 270; input_points[1][1] = 410;
    input_points[2][0] = 580; input_points[2][1] = 355;
    input_points[3][0] = 520; input_points[3][1] = 40;

    output_points[0][0] = 0; output_points[0][1] = 0;
    output_points[1][0] = 0; output_points[1][1] = height_out;
    output_points[2][0] = width_out; output_points[2][1] = height_out;
    output_points[3][0] = width_out; output_points[3][1] = 0;


    /*pronalazenje odgovarajuceg projektivnog preslikavanja*/
    int i,j,k;
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne
␣␣␣␣␣␣␣␣ispunjavaju uslov da ni jedna trojka tacaka nije kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne
␣␣␣␣␣␣␣␣poklapaju se ulazne i izlazne tacke u projektivnom preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }


    }


    //pronalazenje inverza nadjenog projektivnog preslikavanja
    float projection_inv[3][3];
    result = invers_3(projection, projection_inv);
    if(result == -1){
         printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
         return;
    }
```

```
printf("\nInverz projektivnog preslikavanja:\n");
for(i=0; i<3; i++){
        for(j=0; j<3; j++){
            printf("%lf ", projection_inv[i][j]);
        }
    printf("\n");
}

uint8_t* input_image_rgb_pixels = NULL;
input_image_rgb_pixels =
(uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_in_rgb
*height_in);
TEST_ASSERT_NOT_NULL(input_image_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer, width_in, height_in, input_image_rgb_pixels);

progMem = gf_Memalign(64, CVE_size_of_cve_projection_neighbor_text);
memories[0] = (uint32_t *)gf_Memalign(64, sizeof(uint32_t)*width_in_rgb *height_in);
memories[1] = (float *)gf_Memalign(64, sizeof(float)*9);
memories[2] = (uint32_t *)gf_Memalign(64, sizeof(uint32_t)*width_out_rgb*height_out);

TEST_ASSERT_NOT_NULL(progMem);
TEST_ASSERT_NOT_NULL(memories[0]);
TEST_ASSERT_NOT_NULL(memories[1]);
TEST_ASSERT_NOT_NULL(memories[2]);

memset(memories[0], 0x00, width_in_rgb*height_in*sizeof(uint32_t));
memset(memories[1], 0x00, 9 * sizeof(float));
memset(memories[2], 0x00, width_out_rgb*height_out*sizeof(uint32_t));


for(i=0; i<height_in; i++){
    for(j=0; j<width_in_rgb; j++){
        *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j) =
        input_image_rgb_pixels[i*width_in_rgb+j];
    }
}

memcpy(memories[1], projection_inv, 9*sizeof(float));

/* Copy program to allocated memory for the ASM code */
memcpy(progMem, program, sizeof(program));
/* Prepare CL data structure for CVe0*/
clData0.cltype = R_ATMLIB_CLTYPE_OCV;
clData0.align  = 64;
clData0.size   = R_CL_SIZE;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData0));

data[0] = gf_GetPhysAddr(progMem);
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
```

```
r_atmlib_OCV_WPR(&clData0, clData0.cur_addr - clData0.top_addr,
R_IMPX5P_REG_OCV_VIBAR, 1, data));

data[0]    = 0x000000FF; /* 8 masters */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));

TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_UNR0,
CVE_size_of_cve_projection_neighbor_uniform >> 2,
(uint32_t*)unifirom));


data[0] = gf_GetPhysAddr(memories[0]); /* phy address */
data[1] = width_in_rgb * sizeof(uint32_t); /* stride */
data[2] = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));

data[0] = gf_GetPhysAddr(memories[1]); /* phy address */
data[1] = 9 * sizeof(float); /* stride */
data[2] = 0x00070107; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));

data[0] = gf_GetPhysAddr(memories[2]); /* phy address */
data[1] = width_out_rgb * sizeof(uint32_t); /* stride */
data[2] = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));


data[0]    = CVE_entry_master_cve_projection_neighbor__main;
data[1]    = 0;
data[2]    = 0;
data[3]    = 0;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_VPCSAR, 4, data));


//RECT for CVe0

rect0.dx1  =     3                                    ;
rect0.dy2  =     0                                    ;
// vertical step
rect0.dx2  =     0                                    ;
rect0.dy1  =     1                                    ;
// rect size
rect0.xlen =     width_out                            ;
rect0.ylen =     height_out                           ;
// target point
```

```
rect0.xs    =      0                                          ;
rect0.ys    =      0                                          ;


/*Cv0 setup */


TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_RECT(&clData0, clData0.cur_addr -
clData0.top_addr, 0, 0, 0, &rect0));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_TRAP(&clData0, clData0.cur_addr -
clData0.top_addr, 0 ));



gf_DCacheFlushRange((uintptr_t)clData0.top_addr,

gf_MallocUsableSize(clData0.top_addr));
gf_DCacheFlushRange((uintptr_t)progMem,          gf_MallocUsableSize(progMem));
gf_DCacheFlushRange((uintptr_t)memories[0],      gf_MallocUsableSize(memories[0]));
gf_DCacheFlushRange((uintptr_t)memories[1],      gf_MallocUsableSize(memories[1]));
gf_DCacheFlushRange((uintptr_t)memories[2],      gf_MallocUsableSize(memories[2]));

clock_gettime(CLOCK_MONOTONIC, &time_s);
/* Execute the IMP task              */
running_OCV0  = 1;

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
            &(getControl()->rcvdrvctl),
            (RCvUint)gf_GetPhysAddr(clData0.top_addr),
            RCVDRV_CORE_OCV0,
            core_map,
            CallBackFuncCVe0));


while(running_OCV0)
{
    gf_SuspendTask(1);
}

clock_gettime(CLOCK_MONOTONIC, &time_e);

gf_DCacheInvalidateRange((uintptr_t)memories[0], gf_MallocUsableSize(memories[0]));
gf_DCacheInvalidateRange((uintptr_t)memories[1], gf_MallocUsableSize(memories[1]));
gf_DCacheInvalidateRange((uintptr_t)memories[2], gf_MallocUsableSize(memories[2]));



uint8_t* output_image_rgb_pixels = NULL;
output_image_rgb_pixels = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(output_image_rgb_pixels);
```

```
for(i=0; i<height_out; i++)
    for(j=0; j<width_out_rgb; j++)
        output_image_rgb_pixels[i*width_out_rgb+j] =
        *(uint32_t*)((uint32_t*)memories[2]+i*width_out_rgb+j);




output_image_ARM = read_ppm("output_ARM_neighbor.ppm");
if(output_image_ARM==NULL) exit(EXIT_FAILURE);

//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(output_image_ARM, &dataBuffer_ARM,
&width_ARM, &height_ARM, &ppm_ARM, &nChannels_ARM))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels_ARM != 3)
{
    printf("Invalid number of ARM output channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

output_image_ARM_rgb_pixels =
(uint8_t *)gf_Memalign(64, sizeof(uint8_t)*width_ARM* 3 *height_ARM);
TEST_ASSERT_NOT_NULL(output_image_ARM_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer_ARM,
width_ARM, height_ARM, output_image_ARM_rgb_pixels);

printf("Provera korektnosti izlaza: ");
if(width_out != width_ARM || height_out != height_ARM){
    printf("DIMENZIJE SLIKE NISU DOBRE\n");
}else{

    int num_of_diff_pixels =
    compare_pixels(output_image_ARM_rgb_pixels,
    output_image_rgb_pixels, width_out_rgb*height_out);
    if(num_of_diff_pixels != 0){
        printf("BROJ RAZLICITIH PIKSELA JE %d\n", num_of_diff_pixels);
    }else{
        printf("IZLAZI SU ISTI\n");
    }
}

uint8_t *pixel_differences = NULL;
```

```
    pixel_differences = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)*width_out_rgb
    *height_out);
    TEST_ASSERT_NOT_NULL(pixel_differences);


    for(i=0; i<height_out; i++)
        for(j=0; j<width_out; j++){
                if(output_image_rgb_pixels[i*width_out_rgb+3*j]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j]
                ||
                    output_image_rgb_pixels[i*width_out_rgb+3*j+1]!=
                    output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+1]
                    ||
                    output_image_rgb_pixels[i*width_out_rgb+3*j+2]!=
                    output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+2]
                ){
                    pixel_differences[i*width_out_rgb+3*j] = 255;
                    pixel_differences[i*width_out_rgb+3*j+1] = 0;
                    pixel_differences[i*width_out_rgb+3*j+2] = 0;
                }
                else{
                    pixel_differences[i*width_out_rgb+3*j] = 0;
                    pixel_differences[i*width_out_rgb+3*j+1] = 0;
                    pixel_differences[i*width_out_rgb+3*j+2] = 0;
                }
        }

    write_ppm("./output_1CVe_neighbor.ppm", output_image_rgb_pixels, width_out,
    height_out);
    write_ppm("./output_1CVe_neighbor_pixel_differences.ppm",
    pixel_differences, width_out, height_out);
    printf("\nTEST_1CVe_PROJECTION_TRANSFORMATION_NEIGHBOR␣image␣[ms]:%f\n\n",
    (double)(time_e.tv_nsec - time_s.tv_nsec) / 1000000.0 +
    (double)(time_e.tv_sec - time_s.tv_sec) * 1000.0);
    r_atmlib_ReleaseCL(&clData0);
    gf_Free(progMem);
    gf_Free(memories[0]);
    gf_Free(memories[1]);
    gf_Free(memories[2]);
    gf_Free(output_image_rgb_pixels);
    gf_Free(input_image_rgb_pixels);
    gf_Free(output_image_ARM_rgb_pixels);
    gf_Free(pixel_differences);
    free_resources();
}
```

A.10 Оптимизација $2CVe$

```
TEST(C_Group, TEST_2CVe_PROJECTION_TRANSFORMATION){
    R_ATMLIB_CLData clData_0, clData_1;
    R_ATMLIB_OCVRectParam rect;
```

```
uint32_t     data[10];
float* floatp;
uint8_t program[CVE_size_of_cve_projection_text] = {CVE_cve_projection_text};
uint8_t uniform[CVE_size_of_cve_projection_uniform] = {CVE_cve_projection_uniform};
uint32_t *progMem = NULL;
uint32_t  *memories[3] = {NULL, NULL, NULL};
struct timespec time_s, time_e;

void    * dataBuffer;
uint8_t  ppm;
size_t    nChannels;
size_t    width_in, height_in, width_in_rgb;
size_t    width_out, height_out, width_out_rgb;
unsigned char * input_img  = NULL;
unsigned char * output_img = NULL;

void    * dataBuffer_ARM;
uint8_t  ppm_ARM;
size_t    nChannels_ARM;
size_t    width_ARM, height_ARM;
unsigned char * output_image_ARM  = NULL;
uint8_t * output_image_ARM_rgb_pixels = NULL;


input_img = read_ppm("parking.ppm");
if(input_img==NULL) exit(EXIT_FAILURE);

/* Read PPM image header */
//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(input_img, &dataBuffer, &width_in, &height_in,
&ppm, &nChannels))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels != 3)
{
    printf("Invalid number of input channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

width_in_rgb = width_in * 3;
width_out = 300;
height_out = 300;
width_out_rgb = width_out*3;

unsigned input_points[4][2];
unsigned output_points[4][2];
```

```
    input_points[0][0] = 280; input_points[0][1] = 125;
    input_points[1][0] = 270; input_points[1][1] = 410;
    input_points[2][0] = 580; input_points[2][1] = 355;
    input_points[3][0] = 520; input_points[3][1] = 40;

    output_points[0][0] = 0; output_points[0][1] = 0;
    output_points[1][0] = 0; output_points[1][1] = height_out;
    output_points[2][0] = width_out; output_points[2][1] = height_out;
    output_points[3][0] = width_out; output_points[3][1] = 0;


    /*pronalazenje odgovarajuceg projektivnog preslikavanja*/
    int i,j,k;
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne ispunjavaju uslov da ni
        jedna trojka tacaka nije kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne poklapaju se ulazne i izlazne
        tacke u projektivnom preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }


    }


    //pronalazenje inverza nadjenog projektivnog preslikavanja
    float projection_inv[3][3];
    result = invers_3(projection, projection_inv);
    if(result == -1){
         printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
         return;
    }

    printf("\nInverz projektivnog preslikavanja:\n");
    for(i=0; i<3; i++){
            for(j=0; j<3; j++){
                printf("%lf ", projection_inv[i][j]);
            }
        printf("\n");
    }
```

```
uint8_t* input_image_rgb_pixels = NULL;
input_image_rgb_pixels = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)
*width_in_rgb*height_in);
TEST_ASSERT_NOT_NULL(input_image_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer, width_in, height_in, input_image_rgb_pixels);



progMem = gf_Memalign(64, CVE_size_of_cve_projection_text);
memories[0] = (uint32_t *) gf_Memalign(64, width_in_rgb * height_in *
sizeof(uint32_t));
memories[1] = (float *) gf_Memalign(64, 9 * sizeof(float));
memories[2] = (uint32_t *) gf_Memalign(64, width_out_rgb * height_out
* sizeof(uint32_t));


TEST_ASSERT_NOT_NULL(progMem);
TEST_ASSERT_NOT_NULL(memories[0]);
TEST_ASSERT_NOT_NULL(memories[1]);
TEST_ASSERT_NOT_NULL(memories[2]);


memset(memories[0], 0x00, width_in_rgb*height_in*sizeof(uint32_t));
memset(memories[1], 0x00, 9 * sizeof(float));
memset(memories[2], 0x00, width_out_rgb*height_out*sizeof(uint32_t));


for(i=0; i<height_in; i++){
    for(j=0; j<width_in_rgb; j++){
        *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j) =
        input_image_rgb_pixels[i*width_in_rgb+j];
    }
}

memcpy(memories[1], projection_inv, 9*sizeof(float));

/* Copy program to allocated memory for the ASM code */
memcpy(progMem, program, sizeof(program));

/* Prepare CL data structure */
clData_0.cltype = R_ATMLIB_CLTYPE_OCV;
clData_0.align  = 64;
clData_0.size   = R_CL_SIZE;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData_0));

clData_1.cltype = R_ATMLIB_CLTYPE_OCV;
clData_1.align  = 64;
clData_1.size   = R_CL_SIZE;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData_1));
```

```
data[0] = gf_GetPhysAddr(progMem);
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr - clData_0.top_addr,
R_IMPX5P_REG_OCV_VIBAR, 1, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_VIBAR, 1, data));

TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr
- clData_0.top_addr, R_IMPX5P_REG_OCV_UNR0, CVE_size_of_cve_projection_uniform >> 2,
(uint32_t*)uniform));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_UNR0,
CVE_size_of_cve_projection_uniform >> 2,
(uint32_t*)uniform));

data[0] = 0x000000FF; /* 8 masters */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr - clData_0.top_addr,
R_IMPX5P_REG_OCV_TGSEN1, 1, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));

data[0] = gf_GetPhysAddr(memories[0]); /* phy address */
data[1] = width_in_rgb * sizeof(uint32_t); /* stride */
data[2] = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr
- clData_0.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));

data[0] = gf_GetPhysAddr(memories[1]); /* phy address */
data[1] = 9 * sizeof(float); /* stride */
data[2] = 0x00070107; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr
- clData_0.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));

data[0] = gf_GetPhysAddr(memories[2]); /* phy address */
data[1] = width_out_rgb * sizeof(uint32_t); /* stride */
data[2] = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr
- clData_0.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));


data[0] = CVE_entry_master_cve_projection__main;
data[1] = 0;
data[2] = 0;
data[3] = 0;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_0, clData_0.cur_addr
```

```
- clData_0.top_addr, R_IMPX5P_REG_OCV_VPCSAR, 4, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData_1, clData_1.cur_addr
- clData_1.top_addr, R_IMPX5P_REG_OCV_VPCSAR, 4, data));

rect.dx1 = 3;
rect.dy1 = 1;
rect.dx2 = 0;
rect.dy2 = 0;
rect.xlen = width_out;
rect.ylen = height_out / 2;
rect.xs = 0;
rect.ys = 0;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_RECT(&clData_0, clData_0.cur_addr
- clData_0.top_addr, 0, 0, 0, &rect));

rect.dx1 = 3;
rect.dy1 = 1;
rect.dx2 = 0;
rect.dy2 = 0;
rect.xlen = width_out;
rect.ylen = height_out - height_out / 2;
rect.xs = 0;
rect.ys = height_out / 2;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_RECT(&clData_1,
clData_1.cur_addr - clData_1.top_addr, 0, 0, 0, &rect));

TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_TRAP(&clData_0,
clData_0.cur_addr - clData_0.top_addr, 0 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_TRAP(&clData_1,
clData_1.cur_addr - clData_1.top_addr, 0 ));

gf_DCacheFlushRange((uintptr_t)clData_0.top_addr,
gf_MallocUsableSize(clData_0.top_addr));
gf_DCacheFlushRange((uintptr_t)clData_1.top_addr,
gf_MallocUsableSize(clData_1.top_addr));
gf_DCacheFlushRange((uintptr_t)progMem, gf_MallocUsableSize(progMem));
gf_DCacheFlushRange((uintptr_t)memories[0], gf_MallocUsableSize(memories[0]));
gf_DCacheFlushRange((uintptr_t)memories[1], gf_MallocUsableSize(memories[1]));
gf_DCacheFlushRange((uintptr_t)memories[2], gf_MallocUsableSize(memories[2]));

clock_gettime(CLOCK_MONOTONIC, &time_s);

/* Execute the IMP task              */
running_OCV0 = 1;
running_OCV1 = 1;

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
            &(getControl()->rcvdrvctl),
            (RCvUint)gf_GetPhysAddr(clData_0.top_addr),
            RCVDRV_CORE_OCV0,
            core_map,
            CallBackFuncCVe0));
```

```
TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
                &(getControl()->rcvdrvctl),
                (RCvUint)gf_GetPhysAddr(clData_1.top_addr),
                RCVDRV_CORE_OCV1,
                core_map,
                CallBackFuncCVe1));

while(running_OCV0 | running_OCV1)
{
    gf_SuspendTask(1);
}
clock_gettime(CLOCK_MONOTONIC, &time_e);

gf_DCacheInvalidateRange((uintptr_t)memories[0], gf_MallocUsableSize(memories[0]));
gf_DCacheInvalidateRange((uintptr_t)memories[1], gf_MallocUsableSize(memories[1]));
gf_DCacheInvalidateRange((uintptr_t)memories[2], gf_MallocUsableSize(memories[2]));

uint8_t* output_image_rgb_pixels = NULL;
output_image_rgb_pixels = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(output_image_rgb_pixels);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out_rgb; j++)
        output_image_rgb_pixels[i*width_out_rgb+j] =
        *(uint32_t*)((uint32_t*)memories[2]+i*width_out_rgb+j);




output_image_ARM = read_ppm("outputARM.ppm");
if(output_image_ARM==NULL) exit(EXIT_FAILURE);

//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(output_image_ARM, &dataBuffer_ARM, &width_ARM, &height_ARM,
&ppm_ARM, &nChannels_ARM))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels_ARM != 3)
{
    printf("Invalid number of ARM output channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

output_image_ARM_rgb_pixels = (uint8_t *)gf_Memalign(64,
```

```
sizeof(uint8_t)*width_ARM* 3 *height_ARM);
TEST_ASSERT_NOT_NULL(output_image_ARM_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer_ARM, width_ARM, height_ARM,
output_image_ARM_rgb_pixels);

printf("Provera korektnosti izlaza: ");
if(width_out != width_ARM || height_out != height_ARM){
    printf("DIMENZIJE SLIKE NISU DOBRE\n");
}else{

    int num_of_diff_pixels = compare_pixels(output_image_ARM_rgb_pixels,
    output_image_rgb_pixels, width_out_rgb*height_out);
    if(num_of_diff_pixels != 0){
        printf("BROJ RAZLICITIH PIKSELA JE %d\n", num_of_diff_pixels);
    }else{
        printf("IZLAZI SU ISTI\n");
    }
}

uint8_t *pixel_differences = NULL;
pixel_differences = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(pixel_differences);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out; j++){
            if(output_image_rgb_pixels[i*width_out_rgb+3*j]!=
            output_image_ARM_rgb_pixels[i*width_out_rgb+3*j] ||

                output_image_rgb_pixels[i*width_out_rgb+3*j+1]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+1] ||

                output_image_rgb_pixels[i*width_out_rgb+3*j+2]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+2]
            ){
                pixel_differences[i*width_out_rgb+3*j] = 255;
                pixel_differences[i*width_out_rgb+3*j+1] = 0;
                pixel_differences[i*width_out_rgb+3*j+2] = 0;
            }
            else{
                pixel_differences[i*width_out_rgb+3*j] = 0;
                pixel_differences[i*width_out_rgb+3*j+1] = 0;
                pixel_differences[i*width_out_rgb+3*j+2] = 0;
            }
    }

write_ppm("./output2CVe.ppm", output_image_rgb_pixels, width_out, height_out);
write_ppm("./output2CVepixel_differences.ppm", pixel_differences, width_out,
height_out);
printf("\nTEST_2CVe_IMAGE_SCALING image [ms]:%f\n\n", (double)(time_e.tv_nsec -
```

```
    time_s.tv_nsec) / 1000000.0 + (double)(time_e.tv_sec - time_s.tv_sec) * 1000.0);
    r_atmlib_ReleaseCL(&clData_0);
    r_atmlib_ReleaseCL(&clData_1);
    gf_Free(progMem);
    gf_Free(memories[0]);
    gf_Free(memories[1]);
    gf_Free(memories[2]);
    gf_Free(output_image_rgb_pixels);
    gf_Free(input_image_rgb_pixels);
    gf_Free(output_image_ARM_rgb_pixels);
    gf_Free(pixel_differences);
    free_resources();
}
```

A.11 Оптимизација *Scratchpad*

```
TEST(C_Group, TEST_iDMA_Scratchpad_PROJECTION_TRANSFORMATION){
    R_ATMLIB_CLData clData0, clData1;
    R_ATMLIB_CLData cl_iDMA;
    /* Input matrix (DDR->Scratchpad) copy parameters */
    R_ATMLIB_ImageParam      dma_param_img_input_to_scp;
    R_ATMLIB_DMASubsetParam dma_sub_param_input_to_scp;
    /* Output matrix (Scratchpad->DDR) copy parameters */
    R_ATMLIB_ImageParam      dma_param_img_output_to_ddr;
    R_ATMLIB_DMASubsetParam dma_sub_param_img_output_to_ddr;

    R_ATMLIB_OCVRectParam rect0, rect1;
    uint32_t data[10];
    uint8_t  program[CVE_size_of_cve_projection_idma_text]    =
    {CVE_cve_projection_idma_text};
    uint8_t  uniform[CVE_size_of_cve_projection_idma_uniform] =
    {CVE_cve_projection_idma_uniform};
    uint32_t *progMem      = NULL;
    uint32_t  *memories[2] = {NULL, NULL};
    uint32_t *memories64[3] = {NULL, NULL, NULL};
    struct timespec time_s, time_e;

    void   * dataBuffer;
    uint8_t  ppm;
    size_t   nChannels;
    size_t   width_in, height_in, width_in_rgb, width_in_64_rgb, width_in_64;
    size_t   width_out, height_out, width_out_rgb, width_out_64, width_out_64_rgb;
    unsigned char * input_img  = NULL;

    void   * dataBuffer_ARM;
    uint8_t  ppm_ARM;
    size_t   nChannels_ARM;
    size_t   width_ARM, height_ARM;
    unsigned char * output_image_ARM  = NULL;
    uint8_t * output_image_ARM_rgb_pixels = NULL;
    int i,j;
```

```
input_img = read_ppm("parking.ppm");
if(input_img==NULL) return;

/* Read PPM image header */
if (!getPPMPGMImageInfo(input_img, &dataBuffer, &width_in, &height_in, &ppm,
&nChannels))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    return;
}

if (nChannels != 3)
{
    printf("Invalid number of input channels\n");
    printf("ABORT\n");
    return;
}

width_in_rgb   = width_in * 3;
width_out = 300;
height_out = 300;

if(width_in % 64 != 0){
    width_in_64 = width_in + 64 - width_in%64;
    width_in_64_rgb = width_in_64*3;
}
else{
    width_in_64 = width_in;
    width_in_64_rgb = width_in_rgb;
}

if(width_out % 64 != 0){
    width_out_64 = width_out + 64 - width_out%64;
    width_out_64_rgb = width_out_64*3;
}
else{
    width_out_64 = width_out;
    width_out_64_rgb = width_out * 3;
}

unsigned input_points[4][2];
unsigned output_points[4][2];


input_points[0][0] = 280; input_points[0][1] = 125;
input_points[1][0] = 270; input_points[1][1] = 410;
input_points[2][0] = 580; input_points[2][1] = 355;
input_points[3][0] = 520; input_points[3][1] = 40;

output_points[0][0] = 0; output_points[0][1] = 0;
```

```
    output_points[1][0] = 0; output_points[1][1] = height_out;
    output_points[2][0] = width_out; output_points[2][1] = height_out;
    output_points[3][0] = width_out; output_points[3][1] = 0;


    /*pronalazenje odgovarajuceg projektivnog preslikavanja*/
    int k;
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne ispunjavaju uslov da ni
        jedna trojka tacaka nije kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne poklapaju se ulazne i izlazne
        tacke u projektivnom preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }


    }


    //pronalazenje inverza nadjenog projektivnog preslikavanja
    float projection_inv[3][3];
    result = invers_3(projection, projection_inv);
    if(result == -1){
         printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
         return;
    }

    printf("\nInverz projektivnog preslikavanja:\n");
    for(i=0; i<3; i++){
            for(j=0; j<3; j++){
                printf("%lf ", projection_inv[i][j]);
            }
        printf("\n");
    }

    uint8_t* input_image_rgb_pixels = NULL;
    input_image_rgb_pixels = (uint8_t *)gf_Memalign(64,
    sizeof(uint8_t)*width_in_rgb*height_in);
    TEST_ASSERT_NOT_NULL(input_image_rgb_pixels);


    seperatePPMChannelRGB8(dataBuffer, width_in, height_in, input_image_rgb_pixels);
```

```
progMem      = gf_Memalign(64, CVE_size_of_cve_projection_idma_text);
memories[0] = (uint32_t *)gf_Memalign(64,
width_in_rgb * height_in * sizeof(uint32_t));
memories[1] = (uint32_t *)gf_Memalign(64,
width_out_rgb * height_out * sizeof(uint32_t));
memories64[0] = (uint32_t *)gf_Memalign(64,
width_in_64_rgb * height_in * sizeof(uint32_t));
memories64[1] = (uint32_t *)gf_Memalign(64,
width_out_64_rgb * height_out * sizeof(uint32_t));
memories64[2] = (float *)gf_Memalign(64,
64*sizeof(float));



TEST_ASSERT_NOT_NULL(progMem);
TEST_ASSERT_NOT_NULL(memories[0]);
TEST_ASSERT_NOT_NULL(memories[1]);
TEST_ASSERT_NOT_NULL(memories64[0]);
TEST_ASSERT_NOT_NULL(memories64[1]);
TEST_ASSERT_NOT_NULL(memories64[2]);

memset(memories[0], 0, width_in_rgb * height_in * sizeof(uint32_t));
memset(memories[1], 0, width_out_rgb * height_out * sizeof(uint32_t));
memset(memories64[0], 0, width_in_64_rgb * height_in * sizeof(uint32_t));
memset(memories64[1], 0, width_out_64_rgb * height_out * sizeof(uint32_t));
memset(memories64[2], 0, 64 * sizeof(float));

for(i=0; i<height_in; i++){
    for(j=0; j<width_in_rgb; j++){
        *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j) =
        input_image_rgb_pixels[i*width_in_rgb+j];
    }
}

for(i=0; i<height_in; i++){
    for(j=0; j<width_in_rgb; j++){
        *(uint32_t*)((uint32_t*)memories64[0] + i*width_in_64_rgb + j) =
        *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j);
    }
}


memcpy(memories64[2], projection_inv, 9*sizeof(float));

/* Copy program to allocated memory for the ASM code */
memcpy(progMem, program, sizeof(program));

/* Prepare CL data structure for iDMA*/
cl_iDMA.cltype = R_ATMLIB_CLTYPE_DMA;
cl_iDMA.align  = 64;
cl_iDMA.size   = R_CL_SIZE;
```

```
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&cl_iDMA));

/* Prepare CL data structure for CVe0*/
clData0.cltype = R_ATMLIB_CLTYPE_OCV;
clData0.align  = 64;
clData0.size   = R_CL_SIZE;
/* Prepare CL data structure for CVe1*/
clData1.cltype = R_ATMLIB_CLTYPE_OCV;
clData1.align  = 64;
clData1.size   = R_CL_SIZE;
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData0));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData1));

data[0] = gf_GetPhysAddr(progMem);
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_VIBAR, 1, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1, clData1.cur_addr -
clData1.top_addr, R_IMPX5P_REG_OCV_VIBAR, 1, data));

TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_UNR0, CVE_size_of_cve_projection_idma_uniform
>> 2, (uint32_t*)uniform));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1, clData1.cur_addr -
clData1.top_addr, R_IMPX5P_REG_OCV_UNR0, CVE_size_of_cve_projection_idma_uniform
>> 2,(uint32_t*)uniform));

data[0]   = 0x000000FF; /* 8 masters */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1, clData1.cur_addr -
clData1.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));

data[0]   = MEM_SCRATCH_INPUT; /* phy address */
data[1]   = 64 * sizeof(float); /* stride */
data[2]   = 0x00070107; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr
- clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1, clData1.cur_addr
- clData1.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));

data[0]   = MEM_SCRATCH_INPUT + width_out_64_rgb; /* phy address */
data[1]   = width_in_64_rgb * sizeof(uint32_t); /* stride */
data[2]   = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1, clData1.cur_addr -
clData1.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));


data[0]   = MEM_SCRATCH_OUTPUT; /* phy address */
data[1]   = width_out_64_rgb * sizeof(uint32_t); /* stride */
data[2]   = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0, clData0.cur_addr -
```

```
    clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1, clData1.cur_addr -
    clData1.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));


    unsigned output_size_scp = 896*1024*4;
    unsigned num_of_iter = height_out*width_out_64_rgb/(output_size_scp -
    width_out_64_rgb)
    +(height_out*width_out_64_rgb%(output_size_scp-width_out_64_rgb) != 0);

    unsigned begining, end, Ax, Ay, Bx, By, Cx, Cy, Dx, Dy;
    *(uint8_t*)((uint8_t*)uniform + 1) = height_in;
    *(uint8_t*)((uint8_t*)uniform + 2) = height_out;
    *(uint8_t*)((uint8_t*)uniform + 3) = num_of_iter;


    for(i=0; i<num_of_iter; i++){
        *((uint8_t*)uniform) = i;

        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,

        clData0.cur_addr -
        clData0.top_addr, R_IMPX5P_REG_OCV_UNR0,

        CVE_size_of_cve_projection_idma_uniform >> 2, (uint32_t*)uniform));

        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,

        clData1.cur_addr -
        clData1.top_addr, R_IMPX5P_REG_OCV_UNR0,

        CVE_size_of_cve_projection_idma_uniform >> 2, (uint32_t*)uniform));

        data[0]   = CVE_entry_master_cve_projection_idma__main;
        data[1]   = 0;
        data[2]   = 0;
        data[3]   = 0;
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
        clData0.cur_addr - clData0.top_addr,
        R_IMPX5P_REG_OCV_VPCSAR, 4, data));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
        clData1.cur_addr - clData1.top_addr,
        R_IMPX5P_REG_OCV_VPCSAR, 4, data));


        //RECT for CVe0

        rect0.dx1  =     3
;
        rect0.dy2  =     0
;
        // vertical step
        rect0.dx2  =     0
```

```
;
        rect0.dy1  =     1
;
        // rect size
        rect0.xlen =     width_out
;
        rect0.ylen =     (i!=num_of_iter-1)*(height_out / num_of_iter / 2) +
        (i==num_of_iter-1)*((height_out - height_out/num_of_iter * (num_of_iter-1))/2)
;
        // target point
        rect0.xs   =     0
;
        rect0.ys   =     i*(height_out/num_of_iter)
;

        //RECT for CVe1

        rect1.dx1  =     3
;
        rect1.dy2  =     0
;
        // vertical step
        rect1.dx2  =     0
;
        rect1.dy1  =     1
;
        // rect size
        rect1.xlen =     width_out
;
        rect1.ylen =     (i!=num_of_iter-1)*(height_out / num_of_iter -
        height_out / num_of_iter / 2) + (i==num_of_iter-1)*
        (height_out - (height_out - height_out/num_of_iter * (num_of_iter-1))/2)    ;
        // target point
        rect1.xs   =     0
;
        rect1.ys   =     (i!=num_of_iter-1)*(i*(height_out/num_of_iter) +
        height_out / num_of_iter / 2) +
        (i==num_of_iter-1)*(height_out/num_of_iter*(num_of_iter-1)
        + (height_out - height_out/num_of_iter * (num_of_iter-1))/2)                ;

        begining = i*(height_out/num_of_iter);
        end = (i!=num_of_iter-1)*((i+1)*(height_out/num_of_iter) - 1) +
        (i==num_of_iter-1)*(height_out - 1);


        find_boundaries(begining, end, width_out, projection_inv,
        &Ax, &Ay, &Bx, &By, &Cx, &Cy, &Dx, &Dy);

        *(uint8_t*)((uint8_t*)uniform + 4) = (Ay<By)*Ay + (Ay>=By)*By;


        /*Cv0 and Cv1 setup */
```

```
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SLP (&clData0,
R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_SLP (&clData1, R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));

TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_RECT(&clData0, clData0.cur_addr - clData0.top_addr, 0, 0, 0, &rect0));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_RECT(&clData1, clData1.cur_addr - clData1.top_addr, 0, 0, 0, &rect1));

/* wait for all threads to finish */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_SYNCS(&clData0, R_ATMLIB_HELPER_OFFSET_NONE, 1 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_SYNCS(&clData1, R_ATMLIB_HELPER_OFFSET_NONE, 1 ));

/* flush data cache */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_SYNCM(&clData0, R_ATMLIB_HELPER_OFFSET_NONE, 0,0,0,0, 1 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_SYNCM(&clData1, R_ATMLIB_HELPER_OFFSET_NONE, 0,0,0,0, 1 ));

/* wake up DMAC0 */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WUP(&clData0, R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_WUP(&clData1, R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));

/* Parameters for matrix transfer from DDR->Scratchpad */
/* Prepare image parameters */
dma_param_img_input_to_scp.srca_type            = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.srcb_type            = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.dst_type             = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.srca_addr            = (R_ATMLIB_U32) gf_GetPhysAddr(
memories64[0] +(Ay<=By)*Ay*width_in_64_rgb + (By<Ay)*By*width_in_64_rgb);
// image from ddr
dma_param_img_input_to_scp.srcb_addr            = (R_ATMLIB_U32) 0;
dma_param_img_input_to_scp.dst_addr             = (R_ATMLIB_U32) MEM_SCRATCH_INPUT
+width_in_64_rgb;
// address of scrachpad
dma_param_img_input_to_scp.srca_stride          = width_in_64_rgb* sizeof(uint32_t);
dma_param_img_input_to_scp.srcb_stride          = width_in_64_rgb* sizeof(uint32_t);
dma_param_img_input_to_scp.dst_stride           = width_in_64_rgb* sizeof(uint32_t);

/* Prepare DMA parameters */

dma_sub_param_input_to_scp.leng                 = (((Cy>Dy)*((Ay<=By)*
(Cy - Ay + 1) + (Ay>By)*(Cy - By +1)) + (Cy<=Dy)*((Ay<=By)*(Dy - Ay + 1) +
(Ay>By)*(Dy - By +1))) << 16 |
width_in_64_rgb); //Setup width and height
dma_sub_param_input_to_scp.lop
= (R_ATMLIB_U32) R_ATMLIB_HELPER_IDMA_OP_S0;
//Setup operation
```

```
dma_sub_param_input_to_scp.bitcount             = R_ATMLIB_HELPER_DMA_BITCOUNT_DIS;
//No bitcount
dma_sub_param_input_to_scp.src_mode             =

R_ATMLIB_HELPER_DMA_SRC_DATA_MODE_MEM_CONST;
//Data from memory
dma_sub_param_input_to_scp.srca_data            = 0;
dma_sub_param_input_to_scp.srcb_data            = 0;
dma_sub_param_input_to_scp.dst_clip             = 0;
dma_sub_param_input_to_scp.clip_min             = 0;
dma_sub_param_input_to_scp.clip_max             = 0;
dma_sub_param_input_to_scp.srca_addr_mode       =

R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.srcb_addr_mode       =

R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.dst_addr_mode        =

R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode



dma_param_img_input_to_scp.srca_type            = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.srcb_type            = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.dst_type             = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.srca_addr            = (R_ATMLIB_U32)
gf_GetPhysAddr(memories64[2]);    // image from ddr
dma_param_img_input_to_scp.srcb_addr            = (R_ATMLIB_U32) 0;
dma_param_img_input_to_scp.dst_addr             = (R_ATMLIB_U32) MEM_SCRATCH_INPUT;
// address of scrachpad
dma_param_img_input_to_scp.srca_stride          = width_in_64_rgb * sizeof(float);
dma_param_img_input_to_scp.srcb_stride          = width_in_64_rgb * sizeof(float);
dma_param_img_input_to_scp.dst_stride           = width_in_64_rgb * sizeof(float);

/* Prepare DMA parameters */

dma_sub_param_input_to_scp.leng
= ((1) << 16 | width_in_64_rgb); //Setup width and height
dma_sub_param_input_to_scp.lop                  =

(R_ATMLIB_U32) R_ATMLIB_HELPER_IDMA_OP_S0;     //Setup operation
dma_sub_param_input_to_scp.bitcount             =
R_ATMLIB_HELPER_DMA_BITCOUNT_DIS;              //No bitcount
dma_sub_param_input_to_scp.src_mode             =
R_ATMLIB_HELPER_DMA_SRC_DATA_MODE_MEM_CONST;   //Data from memory
dma_sub_param_input_to_scp.srca_data            = 0;
dma_sub_param_input_to_scp.srcb_data            = 0;
dma_sub_param_input_to_scp.dst_clip             = 0;
dma_sub_param_input_to_scp.clip_min             = 0;
dma_sub_param_input_to_scp.clip_max             = 0;
dma_sub_param_input_to_scp.srca_addr_mode       =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
```

```
dma_sub_param_input_to_scp.srcb_addr_mode
= R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.dst_addr_mode
= R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode

/* Parameters for matrix transfer from Scratchpad->DDR */

/* Prepare image parameters */
dma_param_img_output_to_ddr.srca_type           = R_ATMLIB_IMG_8U;
dma_param_img_output_to_ddr.srcb_type           = R_ATMLIB_IMG_8U;
dma_param_img_output_to_ddr.dst_type            = R_ATMLIB_IMG_8U;
dma_param_img_output_to_ddr.srca_addr           = (R_ATMLIB_U32) MEM_SCRATCH_OUTPUT;
dma_param_img_output_to_ddr.srcb_addr           = (R_ATMLIB_U32) 0;
dma_param_img_output_to_ddr.dst_addr            = (R_ATMLIB_U32) gf_GetPhysAddr(
memories64[1] + i*width_out_64_rgb*(height_out/num_of_iter));
dma_param_img_output_to_ddr.srca_stride         = width_out_64_rgb *
sizeof(uint32_t);
dma_param_img_output_to_ddr.srcb_stride         = width_out_64_rgb *
sizeof(uint32_t);
dma_param_img_output_to_ddr.dst_stride          = width_out_64_rgb *
sizeof(uint32_t);

/* Prepare DMA parameters */
dma_sub_param_img_output_to_ddr.leng            =
(((i!=num_of_iter-1)*(height_out/num_of_iter) + (i==num_of_iter-1)*(height_out -
(num_of_iter-1)*(height_out/num_of_iter))) << 16)
| width_out_64_rgb;    //Setup width and height
dma_sub_param_img_output_to_ddr.lop             = (R_ATMLIB_U32)
R_ATMLIB_HELPER_IDMA_OP_S0;     //Setup operation
dma_sub_param_img_output_to_ddr.bitcount
= R_ATMLIB_HELPER_DMA_BITCOUNT_DIS;              //No bitcount
dma_sub_param_img_output_to_ddr.src_mode        =
R_ATMLIB_HELPER_DMA_SRC_DATA_MODE_MEM_CONST;  //Data from memory
dma_sub_param_img_output_to_ddr.srca_data       = 0;
dma_sub_param_img_output_to_ddr.srcb_data       = 0;
dma_sub_param_img_output_to_ddr.dst_clip        = 0;
dma_sub_param_img_output_to_ddr.clip_min        = 0;
dma_sub_param_img_output_to_ddr.clip_max        = 0;
dma_sub_param_img_output_to_ddr.srca_addr_mode
= R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_img_output_to_ddr.srcb_addr_mode
= R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_img_output_to_ddr.dst_addr_mode
= R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode

/*Transfer image DDR->Scratchpad*/
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_SetDataDMACL(&cl_iDMA,
R_ATMLIB_HELPER_OFFSET_NONE,
R_ATMLIB_DMA_BASIC, &dma_sub_param_input_to_scp, &dma_param_img_input_to_scp));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_DMA_SYNCS(&cl_iDMA, cl_iDMA.cur_addr - cl_iDMA.top_addr - 1));

/* Wake up OCV0 and OCV1 */
```

```
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,

        r_atmlib_DMA_WUP(&cl_iDMA, R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_OCV0 |
        RCVDRV_CORE_OCV1));

        /* Wait for CVE0 and OCV1 to finish */
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,

        r_atmlib_DMA_SLP(&cl_iDMA, R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_OCV0 |
        RCVDRV_CORE_OCV1));

        /*Transfer image Scratchpad->DDR*/
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
        r_atmlib_SetDataDMACL(&cl_iDMA, R_ATMLIB_HELPER_OFFSET_NONE,
        R_ATMLIB_DMA_BASIC, &dma_sub_param_img_output_to_ddr,
        &dma_param_img_output_to_ddr));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
        r_atmlib_DMA_SYNCS(&cl_iDMA, cl_iDMA.cur_addr - cl_iDMA.top_addr - 1));

}
/* Add TRAP */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_TRAP(&clData0, clData0.cur_addr - clData0.top_addr, 0 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_OCV_TRAP(&clData1, clData1.cur_addr - clData1.top_addr, 0 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
r_atmlib_DMA_TRAP(&cl_iDMA, R_ATMLIB_HELPER_OFFSET_NONE, 0));


gf_DCacheFlushRange((uintptr_t)cl_iDMA.top_addr,gf_MallocUsableSize(cl_iDMA.top_addr));
gf_DCacheFlushRange((uintptr_t)clData0.top_addr,gf_MallocUsableSize(clData0.top_addr));
gf_DCacheFlushRange((uintptr_t)clData1.top_addr,gf_MallocUsableSize(clData1.top_addr));
gf_DCacheFlushRange((uintptr_t)progMem,          gf_MallocUsableSize(progMem));
gf_DCacheFlushRange((uintptr_t)memories[0],      gf_MallocUsableSize(memories[0]));
gf_DCacheFlushRange((uintptr_t)memories[1],      gf_MallocUsableSize(memories[1]));
gf_DCacheFlushRange((uintptr_t)memories64[0],      gf_MallocUsableSize(memories64[0]));
gf_DCacheFlushRange((uintptr_t)memories64[1],      gf_MallocUsableSize(memories64[1]));
gf_DCacheFlushRange((uintptr_t)memories64[2],      gf_MallocUsableSize(memories64[2]));

/* Execute the IMP task              */
clock_gettime(CLOCK_MONOTONIC, &time_s);
running_OCV0  = 1;
running_OCV1  = 1;
running_DMAC0 = 1;

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
            &(getControl()->rcvdrvctl),
            (RCvUint)gf_GetPhysAddr(clData0.top_addr),
            RCVDRV_CORE_OCV0,
            core_map,
            CallBackFuncCVe0));

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
```

```
                &(getControl()->rcvdrvctl),
                (RCvUint)gf_GetPhysAddr(clData1.top_addr),
                RCVDRV_CORE_OCV1,
                core_map,
                CallBackFuncCVe1));

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
                &getControl()->rcvdrvctl,
                (RCvUint)gf_GetPhysAddr(cl_iDMA.top_addr),
                RCVDRV_CORE_DMAC0,
                core_map,
                CallBackFuncDMAC0));

while(running_OCV0 | running_OCV1 | running_DMAC0)
{
    gf_SuspendTask(1);
}

clock_gettime(CLOCK_MONOTONIC, &time_e);

gf_DCacheInvalidateRange((uintptr_t)memories[0], gf_MallocUsableSize(memories[0]));
gf_DCacheInvalidateRange((uintptr_t)memories[1], gf_MallocUsableSize(memories[1]));
gf_DCacheInvalidateRange((uintptr_t)memories64[0], gf_MallocUsableSize(memories64[0]));
gf_DCacheInvalidateRange((uintptr_t)memories64[1], gf_MallocUsableSize(memories64[1]));
gf_DCacheInvalidateRange((uintptr_t)memories64[2], gf_MallocUsableSize(memories64[2]));

//copy memories64[1]->memories[1]
for(i=0; i<height_out; i++){
    for(j=0; j<width_out_rgb; j++){
        *(uint32_t*)((uint32_t*)memories[1] + i*width_out*3 + j) =
        *(uint32_t*)((uint32_t*)memories64[1] + i*width_out_64_rgb + j);
    }
}

uint8_t* output_image_rgb_pixels = NULL;
output_image_rgb_pixels = (uint8_t *)gf_Memalign(64,

sizeof(uint8_t)*width_out_rgb*height_out);

TEST_ASSERT_NOT_NULL(output_image_rgb_pixels);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out_rgb; j++)
        output_image_rgb_pixels[i*width_out_rgb+j] =
        *(uint32_t*)((uint32_t*)memories[2]+i*width_out_rgb+j);




output_image_ARM = read_ppm("outputARM.ppm");
if(output_image_ARM==NULL) exit(EXIT_FAILURE);
```

```
//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(output_image_ARM, &dataBuffer_ARM, &width_ARM,
&height_ARM, &ppm_ARM, &nChannels_ARM))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels_ARM != 3)
{
    printf("Invalid number of ARM output channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

output_image_ARM_rgb_pixels = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_ARM*
3 *height_ARM);
TEST_ASSERT_NOT_NULL(output_image_ARM_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer_ARM, width_ARM, height_ARM,
output_image_ARM_rgb_pixels);

printf("Provera korektnosti izlaza: ");
if(width_out != width_ARM || height_out != height_ARM){
    printf("DIMENZIJE SLIKE NISU DOBRE\n");
}else{

    int num_of_diff_pixels =
    compare_pixels(output_image_ARM_rgb_pixels, output_image_rgb_pixels,
    width_out_rgb*height_out);
    if(num_of_diff_pixels != 0){
        printf("BROJ RAZLICITIH PIKSELA JE %d\n", num_of_diff_pixels);
    }else{
        printf("IZLAZI SU ISTI\n");
    }
}

uint8_t *pixel_differences = NULL;
pixel_differences = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(pixel_differences);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out; j++){
            if(output_image_rgb_pixels[i*width_out_rgb+3*j]!=
            output_image_ARM_rgb_pixels[i*width_out_rgb+3*j] ||
                output_image_rgb_pixels[i*width_out_rgb+3*j+1]!=
```

```
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+1] ||
                    output_image_rgb_pixels[i*width_out_rgb+3*j+2]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+2]
                ){
                    pixel_differences[i*width_out_rgb+3*j] = 255;
                    pixel_differences[i*width_out_rgb+3*j+1] = 0;
                    pixel_differences[i*width_out_rgb+3*j+2] = 0;
                }
                else{
                    pixel_differences[i*width_out_rgb+3*j] = 0;
                    pixel_differences[i*width_out_rgb+3*j+1] = 0;
                    pixel_differences[i*width_out_rgb+3*j+2] = 0;
                }
            }

    write_ppm("./outputiDMA_Scartchpad.ppm", output_image_rgb_pixels,
    width_out, height_out);
    write_ppm("./outputiDMA_Scratchapadpixel_differences.ppm", pixel_differences,
    width_out, height_out);
    printf("\nTEST_iDMA_Scratchpad_IMAGE_SCALING image [ms]:%f\n\n", (double)
    (time_e.tv_nsec -
    time_s.tv_nsec) / 1000000.0 + (double)(time_e.tv_sec - time_s.tv_sec) * 1000.0);
    r_atmlib_ReleaseCL(&clData0);
    r_atmlib_ReleaseCL(&clData1);
    gf_Free(progMem);
    gf_Free(memories[0]);
    gf_Free(memories[1]);
    gf_Free(memories64[0]);
    gf_Free(memories64[1]);
    gf_Free(memories64[2]);
    gf_Free(output_image_rgb_pixels);
    gf_Free(input_image_rgb_pixels);
    gf_Free(output_image_ARM_rgb_pixels);
    gf_Free(pixel_differences);
    free_resources();


}
```

A.12 Кôд за нити оптимизације *Scratchpad*

```
#kod za najblizeg suseda

#include <stdint.h>


const int iter_num = 0;
const int height_in = 0;
const int height_out = 0;
const int num_of_iter = 0;
const int begining_of_input = 0;
```

```
#pragma section bss lwm
float projection_invers[9];

#pragma entry main master
void main(void)
{
    __setSrcImageID(2);

    int k;
    for(k=0; k<9; k++)
        projection_invers[k] = __getSrcf(k - __getX(), - __getY());


    __setSrcImageID(0);
    __setDstImageID(1);


    unsigned x,y,t; //homogene koordinate piksela izlazne slike
    float pinv_tx, pinv_ty, pinv_t;
    //=pinv*(x,y,t)^T tj. homogene koordinate ulazne slike -> (x,y,t)^T
    unsigned pinv_x, pinv_y; // pinv_x = pinv_tx/pinv_t, slicno za pinv_y
    float dx, dy; // greske pri zaokruzivanju za pinv_x, pinv_y

    unsigned begining_of_output = iter_num*(unsigned)((float)height_out/num_of_iter);

    //prave koordinate
    x = __getX()/3; //kolona
    y = __getY() + begining_of_output; //vrsta
    t = 1; //prosirenje na 3d

    //homogene koordinate tacke koja se slika u (x, y, 1) sa P
    pinv_tx = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
    pinv_ty = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
    pinv_t  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

   //prave koordinate
    pinv_x = pinv_tx / pinv_t;
    pinv_y = pinv_ty / pinv_t;

    //koliko smo omasili pri zaokruzivanju
    dx = pinv_tx / pinv_t - pinv_x;
    dy = pinv_ty / pinv_t - pinv_y;

    if(dx>0.5){
        pinv_x++;
    }

    if(dy>0.5){
        pinv_y++;
    }

    pinv_y -=begining_of_input;
```

```
    for(k=0; k<3; k++){ //za rgb
        __setDst(k, 0, __getSrc(3*pinv_x + k - __getX(), pinv_y  - __getY()));
    }



    __trap();

}

#kod za bilinearnu interpolaciju
#include <stdint.h>


const int iter_num = 0;
const int height_in = 0;
const int height_out = 0;
const int num_of_iter = 0;
const int begining_of_input = 0;

float lerp(float s, float e, float t){return s+(e-s)*t;}

float blerp(float c00, float c10, float c01, float c11, float tx, float ty){
    return lerp(lerp(c00, c10, tx), lerp(c01, c11, tx), ty);
}

#pragma section bss lwm
float projection_invers[9];

#pragma entry main master
void main(void)
{
    __setSrcImageID(2);

    int k;
    for(k=0; k<9; k++)
        projection_invers[k] = __getSrcf(k - __getX(),  - __getY());


    __setSrcImageID(0);
    __setDstImageID(1);


    unsigned x,y,t; //homogene koordinate piksela izlazne slike
    float pinv_tx, pinv_ty, pinv_t;
    //=pinv*(x,y,t)^T tj. homogene koordinate ulazne slike -> (x,y,t)^T
    unsigned pinv_x, pinv_y; // pinv_x = pinv_tx/pinv_t, slicno za pinv_y
    float dx, dy; // greske pri zaokruzivanju za pinv_x, pinv_y
    unsigned c00, c01, c10, c11; //okolni pikseli za bilineranu interpolaciju

    unsigned begining_of_output = iter_num*(unsigned)((float)height_out/num_of_iter);
```

```
    //prave koordinate
    x = __getX()/3; //kolona
    y = __getY() + begining_of_output; //vrsta
    t = 1; //prosirenje na 3d

    //homogene koordinate tacke koja se slika u (x, y, 1) sa P
    pinv_tx = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
    pinv_ty = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
    pinv_t  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

   //prave koordinate
    pinv_x = pinv_tx / pinv_t;
    pinv_y = pinv_ty / pinv_t;

    //koliko smo omasili pri zaokruzivanju
    dx = pinv_tx / pinv_t - pinv_x;
    dy = pinv_ty / pinv_t - pinv_y;

    pinv_y -=begining_of_input;

    //bilinearnom interpolacijom aproksimiramo vrednost
    for(k=0; k<3; k++){ //za rgb
        c00 = __getSrc(3*pinv_x + k - __getX(), pinv_y  - __getY());
        c10 = __getSrc(3*(pinv_x+1) + k - __getX(), pinv_y - __getY());
        c01 = __getSrc(3*pinv_x + k - __getX(), pinv_y + 1 - __getY());
        c11 = __getSrc(3*(pinv_x+1) + k - __getX(), pinv_y + 1 - __getY());

        float rez = lerp(lerp(c00, c10, dx), lerp(c01, c11, dx), dy);
        __setDst(k, 0, (uint32_t)rez);

    }



    __trap();

}
```

A.13 Оптимизација $TGDMAC + LWM$

```
TEST(C_Group, TEST_TGDMAc_LWM_PROJECTION_TRANSFORMATION){
    R_ATMLIB_CLData clData0, clData1;
    R_ATMLIB_CLData cl_iDMA;
    /* Input matrix (DDR->Scratchpad) copy parameters */
    R_ATMLIB_ImageParam      dma_param_img_input_to_scp;
    R_ATMLIB_DMASubsetParam dma_sub_param_input_to_scp;
    /* Output matrix (Scratchpad->DDR) copy parameters */
    R_ATMLIB_ImageParam      dma_param_img_output_to_ddr;
    R_ATMLIB_DMASubsetParam dma_sub_param_img_output_to_ddr;

    R_ATMLIB_OCVRectParam rect0, rect1;
```

```
uint32_t data[10];
uint8_t  program[CVE_size_of_cve_projection_lwm_text]    =
{CVE_cve_projection_lwm_text};
uint8_t  uniform[CVE_size_of_cve_projection_lwm_uniform] =
{CVE_cve_projection_lwm_uniform};
uint32_t *progMem      =  NULL;
uint32_t  *memories[2] = {NULL, NULL};
uint32_t *memories64[3] = {NULL, NULL, NULL};
struct timespec time_s, time_e;

void   * dataBuffer;
uint8_t  ppm;
size_t   nChannels;
size_t   width_in, height_in, width_in_rgb, width_in_64_rgb, width_in_64;
size_t   width_out, height_out, width_out_rgb, width_out_64, width_out_64_rgb;
unsigned char * input_img  = NULL;

void   * dataBuffer_ARM;
uint8_t  ppm_ARM;
size_t   nChannels_ARM;
size_t   width_ARM, height_ARM;
unsigned char * output_image_ARM  = NULL;
uint8_t * output_image_ARM_rgb_pixels = NULL;
int i,j;


input_img = read_ppm("parking.ppm");
if(input_img==NULL) return;

/* Read PPM image header */
if (!getPPMPGMImageInfo(input_img, &dataBuffer, &width_in, &height_in, &ppm,
&nChannels))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    return;
}

if (nChannels != 3)
{
    printf("Invalid number of input channels\n");
    printf("ABORT\n");
    return;
}

width_in_rgb   = width_in * 3;
width_out = 300;
height_out = 300;

if(width_in % 64 != 0){
    width_in_64 = width_in + 64 - width_in%64;
    width_in_64_rgb = width_in_64*3;
}
```

```
    else{
        width_in_64 = width_in;
        width_in_64_rgb = width_in_rgb;
    }

    if(width_out % 64 != 0){
        width_out_64 = width_out + 64 - width_out%64;
        width_out_64_rgb = width_out_64*3;
    }
    else{
        width_out_64 = width_out;
        width_out_64_rgb = width_out * 3;
    }

    unsigned input_points[4][2];
    unsigned output_points[4][2];


    input_points[0][0] = 280; input_points[0][1] = 125;
    input_points[1][0] = 270; input_points[1][1] = 410;
    input_points[2][0] = 580; input_points[2][1] = 355;
    input_points[3][0] = 520; input_points[3][1] = 40;

    output_points[0][0] = 0; output_points[0][1] = 0;
    output_points[1][0] = 0; output_points[1][1] = height_out;
    output_points[2][0] = width_out; output_points[2][1] = height_out;
    output_points[3][0] = width_out; output_points[3][1] = 0;


    /*pronalazenje odgovarajuceg projektivnog preslikavanja*/
    int k;
    float projection[3][3];
    int result = find_projection(input_points, output_points, projection);

    if(result == -1){
        printf("GRESKA: Tacke projektinog preslikavanja ne ispunjavaju uslov da ni jedna
        trojka tacaka nije kolinerana\n");
        return;
    }else if(result == -2){
        printf("GRESKA: Doslo je do greske pri racunu i ne poklapaju se ulazne i
        izlazne tacke u projektivnom preslikavanju\n");
        return;
    }else{
        printf("\nIzracunato je projektivno preslikavanje:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++)
                printf("%lf ", projection[i][j]);
            printf("\n");
        }


    }
```

```
//pronalazenje inverza nadjenog projektivnog preslikavanja
float projection_inv[3][3];
result = invers_3(projection, projection_inv);
if(result == -1){
     printf("GRESKA: Determinanta projektivnog preslikavanja je 0\n");
     return;
}

printf("\nInverz projektivnog preslikavanja:\n");
for(i=0; i<3; i++){
       for(j=0; j<3; j++){
            printf("%lf ", projection_inv[i][j]);
       }
    printf("\n");
}

uint8_t* input_image_rgb_pixels = NULL;
input_image_rgb_pixels = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)*width_in_rgb
*height_in);
TEST_ASSERT_NOT_NULL(input_image_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer, width_in, height_in, input_image_rgb_pixels);


progMem      = gf_Memalign(64, CVE_size_of_cve_projection_lwm_text);
memories[0] = (uint32_t *)gf_Memalign(64, width_in_rgb * height_in *
sizeof(uint32_t));
memories[1] = (uint32_t *)gf_Memalign(64, width_out_rgb * height_out *
sizeof(uint32_t));
memories64[0] = (uint32_t *)gf_Memalign(64, width_in_64_rgb * height_in *
sizeof(uint32_t));
memories64[1] = (uint32_t *)gf_Memalign(64, width_out_64_rgb * height_out *
sizeof(uint32_t));
memories64[2] = (float *)gf_Memalign(64, 64*sizeof(float));



TEST_ASSERT_NOT_NULL(progMem);
TEST_ASSERT_NOT_NULL(memories[0]);
TEST_ASSERT_NOT_NULL(memories[1]);
TEST_ASSERT_NOT_NULL(memories64[0]);
TEST_ASSERT_NOT_NULL(memories64[1]);
TEST_ASSERT_NOT_NULL(memories64[2]);

memset(memories[0], 0, width_in_rgb * height_in * sizeof(uint32_t));
memset(memories[1], 0, width_out_rgb * height_out * sizeof(uint32_t));
memset(memories64[0], 0, width_in_64_rgb * height_in * sizeof(uint32_t));
memset(memories64[1], 0, width_out_64_rgb * height_out * sizeof(uint32_t));
memset(memories64[2], 0, 64 * sizeof(float));

for(i=0; i<height_in; i++){
```

```
        for(j=0; j<width_in_rgb; j++){
            *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j) =
            input_image_rgb_pixels[i*width_in_rgb+j];
        }
    }

    for(i=0; i<height_in; i++){
        for(j=0; j<width_in_rgb; j++){
            *(uint32_t*)((uint32_t*)memories64[0] + i*width_in_64_rgb + j) =
            *(uint32_t*)((uint32_t*)memories[0]+i*width_in_rgb+j);
        }
    }


    memcpy(memories64[2], projection_inv, 9*sizeof(float));

    /* Copy program to allocated memory for the ASM code */
    memcpy(progMem, program, sizeof(program));

    /* Prepare CL data structure for iDMA*/
    cl_iDMA.cltype = R_ATMLIB_CLTYPE_DMA;
    cl_iDMA.align  = 64;
    cl_iDMA.size   = R_CL_SIZE;
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&cl_iDMA));

    /* Prepare CL data structure for CVe0*/
    clData0.cltype = R_ATMLIB_CLTYPE_OCV;
    clData0.align  = 64;
    clData0.size   = R_CL_SIZE;
    /* Prepare CL data structure for CVe1*/
    clData1.cltype = R_ATMLIB_CLTYPE_OCV;
    clData1.align  = 64;
    clData1.size   = R_CL_SIZE;
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData0));
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_CreateCL(&clData1));

    data[0] = gf_GetPhysAddr(progMem);
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
    clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_VIBAR, 1, data));
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
    clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_VIBAR, 1, data));

    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
    clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_UNR0,
    CVE_size_of_cve_projection_lwm_uniform >> 2, (uint32_t*)uniform));
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
    clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_UNR0,
    CVE_size_of_cve_projection_lwm_uniform >> 2, (uint32_t*)uniform));

    data[0]   = 0x00FE0001; /* 1 master and 7 slave */
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
    clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
```

```
clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_TGSEN1, 1, data));

data[0]   = MEM_SCRATCH_INPUT; /* phy address */
data[1]   = 64 * sizeof(float); /* stride */
data[2]   = 0x00070107; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_IMGBAR2, 3, data));

data[0]   = MEM_SCRATCH_INPUT + width_out_64_rgb; /* phy address */
data[1]   = width_in_64_rgb * sizeof(uint32_t); /* stride */
data[2]   = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_IMGBAR0, 3, data));


data[0]   = MEM_SCRATCH_OUTPUT; /* phy address */
data[1]   = width_out_64_rgb * sizeof(uint32_t); /* stride */
data[2]   = 0x00050105; /* image control register */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_IMGBAR1, 3, data));


unsigned output_size_scp = 896*1024*4;
unsigned num_of_iter = height_out*width_out_64_rgb/(output_size_scp - width_out_64_rgb)
+(height_out*width_out_64_rgb%(output_size_scp -
width_out_64_rgb) != 0);

unsigned begining, end, Ax, Ay, Bx, By, Cx, Cy, Dx, Dy;
*(uint8_t*)((uint8_t*)uniform + 1) = height_in;
*(uint8_t*)((uint8_t*)uniform + 2) = height_out;
*(uint8_t*)((uint8_t*)uniform + 3) = num_of_iter;
*(uint8_t*)((uint8_t*)uniform + 5) = width_out;
*(uint8_t*)((uint8_t*)uniform + 6) = width_in;



for(i=0; i<num_of_iter; i++){
    *((uint8_t*)uniform) = i;

    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
    clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_UNR0,
    CVE_size_of_cve_projection_lwm_uniform >> 2, (uint32_t*)uniform));
    TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
    clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_UNR0,
    CVE_size_of_cve_projection_lwm_uniform >> 2, (uint32_t*)uniform));

    data[0]   = CVE_entry_master_cve_projection_lwm__f_master;
```

```
        data[1]    = 0;
        data[2]    = CVE_entry_slave_cve_projection_lwm__f_slave;
        data[3]    = 0;
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData0,
        clData0.cur_addr - clData0.top_addr, R_IMPX5P_REG_OCV_VPCSAR, 4, data));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WPR(&clData1,
        clData1.cur_addr - clData1.top_addr, R_IMPX5P_REG_OCV_VPCSAR, 4, data));


        //RECT for CVe0

        rect0.dx1  =      0
;
        rect0.dy2  =      0
;
        // vertical step
        rect0.dx2  =      0
;
        rect0.dy1  =      1
;
        // rect size
        rect0.xlen =      1
;
        rect0.ylen =      (i!=num_of_iter-1)*(height_out / num_of_iter / 2)
        + (i==num_of_iter-1)*((height_out - height_out/num_of_iter * (num_of_iter-1))/2)
;
        // target point
        rect0.xs   =      0
;
        rect0.ys   =      i*(height_out/num_of_iter)
;


        //RECT for CVe1

        rect1.dx1  =      0
;
        rect1.dy2  =      0
;
        // vertical step
        rect1.dx2  =      0
;
        rect1.dy1  =      1
;
        // rect size
        rect1.xlen =      1
;
        rect1.ylen =      (i!=num_of_iter-1)*(height_out / num_of_iter -
        height_out / num_of_iter / 2) + (i==num_of_iter-1)*(height_out -
        (height_out - height_out/num_of_iter * (num_of_iter-1))/2)    ;
        // target point
        rect1.xs   =      0
;
        rect1.ys   =      (i!=num_of_iter-1)*(i*(height_out/num_of_iter) +
```

```
        height_out / num_of_iter / 2) +(i==num_of_iter-1)*(height_out/num_of_iter*
        (num_of_iter-1) + (height_out - height_out/num_of_iter * (num_of_iter-1))/2)
;

        begining = i*(height_out/num_of_iter);
        end = (i!=num_of_iter-1)*((i+1)*(height_out/num_of_iter) - 1) +
        (i==num_of_iter-1)*(height_out - 1);


        find_boundaries(begining, end, width_out, projection_inv, &Ax,
        &Ay, &Bx, &By, &Cx, &Cy, &Dx, &Dy);

        *(uint8_t*)((uint8_t*)uniform + 4) = (Ay<By)*Ay + (Ay>=By)*By;

        //pokusaj jedan, prebacujemo sve od 0 do width_out i od min(Ay, By)
        //i do max(Cy, Dy)
        /*Cv0 and Cv1 setup */

        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SLP (&clData0,
        R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SLP (&clData1,
        R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));

        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_RECT(&clData0,
        clData0.cur_addr - clData0.top_addr, 0, 0, 0, &rect0));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_RECT(&clData1,
        clData1.cur_addr - clData1.top_addr, 0, 0, 0, &rect1));

        /* wait for all threads to finish */
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SYNCS(&clData0,
        R_ATMLIB_HELPER_OFFSET_NONE, 1 ));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SYNCS(&clData1,
        R_ATMLIB_HELPER_OFFSET_NONE, 1 ));

        /* flush data cache */
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SYNCM(&clData0,
        R_ATMLIB_HELPER_OFFSET_NONE, 0,0,0,0, 1 ));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_SYNCM(&clData1,
        R_ATMLIB_HELPER_OFFSET_NONE, 0,0,0,0, 1 ));

        /* wake up DMAC0 */
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WUP(&clData0,
        R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_WUP(&clData1,
        R_ATMLIB_HELPER_OFFSET_NONE, RCVDRV_CORE_DMAC0));

        /* Parameters for matrix transfer from DDR->Scratchpad */
        /* Prepare image parameters */
        dma_param_img_input_to_scp.srca_type             = R_ATMLIB_IMG_8U;
        dma_param_img_input_to_scp.srcb_type             = R_ATMLIB_IMG_8U;
        dma_param_img_input_to_scp.dst_type              = R_ATMLIB_IMG_8U;
        dma_param_img_input_to_scp.srca_addr             = (R_ATMLIB_U32)
        gf_GetPhysAddr(memories64[0] + (Ay<=By)*Ay*width_in_64_rgb +
```

```
(By<Ay)*By*width_in_64_rgb);    // image from ddr
dma_param_img_input_to_scp.srcb_addr            = (R_ATMLIB_U32) 0;
dma_param_img_input_to_scp.dst_addr             = (R_ATMLIB_U32)
MEM_SCRATCH_INPUT + width_in_64_rgb;            // address of scrachpad
dma_param_img_input_to_scp.srca_stride          = width_in_64_rgb* sizeof(uint32_t);
dma_param_img_input_to_scp.srcb_stride          = width_in_64_rgb* sizeof(uint32_t);
dma_param_img_input_to_scp.dst_stride           = width_in_64_rgb* sizeof(uint32_t);

/* Prepare DMA parameters */

dma_sub_param_input_to_scp.leng                 = (((Cy>Dy)*((Ay<=By)*(Cy - Ay + 1)
+(Ay>By)*(Cy - By +1))+(Cy<=Dy)*((Ay<=By)*(Dy - Ay + 1) + (Ay>By)*(Dy - By +1)))
<< 16 | width_in_64_rgb); //Setup width and height
dma_sub_param_input_to_scp.lop                  =
(R_ATMLIB_U32) R_ATMLIB_HELPER_IDMA_OP_S0;     //Setup operation
dma_sub_param_input_to_scp.bitcount             =
R_ATMLIB_HELPER_DMA_BITCOUNT_DIS;
//No bitcount
dma_sub_param_input_to_scp.src_mode             =
R_ATMLIB_HELPER_DMA_SRC_DATA_MODE_MEM_CONST;   //Data from memory
dma_sub_param_input_to_scp.srca_data            = 0;
dma_sub_param_input_to_scp.srcb_data            = 0;
dma_sub_param_input_to_scp.dst_clip             = 0;
dma_sub_param_input_to_scp.clip_min             = 0;
dma_sub_param_input_to_scp.clip_max             = 0;
dma_sub_param_input_to_scp.srca_addr_mode       =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.srcb_addr_mode       =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.dst_addr_mode        =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode



dma_param_img_input_to_scp.srca_type            = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.srcb_type            = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.dst_type             = R_ATMLIB_IMG_8U;
dma_param_img_input_to_scp.srca_addr            = (R_ATMLIB_U32)
gf_GetPhysAddr(memories64[2]);    // image from ddr
dma_param_img_input_to_scp.srcb_addr            = (R_ATMLIB_U32) 0;
dma_param_img_input_to_scp.dst_addr             = (R_ATMLIB_U32)
MEM_SCRATCH_INPUT;              // address of scrachpad
dma_param_img_input_to_scp.srca_stride          = width_in_64_rgb*
sizeof(float);
dma_param_img_input_to_scp.srcb_stride          = width_in_64_rgb*
sizeof(float);
dma_param_img_input_to_scp.dst_stride           = width_in_64_rgb*
sizeof(float);

/* Prepare DMA parameters */

dma_sub_param_input_to_scp.leng                 = ((1) << 16 | width_in_64_rgb);
//Setup width and height
```

```
dma_sub_param_input_to_scp.lop                  = (R_ATMLIB_U32)
R_ATMLIB_HELPER_IDMA_OP_S0;     //Setup operation
dma_sub_param_input_to_scp.bitcount             = R_ATMLIB_HELPER_DMA_BITCOUNT_DIS;
//No bitcount
dma_sub_param_input_to_scp.src_mode             =
R_ATMLIB_HELPER_DMA_SRC_DATA_MODE_MEM_CONST;  //Data from memory
dma_sub_param_input_to_scp.srca_data            = 0;
dma_sub_param_input_to_scp.srcb_data            = 0;
dma_sub_param_input_to_scp.dst_clip             = 0;
dma_sub_param_input_to_scp.clip_min             = 0;
dma_sub_param_input_to_scp.clip_max             = 0;
dma_sub_param_input_to_scp.srca_addr_mode       =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.srcb_addr_mode       =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
dma_sub_param_input_to_scp.dst_addr_mode        =
R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode

/* Parameters for matrix transfer from Scratchpad->DDR */

/* Prepare image parameters */
dma_param_img_output_to_ddr.srca_type           = R_ATMLIB_IMG_8U;
dma_param_img_output_to_ddr.srcb_type           = R_ATMLIB_IMG_8U;
dma_param_img_output_to_ddr.dst_type            = R_ATMLIB_IMG_8U;
dma_param_img_output_to_ddr.srca_addr           = (R_ATMLIB_U32)
MEM_SCRATCH_OUTPUT;
dma_param_img_output_to_ddr.srcb_addr           = (R_ATMLIB_U32) 0;
dma_param_img_output_to_ddr.dst_addr            = (R_ATMLIB_U32)
gf_GetPhysAddr(memories64[1] + i*width_out_64_rgb*(height_out/num_of_iter));
dma_param_img_output_to_ddr.srca_stride         = width_out_64_rgb *
sizeof(uint32_t);
dma_param_img_output_to_ddr.srcb_stride         = width_out_64_rgb *
sizeof(uint32_t);
dma_param_img_output_to_ddr.dst_stride          = width_out_64_rgb *
sizeof(uint32_t);

/* Prepare DMA parameters */
dma_sub_param_img_output_to_ddr.leng            = (((i!=num_of_iter-1)*
(height_out/num_of_iter) + (i==num_of_iter-1)*(height_out -
(num_of_iter-1)*(height_out/num_of_iter))) << 16) | width_out_64_rgb;
//Setup width and height
dma_sub_param_img_output_to_ddr.lop             = (R_ATMLIB_U32)
R_ATMLIB_HELPER_IDMA_OP_S0;     //Setup operation
dma_sub_param_img_output_to_ddr.bitcount        =
R_ATMLIB_HELPER_DMA_BITCOUNT_DIS;              //No bitcount
dma_sub_param_img_output_to_ddr.src_mode        =
R_ATMLIB_HELPER_DMA_SRC_DATA_MODE_MEM_CONST;  //Data from memory
dma_sub_param_img_output_to_ddr.srca_data       = 0;
dma_sub_param_img_output_to_ddr.srcb_data       = 0;
dma_sub_param_img_output_to_ddr.dst_clip        = 0;
dma_sub_param_img_output_to_ddr.clip_min        = 0;
dma_sub_param_img_output_to_ddr.clip_max        = 0;
dma_sub_param_img_output_to_ddr.srca_addr_mode =
```

```
        R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
        dma_sub_param_img_output_to_ddr.srcb_addr_mode =
        R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode
        dma_sub_param_img_output_to_ddr.dst_addr_mode  =
        R_ATMLIB_HELPER_DMA_ADDR_MODE_LINEAR;          //Linear address mode

        /*Transfer image DDR->Scratchpad*/
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_SetDataDMACL(&cl_iDMA,
        R_ATMLIB_HELPER_OFFSET_NONE, R_ATMLIB_DMA_BASIC,
        &dma_sub_param_input_to_scp, &dma_param_img_input_to_scp));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_DMA_SYNCS(&cl_iDMA,
        cl_iDMA.cur_addr - cl_iDMA.top_addr - 1));

        /* Wake up OCV0 and OCV1 */
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
        r_atmlib_DMA_WUP(&cl_iDMA, R_ATMLIB_HELPER_OFFSET_NONE,
        RCVDRV_CORE_OCV0 | RCVDRV_CORE_OCV1));

        /* Wait for CVE0 and OCV1 to finish */
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK,
        r_atmlib_DMA_SLP(&cl_iDMA, R_ATMLIB_HELPER_OFFSET_NONE,
        RCVDRV_CORE_OCV0 | RCVDRV_CORE_OCV1));

        /*Transfer image Scratchpad->DDR*/
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_SetDataDMACL(&cl_iDMA,
        R_ATMLIB_HELPER_OFFSET_NONE, R_ATMLIB_DMA_BASIC,
        &dma_sub_param_img_output_to_ddr, &dma_param_img_output_to_ddr));
        TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_DMA_SYNCS(&cl_iDMA,
        cl_iDMA.cur_addr - cl_iDMA.top_addr - 1));

}
/* Add TRAP */
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_TRAP(&clData0,
clData0.cur_addr - clData0.top_addr, 0 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_OCV_TRAP(&clData1,
clData1.cur_addr - clData1.top_addr, 0 ));
TEST_ASSERT_EQUAL_INT(R_ATMLIB_E_OK, r_atmlib_DMA_TRAP(&cl_iDMA,
R_ATMLIB_HELPER_OFFSET_NONE, 0));


gf_DCacheFlushRange((uintptr_t)cl_iDMA.top_addr,
gf_MallocUsableSize(cl_iDMA.top_addr));
gf_DCacheFlushRange((uintptr_t)clData0.top_addr,
gf_MallocUsableSize(clData0.top_addr));
gf_DCacheFlushRange((uintptr_t)clData1.top_addr,
gf_MallocUsableSize(clData1.top_addr));
gf_DCacheFlushRange((uintptr_t)progMem,
gf_MallocUsableSize(progMem));
gf_DCacheFlushRange((uintptr_t)memories[0],
gf_MallocUsableSize(memories[0]));
gf_DCacheFlushRange((uintptr_t)memories[1],
gf_MallocUsableSize(memories[1]));
gf_DCacheFlushRange((uintptr_t)memories64[0],
```

```
gf_MallocUsableSize(memories64[0]));
gf_DCacheFlushRange((uintptr_t)memories64[1],
gf_MallocUsableSize(memories64[1]));
gf_DCacheFlushRange((uintptr_t)memories64[2],
gf_MallocUsableSize(memories64[2]));

/* Execute the IMP task                */
clock_gettime(CLOCK_MONOTONIC, &time_s);
running_OCV0  = 1;
running_OCV1  = 1;
running_DMAC0 = 1;

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
              &(getControl()->rcvdrvctl),
              (RCvUint)gf_GetPhysAddr(clData0.top_addr),
              RCVDRV_CORE_OCV0,
              core_map,
              CallBackFuncCVe0));

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
              &(getControl()->rcvdrvctl),
              (RCvUint)gf_GetPhysAddr(clData1.top_addr),
              RCVDRV_CORE_OCV1,
              core_map,
              CallBackFuncCVe1));

TEST_ASSERT_EQUAL_INT(RCV_EC_OK_DRV, rcv_impdrv_ExecuteCB(
              &getControl()->rcvdrvctl,
              (RCvUint)gf_GetPhysAddr(cl_iDMA.top_addr),
              RCVDRV_CORE_DMAC0,
              core_map,
              CallBackFuncDMAC0));

while(running_OCV0 | running_OCV1 | running_DMAC0)
{
    gf_SuspendTask(1);
}

clock_gettime(CLOCK_MONOTONIC, &time_e);

gf_DCacheInvalidateRange((uintptr_t)memories[0],
gf_MallocUsableSize(memories[0]));
gf_DCacheInvalidateRange((uintptr_t)memories[1],
gf_MallocUsableSize(memories[1]));
gf_DCacheInvalidateRange((uintptr_t)memories64[0],
gf_MallocUsableSize(memories64[0]));
gf_DCacheInvalidateRange((uintptr_t)memories64[1],
gf_MallocUsableSize(memories64[1]));
gf_DCacheInvalidateRange((uintptr_t)memories64[2],
gf_MallocUsableSize(memories64[2]));

//copy memories64[1]->memories[1]
for(i=0; i<height_out; i++){
```

```
    for(j=0; j<width_out_rgb; j++){
        *(uint32_t*)((uint32_t*)memories[1] + i*width_out*3 + j) =
        *(uint32_t*)((uint32_t*)memories64[1] + i*width_out_64_rgb + j);
    }
}

uint8_t* output_image_rgb_pixels = NULL;
output_image_rgb_pixels = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(output_image_rgb_pixels);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out_rgb; j++)
        output_image_rgb_pixels[i*width_out_rgb+j] =
        *(uint32_t*)((uint32_t*)memories[2]+i*width_out_rgb+j);




output_image_ARM = read_ppm("outputARM.ppm");
if(output_image_ARM==NULL) exit(EXIT_FAILURE);

//dataBuffer ce pokazivati na neki deo input_img
//tj. cuvace samo adresu
if (!getPPMPGMImageInfo(output_image_ARM, &dataBuffer_ARM,
&width_ARM, &height_ARM, &ppm_ARM, &nChannels_ARM))
{
    printf("Failed to decode ppm image header\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}


if (nChannels_ARM != 3)
{
    printf("Invalid number of ARM output channels\n");
    printf("ABORT\n");
    exit(EXIT_FAILURE);
}

output_image_ARM_rgb_pixels = (uint8_t *)gf_Memalign(64,
sizeof(uint8_t)*width_ARM* 3 *height_ARM);
TEST_ASSERT_NOT_NULL(output_image_ARM_rgb_pixels);


seperatePPMChannelRGB8(dataBuffer_ARM, width_ARM, height_ARM,
output_image_ARM_rgb_pixels);

printf("Provera korektnosti izlaza: ");
if(width_out != width_ARM || height_out != height_ARM){
    printf("DIMENZIJE SLIKE NISU DOBRE\n");
}else{
```

```
    int num_of_diff_pixels = compare_pixels(output_image_ARM_rgb_pixels,
    output_image_rgb_pixels, width_out_rgb*height_out);
    if(num_of_diff_pixels != 0){
        printf("BROJ RAZLICITIH PIKSELA JE %d\n", num_of_diff_pixels);
    }else{
        printf("IZLAZI SU ISTI\n");
    }
}

uint8_t *pixel_differences = NULL;
pixel_differences = (uint8_t *)gf_Memalign(64, sizeof(uint8_t)*
width_out_rgb*height_out);
TEST_ASSERT_NOT_NULL(pixel_differences);


for(i=0; i<height_out; i++)
    for(j=0; j<width_out; j++){
            if(output_image_rgb_pixels[i*width_out_rgb+3*j]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j] ||
                output_image_rgb_pixels[i*width_out_rgb+3*j+1]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+1] ||
                output_image_rgb_pixels[i*width_out_rgb+3*j+2]!=
                output_image_ARM_rgb_pixels[i*width_out_rgb+3*j+2]
            ){
                pixel_differences[i*width_out_rgb+3*j] = 255;
                pixel_differences[i*width_out_rgb+3*j+1] = 0;
                pixel_differences[i*width_out_rgb+3*j+2] = 0;
            }
            else{
                pixel_differences[i*width_out_rgb+3*j] = 0;
                pixel_differences[i*width_out_rgb+3*j+1] = 0;
                pixel_differences[i*width_out_rgb+3*j+2] = 0;
            }
    }

write_ppm("./outputTGDMAc_LWM.ppm", output_image_rgb_pixels,
width_out, height_out);
write_ppm("./outputTGDMAc_LWMpixel_differences.ppm", pixel_differences,
width_out, height_out);
printf("\nTEST_TDGMAc_LWM_PROJECTION_TRANSFORMATION image [ms]:%f\n\n",
(double)(time_e.tv_nsec - time_s.tv_nsec) / 1000000.0 + (double)(time_e.tv_sec -
time_s.tv_sec) * 1000.0);
r_atmlib_ReleaseCL(&clData0);
r_atmlib_ReleaseCL(&clData1);
gf_Free(progMem);
gf_Free(memories[0]);
gf_Free(memories[1]);
gf_Free(memories64[0]);
gf_Free(memories64[1]);
gf_Free(memories64[2]);
gf_Free(output_image_rgb_pixels);
gf_Free(input_image_rgb_pixels);
```

```
    gf_Free(output_image_ARM_rgb_pixels);
    gf_Free(pixel_differences);
    free_resources();
}
```

A.14 Кôд за нити оптимизације $TGDMAC + LWM$

```
// kod za bilinearnu interpolaciju
#include <stdint.h>
#define MAX_WIDTH_OUT 750

#define MAX_WIDTH_IN 3000

const int iter_num = 0;
const int height_in = 0;
const int height_out = 0;
const int num_of_iter = 0;
const int begining_of_input = 0;
const int width_out = 0;
const int width_in = 0;

float lerp(float s, float e, float t){return s+(e-s)*t;}

float blerp(float c00, float c10, float c01, float c11, float tx, float ty){
    return lerp(lerp(c00, c10, tx), lerp(c01, c11, tx), ty);
}


#pragma section bss lwm_comm
uint32_t input_data[MAX_WIDTH_IN];
uint32_t output_data[MAX_WIDTH_OUT];

void dma_copy_spad_lwm(int x, int y, int lwm_base, int width_in);
void dma_copy_lwm_spad(int y, int lwm_base, int stride);

#pragma section bss lwm
float projection_invers[9];

void interpolate(int thread_num){


    int k, j;


    unsigned begining_of_output = iter_num*(unsigned)((float)height_out/num_of_iter);

    __setSrcImageID(0);
    __setDstImageID(1);

    int duzina = width_out/8;
```

```
    unsigned x, y, t;
    float dx, dy, pxt, pyt, pt;
    unsigned px, py;
    for(k=thread_num*duzina; k<(thread_num+1)*duzina; k++){
        x = k;
        y = __getY() + begining_of_output;
        t = 1;

        pxt = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
        pyt = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
        pt  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

        px = pxt/pt;
        py = pyt/pt;

        dx =  pxt/pt - px;
        dy =  pyt/pt - py;
        //input_data za svaki pisekl k sadrzi njegovih okolnih piksela
        //Ck00r Ck00g Ck00b Ck10r Ck10g Ck10b Ck01r Ck01g Ck01b Ck11r Ck11g Ck11b
        for(j=0; j<3; j++){
            unsigned c00 = input_data[4*3*k + j];
            unsigned c10 = input_data[4*3*k + 3 + j];
            unsigned c01 = input_data[4*3*k + 6 + j];
            unsigned c11 = input_data[4*3*k + 9 + j];

            float rez = lerp(lerp(c00, c10, dx), lerp(c01, c11, dx), dy);
            output_data[3*k + j] = (uint32_t)rez;
        }


    }




}


#pragma entry f_master master
void f_master(void)
{

     __setSrcImageID(2);

    int k;

    for(k=0; k<9; k++)
        projection_invers[k] = __getSrcf(k - __getX(), - __getY());

    unsigned begining_of_output = iter_num*(unsigned)((float)height_out/num_of_iter);
```

```
    unsigned x, y;
    float  pxt, pyt, pt;
    unsigned px, py;
    //uzmi iz scratchpad-a pocevsi od pozicije (0,gyi) dva reda i prebaci ih u lwm
    for(k=0; k<width_out; k++){
        x = k;
        y = __getY() + begining_of_output;


        pxt = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
        pyt = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
        pt  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

        px = pxt/pt;
        py = pyt/pt;

        py-=begining_of_input;


        dma_copy_spad_lwm(px, py, (int)(input_data + 12*k), width_in);
        __waitdma();

        dma_copy_spad_lwm(px, py+1, (int)(input_data + 12*k + 6), width_in);
        __waitdma();
    }

    for(k=0; k < 7; k++){
        __actst(k);
    }

    interpolate(7);
    __syncg();

    //prebaci rezultat koji je u output_data na poziciju (0, getY) u scratchpad
    //tj. jedan obradjen red
    dma_copy_lwm_spad( __getY(), (int)output_data, width_out*3);

    __waitdma();

    __trap();

}


#pragma entry f_slave slave
void f_slave(int x)
{
    interpolate(x);

    __trap();


}
```

```
#pragma inline_asm dma_copy_spad_lwm
void dma_copy_spad_lwm(int x, int y, int lwm_base, int width_in){

    MOVU.I  R20, 0xf001c780

    SHLI    R16, R16, 16        # x=x<<16
    OR      R16, R16, R17       # x = x | y
    STI     R16, R20, 4         # postavi x

    STI     R18, R20, 8         # postavi lwm_base
    STI     R19, R20, 12        # postavi width_in koji prestvalja stride

    MOVI    R24, 0x1            # height = 1
    MOVI    R25, 0x6            # treba nam 2 piksela tj. 6 elemenata
    SHLI    R25, R25, 16        # width = width << 16
    OR      R24, R25, R24       # height = width | height
    STI     R24, R20, 16        # postavi height
    MOVI    R22, 0x3101
    # DMA control: dir = 1 (scr=>LWM), selpin = 0 (transfer finish), start

    STI     R22, R20, 0         # TGDMCR0 : DMA control
}

#pragma inline_asm dma_copy_lwm_spad
void dma_copy_lwm_spad(int y, int lwm_base, int stride){

    MOVU.I  R20, 0xf001c780

    STI     R16, R20, 4
    # R16 je prvi argument funkcije, tj. y, trebaju nam
    #pocetne koordinate x<<16|y, x=0, tako da dobijamo lepo 0<<16|y = y

    STI     R17, R20, 8         # postavi lwm_base
    STI     R18, R20, 12        # postavi stride

    SHLI    R18, R18, 16        # width<=>stride   stride=stride << 16
    MOVI    R23, 0x1            # height <=> R23 =1
    OR      R18, R18, R23       # stride= stride | height
    STI     R18, R20, 16        # postavi stride

    MOVI    R22, 0x3011
    STI     R22, R20, 0         # TGDMCR0 : DMA control
}

# kod za najblizeg suseda
#include <stdint.h>
#define MAX_WIDTH_OUT 750
#define MAX_WIDTH_IN 3000

const int iter_num = 0;
const int height_in = 0;
const int height_out = 0;
const int num_of_iter = 0;
```

```
const int begining_of_input = 0;
const int width_out = 0;
const int width_in = 0;

#pragma section bss lwm_comm
uint32_t input_data[MAX_WIDTH_IN];
uint32_t output_data[MAX_WIDTH_OUT];

void dma_copy_spad_lwm(int x, int y, int lwm_base, int width_in);
void dma_copy_lwm_spad(int y, int lwm_base, int stride);

#pragma section bss lwm
float projection_invers[9];

void interpolate(int thread_num){


    int k, j;


    __setSrcImageID(0);
    __setDstImageID(1);

    int duzina = width_out/8;

    for(k=thread_num*duzina; k<(thread_num+1)*duzina; k++){
        for(j=0; j<3; j++){
            output_data[3*k + j] = input_data[3*k + j];
        }
    }
}


#pragma entry f_master master
void f_master(void)
{

     __setSrcImageID(2);

    int k;

    for(k=0; k<9; k++)
        projection_invers[k] = __getSrcf(k - __getX(), - __getY());

    unsigned begining_of_output = iter_num*(unsigned)((float)height_out/num_of_iter);


    unsigned x, y;
    float  pxt, pyt, pt;
    unsigned px, py;
    float dx, dy;
    //uzmi iz scratchpad-a pocevsi od pozicije (0,gyi) dva reda i prebaci ih u lwm
    for(k=0; k<width_out; k++){
```

```
        x = k;
        y = __getY() + begining_of_output;


        pxt = projection_invers[0]*x + projection_invers[1]*y + projection_invers[2]*t;
        pyt = projection_invers[3]*x + projection_invers[4]*y + projection_invers[5]*t;
        pt  = projection_invers[6]*x + projection_invers[7]*y + projection_invers[8]*t;

        px = pxt/pt;
        py = pyt/pt;

        dx = pxt/pt - px;
        dy = pyt/pt - py;

        if(dx>0.5){
            px++;
        }

        if(dy>0.5){
            py++;
        }

        py-=begining_of_input;


        dma_copy_spad_lwm(px, py, (int)(input_data + 3*k), width_in);
        __waitdma();

    }

    for(k=0; k < 7; k++){
        __actst(k);
    }

    interpolate(7);
    __syncg();

    //prebaci rezultat koji je u output_data na poziciju (0, getY) u scratchpad
    //tj. jedan obradjen red
    dma_copy_lwm_spad( __getY(), (int)output_data, width_out*3);

    __waitdma();

    __trap();

}


#pragma entry f_slave slave
void f_slave(int x)
{
    interpolate(x);
```

```
    __trap();


}
#pragma inline_asm dma_copy_spad_lwm
void dma_copy_spad_lwm(int x, int y, int lwm_base, int width_in){

    MOVU.I  R20, 0xf001c780

    SHLI    R16, R16, 16        # x=x<<16
    OR      R16, R16, R17       # x = x | y
    STI     R16, R20, 4         # postavi x

    STI     R18, R20, 8         # postavi lwm_base
    STI     R19, R20, 12        # postavi width_in koji prestvalja stride

    MOVI    R24, 0x1            # height = 1
    MOVI    R25, 0x3            # treba nam 1 piksela tj. 3 elemenata
    SHLI    R25, R25, 16        # width = width << 16
    OR      R24, R25, R24       # height = width | height
    STI     R24, R20, 16        # postavi height
    MOVI    R22, 0x3101
    # DMA control: dir = 1 (scr=>LWM), selpin = 0 (transfer finish), start

    STI     R22, R20, 0         # TGDMCR0 : DMA control
}

#pragma inline_asm dma_copy_lwm_spad
void dma_copy_lwm_spad(int y, int lwm_base, int stride){

    MOVU.I  R20, 0xf001c780

    STI     R16, R20, 4
    # R16 je prvi argument funkcije, tj. y, trebaju nam
    #pocetne koordinate x<<16|y, x=0, tako da dobijamo lepo 0<<16|y = y

    STI     R17, R20, 8         # postavi lwm_base
    STI     R18, R20, 12        # postavi stride

    SHLI    R18, R18, 16        # width<=>stride   stride=stride << 16
    MOVI    R23, 0x1            # height <=> R23 =1
    OR      R18, R18, R23       # stride= stride | height
    STI     R18, R20, 16        # postavi stride

    MOVI    R22, 0x3011
    STI     R22, R20, 0         # TGDMCR0 : DMA control
}
```

A.15 Кôд покретање трећег дела рада

```
res = subprocess.run(['scp', '/c/Users/bozica/Desktop/test.ppm',
'root@10.10.96.176:/home/root/bozica'])
```

```
res = subprocess.run(['scp', '/c/Users/bozica/Desktop/koordinate.txt',
'root@10.10.96.176:/home/root/bozica'])
res = subprocess.run(['ssh', 'root@10.10.96.176',
'cd', 'bozica' + ';', 'export',
'LD_LIBRARY_PATH=/home/root/bozica' + ';', './transformation_app'])
res = subprocess.run(['scp',
'root@10.10.96.176:/home/root/bozica/outputFAST_CORNER.ppm',
'/c/Users/bozica/Desktop/rezultatBilin.ppm'])
res = subprocess.run(['scp',
'root@10.10.96.176:/home/root/bozica/output_FAST_CORNER_neighbor.ppm',
'/c/Users/bozica/Desktop/rezultatNeighbor.ppm'])
```

Додатак B

Решавање система линеарних једначина

$$ax_0 + by_0 + c = 0 \tag{5.1}$$
$$dx_0 + ey_0 + f = 0 \tag{5.2}$$
$$gx_0 + hy_0 + i = t_0 \tag{5.3}$$

$$ax_1 + by_1 + c = 0 \tag{5.4}$$
$$dx_1 + ey_1 + f = t_1 \tag{5.5}$$
$$gx_1 + hy_1 + i = t_1 \tag{5.6}$$

$$ax_2 + by_2 + c = t_2 \tag{5.7}$$
$$dx_2 + ey_2 + f = t_2 \tag{5.8}$$
$$gx_2 + hy_2 + i = t_2 \tag{5.9}$$

$$ax_3 + by_3 + c = t_3 \tag{5.10}$$
$$dx_3 + ey_3 + f = 0 \tag{5.11}$$
$$gx_3 + hy_3 + i = t_3. \tag{5.12}$$

Из (5.1), (5.2) и (5.3) важи $c = -ax_0 - by_0$, $f = -dx_0 - ey_0$, $i = t_0 - gx_0 - hy_0$.

Заменом c, f, i у осталим једначинама добијамо:

$$ax_1 + by_1 - ax_0 - by_0 = 0 \tag{5.4}$$
$$dx_1 + ey_1 - dx_0 - ey_0 = t_1 \tag{5.5}$$
$$gx_1 + hy_1 + t_0 - gx_0 - hy_0 = t_1 \tag{5.6}$$

$$ax_2 + by_2 - ax_0 - by_0 = t_2 \tag{5.7}$$
$$dx_2 + ey_2 - dx_0 - ey_0 = t_2 \tag{5.8}$$
$$gx_2 + hy_2 + t_0 - gx_0 - hy_0 = t_2 \tag{5.9}$$

$$ax_3 + by_3 - ax_0 - by_0 = t_3 \tag{5.10}$$
$$dx_3 + ey_3 - dx_0 - ey_0 = 0 \tag{5.11}$$
$$gx_3 + hy_3 + t_0 - gx_0 - hy_0 = t_3. \tag{5.12}$$

Груписањем добијамо:

$$a(x_1 - x_0) + b(y_1 - y_0) = 0 \tag{5.4}$$
$$d(x_1 - x_0) + e(y_1 - y_0) = t_1 \tag{5.5}$$
$$g(x_1 - x_0) + h(y_1 - y_0) = t_1 - t_0 \tag{5.6}$$

$$a(x_2 - x_0) + b(y_2 - y_0) = t_2 \tag{5.7}$$
$$d(x_2 - x_0) + e(y_2 - y_0) = t_2 \tag{5.8}$$
$$g(x_2 - x_0) + h(y_2 - y_0) = t_2 - t_0 \tag{5.9}$$

$$a(x_3 - x_0) + b(y_3 - y_0) = t_3 \tag{5.10}$$
$$d(x_3 - x_0) + e(y_3 - y_0) = 0 \tag{5.11}$$
$$g(x_3 - x_0) + h(y_3 - y_0) = t_3 - t_0. \tag{5.12}$$

Одавде можемо да изаберемо по којим параметрима желимо да раздвојимо случајеве. Рецимо да гледмао по $x_1 - x_0$, тада имамо два случаја:

1. $x_1 - x_0 = 0$

$$b(y_1 - y_0) = 0 \tag{5.4}$$
$$e(y_1 - y_0) = t_1 \tag{5.5}$$
$$h(y_1 - y_0) = t_1 - t_0 \tag{5.6}$$

$$a(x_2 - x_0) + b(y_2 - y_0) = t_2 \tag{5.7}$$
$$d(x_2 - x_0) + e(y_2 - y_0) = t_2 \tag{5.8}$$
$$g(x_2 - x_0) + h(y_2 - y_0) = t_2 - t_0 \tag{5.9}$$

$$a(x_3 - x_0) + b(y_3 - y_0) = t_3 \tag{5.10}$$
$$d(x_3 - x_0) + e(y_3 - y_0) = 0 \tag{5.11}$$
$$g(x_3 - x_0) + h(y_3 - y_0) = t_3 - t_0. \tag{5.12}$$

Приметимо да мора да важи $y_1 - y_0 \neq 0$. У супротном би важило да је $A = B$, тј. не бисмо имали четири тачке у општем положају. Из овога следи:

$$b = 0 \tag{5.4}$$
$$e = \frac{t_1}{y_1 - y_0} \tag{5.5}$$
$$h = \frac{t_1 - t_0}{y_1 - y_0}. \tag{5.6}$$

Заменом b, e и h, у преостале једначине добијамо:

$$a(x_2 - x_0) = t_2 \tag{5.7}$$

$$d(x_2 - x_0) + t_1\frac{y_2 - y_0}{y_1 - y_0} = t_2 \tag{5.8}$$

$$g(x_2 - x_0) + (y_2 - y_0)\frac{t_1 - t_0}{y_1 - y_0} = t_2 - t_0 \tag{5.9}$$

$$a(x_3 - x_0) = t_3 \tag{5.10}$$

$$d(x_3 - x_0) + (y_3 - y_0)\frac{t_1}{y_1 - y_0} = 0 \tag{5.11}$$

$$g(x_3 - x_0) + (y_3 - y_0)\frac{t_1 - t_0}{y_1 - y_0} = t_3 - t_0. \tag{5.12}$$

Пошто важи да је $x_1 - x_0 = 0$, онда мора да важи да је $x_2 - x_0 \neq 0$. У супротном би тачке A, B и C биле колинеарне. Сличним резоновањем мора да важи $x_3 - x_0 \neq 0$.

На основу тога следи:

$$a = \frac{t_2}{x_2 - x_0} \tag{5.7}$$

$$d + t_1\frac{y_2 - y_0}{(y_1 - y_0)(x_2 - x_0)} = \frac{t_2}{x_2 - x_0} \tag{5.8}$$

$$g + (y_2 - y_0)\frac{t_1 - t_0}{(y_1 - y_0)(x_2 - x_0)} = \frac{t_2 - t_0}{x_2 - x_0} \tag{5.9}$$

$$a = \frac{t_3}{x_3 - x_0} \tag{5.10}$$

$$d + (y_3 - y_0)\frac{t_1}{(y_1 - y_0)(x_3 - x_0)} = 0 \tag{5.11}$$

$$g + (y_3 - y_0)\frac{t_1 - t_0}{(y_1 - y_0)(x_3 - x_0)} = \frac{t_3 - t_0}{x_3 - x_0}. \tag{5.12}$$

Одатле добијамо да морају да важе следеће једнакости:

$$a = \frac{t_2}{x_2 - x_0} \tag{5.7}$$

$$d = \frac{t_2}{x_2 - x_0} - t_1 \frac{y_2 - y_0}{(y_1 - y_0)(x_2 - x_0)} \tag{5.8}$$

$$g = \frac{t_2 - t_0}{x_2 - x_0} - (y_2 - y_0) \frac{t_1 - t_0}{(y_1 - y_0)(x_2 - x_0)} \tag{5.9}$$

$$a = \frac{t_3}{x_3 - x_0} \tag{5.10}$$

$$d = -(y_3 - y_0) \frac{t_1}{(y_1 - y_0)(x_3 - x_0)} \tag{5.11}$$

$$g = \frac{t_3 - t_0}{x_3 - x_0} - (y_3 - y_0) \frac{t_1 - t_0}{(y_1 - y_0)(x_3 - x_0)}. \tag{5.12}$$

Изједначавањем десних страна јендакости (5.7) и (5.10), (5.8) и (5.11), (5.9) и (5.12) добијамо:

$$\frac{t_2}{x_2 - x_0} = \frac{t_3}{x_3 - x_0} \tag{5.7,5.10}$$

$$(y_3 - y_0) \frac{t_1}{(y_1 - y_0)(x_3 - x_0)} = t_1 \frac{y_2 - y_0}{(y_1 - y_0)(x_2 - x_0)} - \frac{t_2}{x_2 - x_0} \tag{5.8, 5.11}$$

$$\begin{aligned} &(y_2 - y_0) \frac{t_1 - t_0}{(y_1 - y_0)(x_2 - x_0)} - \frac{t_2 - t_0}{x_2 - x_0} \\ &= (y_3 - y_0) \frac{t_1 - t_0}{(y_1 - y_0)(x_3 - x_0)} - \frac{t_3 - t_0}{x_3 - x_0}. \end{aligned} \tag{5.9, 5.12}$$

Груписањем можемо изразити t_3 и t_2 преко t_1:

$$t_3 = \frac{t_2(x_3 - x_0)}{x_2 - x_0} \tag{5.7, 5.10}$$

$$t_2 = t_1(x_2 - x_0)\left(\frac{y_2 - y_0}{(y_1 - y_0)(x_2 - x_0)} - \frac{y_3 - y_0}{(y_1 - y_0)(x_3 - x_0)}\right). \tag{5.8, 5.11}$$

Престало је још средити последњу једначину (5.9, 5.12),

$$(y_2 - y_0) \frac{t_1 - t_0}{(y_1 - y_0)(x_2 - x_0)} - \frac{t_2 - t_0}{x_2 - x_0} = (y_3 - y_0) \frac{t_1 - t_0}{(y_1 - y_0)(x_3 - x_0)} - \frac{t_3 - t_0}{x_3 - x_0}$$

у којој желимо да изразимо t_1 преко t_0, при чему ће t_0 представљати слободан параметар.

Помножимо једначину са $(x_3 - x_0)(y_1 - y_0)(x_2 - x_0) \neq 0$:

$$\begin{aligned}(y_2 - y_0)(t_1 - t_0)(x_3 - x_0) - (t_2 - t_0)(y_1 - y_0)(x_3 - x_0) = \\ (y_3 - y_0)(x_2 - x_0)(t_1 - t_0) - (t_3 - t_0)(x_2 - x_0)(y_1 - y_0).\end{aligned}$$

Групишимо шта стоји уз t_0, t_1, t_2, t_3:

$$\begin{aligned}t_1((y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0)) - \\ t_2(y_1 - y_0)(x_3 - x_0) + t_3(x_2 - x_0)(y_1 - y_0) = \\ t_0((x_2 - x_0)(y_1 - y_0) - (y_3 - y_0)(x_2 - x_0) \\ +(y_2 - y_0)(x_3 - x_0) - (y_1 - y_0)(x_3 - x_0)).\end{aligned}$$

Важи да је:

$$\begin{aligned}(x_2 - x_0)(y_1 - y_0) - (y_3 - y_0)(x_2 - x_0) + \\ (y_2 - y_0)(x_3 - x_0) - (y_1 - y_0)(x_3 - x_0) = \\ (x_2 - x_0)(y_1 - y_0) - (y_3 - y_0)(x_2 - x_0) + (y_2 - y_1)(x_3 - x_0),\end{aligned}$$

па то можемо заменити у нашој једначини.

$$\begin{aligned}t_1((y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0)) - \\ t_2(y_1 - y_0)(x_3 - x_0) + t_3(x_2 - x_0)(y_1 - y_0) = \\ t_0((x_2 - x_0)(y_1 - y_0) - (y_3 - y_0)(x_2 - x_0) + (y_2 - y_1)(x_3 - x_0))\end{aligned}$$

Преостаје да убацимо вредности t_3 и t_2, изражене преко t_1, тј. (5.7, 5.10), (5.8, 5.11).

Убацимо прво t_3, чиме добијамо:

$$\begin{aligned}t_1((y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0)) \\ -t_2(y_1 - y_0)(x_3 - x_0) + t_2(x_3 - x_0)(y_1 - y_0) = \\ t_0((x_2 - x_0)(y_1 - y_3) + (y_2 - y_1)(x_3 - x_0)).\end{aligned}$$

Поскрате се коефицијенти који стоје уз t_2, чиме добијамо:

$$t_1((y_2-y_0)(x_3-x_0)-(y_3-y_0)(x_2-x_0)) = t_0((x_2-x_0)(y_1-y_3)+(y_2-y_1)(x_3-x_0)).$$

Да бисмо изразили t_1, морамо да проверимо да ли је $(y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0) \neq 0$.

Претпоставимо супротно. Тада важи:

$$(y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0) = 0$$

$$x_3y_2 - x_3y_0 - x_0y_2 + x_0y_0 - x_2y_3 + x_2y_0 + x_0y_3 - x_0y_0 = 0$$

$$x_3y_2 - x_3y_0 - x_0y_2 - x_2y_3 + x_2y_0 + x_0y_3 = 0$$

$$\begin{vmatrix} x_0 & y_0 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0.$$

Међутим, по претпоставци да имамо тачке у општем положају, ова детерминанта мора да буде различита од нуле. Тиме добијамо:

$$t_1 = t_0 \frac{(x_2 - x_0)(y_1 - y_3) + (y_2 - y_1)(x_3 - x_0)}{(y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0)}.$$

Параметар t_0 можемо произвољно одабрити из $\mathbb{R}_{\neq 0}$.

Могуће је изабрати t_0, тако да елементи матрице буду целобројни. Међутим, тиме може доћи до прекорачења, што ми се десило током изражавања решења на тај начин. Иако неидеално, због заокруживања реалних бројева до одређене цифре у рачунарству, сигурнији и довољно прецизни резултати се постижу одабиром да $t_0 = 1$. С обзиром да су разлике хомогених координата улазних тачака лепо балансиране у имениоцима и бројиоцима решења, вероватноћа прекорачења је знатно мања.

Коначно решење је:

$$t_0 = 1$$

$$t_1 = t_0 \frac{(x_2 - x_0)(y_1 - y_3) + (y_2 - y_1)(x_3 - x_0)}{(y_2 - y_0)(x_3 - x_0) - (y_3 - y_0)(x_2 - x_0)}$$

$$t_2 = t_1(x_2 - x_0)\left(\frac{y_2 - y_0}{(y_1 - y_0)(x_2 - x_0)} - \frac{y_3 - y_0}{(y_1 - y_0)(x_3 - x_0)}\right)$$

$$t_3 = \frac{t_2(x_3 - x_0)}{x_2 - x_0}$$

$$a = \frac{t_2}{x_2 - x_0}$$

$$d = \frac{t_2}{x_2 - x_0} - \frac{t_1(y_2 - y_0)}{(y_1 - y_0)(x_2 - x_0)}$$
$$g = \frac{t_2 - t_0}{x_2 - x_0} - \frac{(t_1 - t_0)(y_2 - y_0)}{(y_1 - y_0)(x_2 - x_0)}$$
$$b = 0$$
$$e = \frac{t_1}{y_1 - y_0}$$
$$h = \frac{t_1 - t_0}{y_1 - y_0}$$
$$c = -ax_0 - by_0$$
$$f = -dx_0 - ey_0$$
$$i = t_0 - gx_0 - hy_0.$$

На жалост ово није био лакши случај. Изведимо решење за други случај.

2. $x_1 - x_0 \neq 0$ Вратимо се до места рачвања случајева.

$$c = -ax_0 - by_0 \tag{5.1}$$
$$f = -dx_0 - ey_0 \tag{5.2}$$
$$i = t_0 - gx_0 - hy_0 \tag{5.3}$$

$$a(x_1 - x_0) + b(y_1 - y_0) = 0 \tag{5.4}$$
$$d(x_1 - x_0) + e(y_1 - y_0) = t_1 \tag{5.5}$$
$$g(x_1 - x_0) + h(y_1 - y_0) = t_1 - t_0 \tag{5.6}$$

$$a(x_2 - x_0) + b(y_2 - y_0) = t_2 \tag{5.7}$$
$$d(x_2 - x_0) + e(y_2 - y_0) = t_2 \tag{5.8}$$
$$g(x_2 - x_0) + h(y_2 - y_0) = t_2 - t_0 \tag{5.9}$$

$$a(x_3 - x_0) + b(y_3 - y_0) = t_3 \tag{5.10}$$
$$d(x_3 - x_0) + e(y_3 - y_0) = 0 \tag{5.11}$$
$$g(x_3 - x_0) + h(y_3 - y_0) = t_3 - t_0 \tag{5.12}$$

Пошто је $x_1 - x_0 \neq 0$, можемо једначине (5.4), (5.5), (5.6) поделити са $x_1 - x_0$. Чиме добијамо:

$$a + b\frac{y_1 - y_0}{x_1 - x_0} = 0 \tag{5.4}$$

$$d + e\frac{y_1 - y_0}{x_1 - x_0} = \frac{t_1}{x_1 - x_0} \tag{5.5}$$

$$g + h\frac{y_1 - y_0}{x_1 - x_0} = \frac{t_1 - t_0}{x_1 - x_0}. \tag{5.6}$$

Одатле можемо изразити елементе матрице a, d и g.

$$a = -b\frac{y_1 - y_0}{x_1 - x_0} \tag{5.4}$$

$$d = \frac{t_1 - e(y_1 - y_0)}{x_1 - x_0} \tag{5.5}$$

$$g = \frac{t_1 - t_0 - h(y_1 - y_0)}{x_1 - x_0} \tag{5.6}$$

Убацимо новоизражене a, d и g у преостих шест једначина.

$$-(x_2 - x_0)b\frac{y_1 - y_0}{x_1 - x_0} + b(y_2 - y_0) = t_2 \tag{5.7}$$

$$(x_2 - x_0)\frac{t_1 - e(y_1 - y_0)}{x_1 - x_0} + e(y_2 - y_0) = t_2 \tag{5.8}$$

$$(x_2 - x_0)t_1 - t_0 - h(y_1 - y_0)x_1 - x_0 + h(y_2 - y_0) = t_2 - t_0 \tag{5.9}$$

$$-(x_3 - x_0)b\frac{y_1 - y_0}{x_1 - x_0} + b(y_3 - y_0) = t_3 \tag{5.10}$$

$$(x_3 - x_0)\frac{t_1 - e(y_1 - y_0)}{x_1 - x_0} + e(y_3 - y_0) = 0 \tag{5.11}$$

$$(x_3 - x_0)\frac{t_1 - t_0 - h(y_1 - y_0)}{x_1 - x_0} + h(y_3 - y_0) = t_3 - t_0 \tag{5.12}$$

Помножимо једначине са $(x_1 - x_0) \neq 0$, чиме добијамо:

$$-(x_2 - x_0)b(y_1 - y_0) + b(y_2 - y_0)(x_1 - x_0) = t_2(x_1 - x_0) \tag{5.7}$$

$$(x_2 - x_0)(t_1 - e(y_1 - y_0)) + e(y_2 - y_0)(x_1 - x_0) = t_2(x_1 - x_0) \tag{5.8}$$

$$(x_2 - x_0)(t_1 - t_0 - h(y_1 - y_0)) + h(y_2 - y_0)(x_1 - x_0) = (t_2 - t_0)(x_1 - x_0) \tag{5.9}$$

$$-(x_3 - x_0)b(y_1 - y_0) + b(y_3 - y_0)(x_1 - x_0) = t_3(x_1 - x_0) \tag{5.10}$$

$$(x_3 - x_0)(t_1 - e(y_1 - y_0)) + e(y_3 - y_0)(x_1 - x_0) = 0 \tag{5.11}$$

$$(x_3 - x_0)(t_1 - t_0 - h(y_1 - y_0)) + h(y_3 - y_0)(x_1 - x_0) = (t_3 - t_0)(x_1 - x_0). \tag{5.12}$$

Груписањем коефицијената уз елементе матрице добијамо:

$$\begin{aligned} b(-(x_2 - x_0)(y_1 - y_0) + (y_2 - y_0)(x_1 - x_0)) & \\ = t_2(x_1 - x_0) & \quad (5.7) \\ e((y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)) & \\ = t_2(x_1 - x_0) - (x_2 - x_0)t_1 & \quad (5.8) \\ h((y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)) & \\ = (t_2 - t_0)(x_1 - x_0) - (x_2 - x_0)(t_1 - t_0) & \quad (5.9) \end{aligned}$$

$$\begin{aligned} b((y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0)) & \\ = t_3(x_1 - x_0) & \quad (5.10) \\ e((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)) & \\ = -(x_3 - x_0)t_1 & \quad (5.11) \\ h((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)) & \\ = (t_3 - t_0)(x_1 - x_0) - (x_3 - x_0)(t_1 - t_0). & \quad (5.12) \end{aligned}$$

Да бисмо изразили елементе матрице пројективног пресликавања, морамо да проверимо да ли смемо да делимо са $((y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0))$ и са $(-(x_2 - x_0)(y_1 - y_0) + (y_2 - y_0)(x_1 - x_0))$, тј. да ли су ти коефицијенти различити од нуле. По претпоставци, $x_1 - x_0 \neq 0$, и $t_2 \neq 0$, чиме је десна страна једнакости једначине (5.7) раличита од 0, па тиме мора и лева страна да буде различита од нуле. Одатле следи да је $(-(x_2 - x_0)(y_1 - y_0) + (y_2 - y_0)(x_1 - x_0)) \neq 0$. Сличним резоновањем, пошто је $t_3 \neq 0$, из јендачине (5.10), добијамо да је и $((y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0)) \neq 0$. Због тога важи:

$$b = \frac{t_2(x_1 - x_0)}{(y_2 - y_0)(x_1 - x_0) - (x_2 - x_0)(y_1 - y_0)} \tag{5.7}$$

$$e = \frac{t_2(x_1 - x_0) - (x_2 - x_0)t_1}{(y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)} \tag{5.8}$$

$$h = \frac{(t_2 - t_0)(x_1 - x_0) - (x_2 - x_0)(t_1 - t_0)}{(y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)} \tag{5.9}$$

$$b = \frac{t_3(x_1 - x_0)}{(y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0)} \tag{5.10}$$

$$e = \frac{-t_1(x_3 - x_0)}{(y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)} \tag{5.11}$$

$$h = \frac{(t_3 - t_0)(x_1 - x_0) - (x_3 - x_0)(t_1 - t_0)}{(y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)}. \tag{5.12}$$

Изједначавањем одговарајућих једначина добијамо:

$$\frac{t_2(x_1 - x_0)}{(y_2 - y_0)(x_1 - x_0) - (x_2 - x_0)(y_1 - y_0)} = \frac{t_3(x_1 - x_0)}{(y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0)} \tag{5.7, 5.10}$$

$$\frac{t_2(x_1 - x_0) - (x_2 - x_0)t_1}{(y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)} = \frac{-t_1(x_3 - x_0)}{(y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)} \tag{5.8, 5.11}$$

$$\frac{(t_2 - t_0)(x_1 - x_0) - (x_2 - x_0)(t_1 - t_0)}{(y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)} = \frac{(t_3 - t_0)(x_1 - x_0) - (x_3 - x_0)(t_1 - t_0)}{(y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)}. \tag{5.9, 5.12}$$

У јендачини (5.7, 5.10) можемо да скратимо $x_1 - x_0 \neq 0$, а све једначине можемо да помножимо са $((y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0)) \neq 0$,

$(-(x_2 - x_0)(y_1 - y_0) + (y_2 - y_0)(x_1 - x_0)) \neq 0$. Тиме добијамо:

$$t_3 = \frac{t_2((y_3 - y_0)(x_1 - x_0) - (x_3 - x_0)(y_1 - y_0))}{(y_2 - y_0)(x_1 - x_0) - (x_2 - x_0)(y_1 - y_0)} \tag{5.7, 5.10}$$

$$((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0))(t_2(x_1 - x_0) - (x_2 - x_0)t_1) = -t_1(x_3 - x_0)((y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)) \tag{5.8, 5.11}$$

$$((t_2 - t_0)(x_1 - x_0) - (x_2 - x_0)(t_1 - t_0))((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)) = ((y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0))((t_3 - t_0)(x_1 - x_0) - (x_3 - x_0)(t_1 - t_0)). \tag{5.9, 5.12}$$

Можемо изразити t_2 преко t_1 у јендачини (5.8, 5.11), чиме добијамо:

$$\begin{aligned} t_2 = t1(&-(x_3 - x_0)(y_2 - y_0)(x_1 - x_0) + (x_3 - x_0)(y_1 - y_0)(x_2 - x_0) + \\ &(x_2 - x_0)(y_3 - y_0)(x_1 - x_0) - (x_2 - x_0)(y_1 - y_0)(x_3 - x_0)))/ \\ &((x_1 - x_0)((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0))). \end{aligned}$$

Остало је још средити једначину (5.9, 5.12). Груписањем добијамо:

$$(t_2(x_1 - x_0) - t_1(x_2 - x_0) + t_0(x_2 - x_1))((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)) = (t_3(x_1 - x_0) - t_1(x_3 - x_0) + t_0(x_3 - x_1))((y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)) \tag{5.9, 5.12}$$

$$\begin{aligned} &t_0((y_3 - y_0)(x_1 - x_0)(x_2 - x_1) - (y_1 - y_0)(x_3 - x_0)(x_2 - x_1) - \\ &(y_2 - y_0)(x_1 - x_0)(x_3 - x_1) + (y_1 - y_0)(x_2 - x_0)(x_3 - x_1)) = \\ &-t_2(x_1 - x_0)((y_3 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_3 - x_0)) + \\ &t_3(x_1 - x_0)((y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)) + \\ &t_1((y_3 - y_0)(x_1 - x_0)(x_2 - x_0) - (y_1 - y_0)(x_3 - x_0)(x_2 - x_0) - \\ &(y_2 - y_0)(x_1 - x_0)(x_3 - x_0) + (y_1 - y_0)(x_2 - x_0)(x_3 - x_0)). \end{aligned} \tag{5.9, 5.12}$$

Да бисмо изразили t_0, морамо да проверимо да ли је $((y_3 - y_0)(x_1 - x_0)(x_2 - x_1) - (y_1 - y_0)(x_3 - x_0)(x_2 - x_1) - (y_2 - y_0)(x_1 - x_0)(x_3 - x_1) + (y_1 - y_0)(x_2 - x_0)(x_3 - x_1)) \neq 0$.

$$\begin{aligned}((y_3-y_0)(x_1-x_0)(x_2-x_1)-(y_1-y_0)(x_3-x_0)(x_2-x_1)-\\(y_2-y_0)(x_1-x_0)(x_3-x_1)+(y_1-y_0)(x_2-x_0)(x_3-x_1))=\\\begin{vmatrix}x_1-x_0 & x_2-x_0 & x_3-x_0\\ y_1-y_0 & y_2-y_0 & y_3-y_0\\ 0 & x_1-x_2 & x_1-x_3\end{vmatrix}\end{aligned}\qquad(\bigstar)$$

Кад у детерминанти одузмемо од треће колоне прву добијамо:

$$\begin{vmatrix}x_1-x_0 & x_2-x_0 & x_3-x_1\\ y_1-y_0 & y_2-y_0 & y_3-y_1\\ 0 & x_1-x_2 & x_1-x_3\end{vmatrix}=\qquad(\bigstar)$$

на прву врсту додајмо трећу:

$$\begin{vmatrix}x_1-x_0 & x_1-x_0 & 0\\ y_1-y_0 & y_2-y_0 & y_3-y_1\\ 0 & x_1-x_2 & x_1-x_3\end{vmatrix}=\qquad(\bigstar)$$

од друге колоне одумимо прву:

$$\begin{vmatrix}x_1-x_0 & 0 & 0\\ |y_1-y_0 & y_2-y_1 & y_3-y_1\\ 0 & x_1-x_2 & x_1-x_3\end{vmatrix}=\qquad(\bigstar)$$

развијањем по првој врсти, добијамо:

$$(x_1-x_0)((x_1-x_3)(y_2-y_1)-(y_3-y_1)(x_1-x_2)).$$

Знамо да је $x_1-x_0\neq 0$, зато је довољно да покажемо да је $(x_1-x_3)(y_2-y_1)-(y_3-y_1)(x_1-x_2)\neq 0$. Важе следеже једнакости:

$$\begin{aligned}(x_2-x_1)(y_3-y_1)-(x_3-x_1)(y_2-y_1)=\\x_2y_3-x_1y_3-x_2y_1+x_1y_1-x_3y_2+x_1y_2+x_3y_1-x_1y_1=\\x_2y_3-x_1y_3-x_2y_1-x_3y_2+x_1y_2+x_3y_1=\\x_2(y_3-y_1)+x_1(y_2-y_3)+x_3(y_1-y_2)=\\-x_2(y_1-y_3)+x_1(y_2-y_3)+x_3(y_1-y_2)=\\\begin{vmatrix}x_1 & y_1 & 1\\ x_2 & y_2 & 1\\ x_3 & y_3 & 1\end{vmatrix}\neq 0.\end{aligned}$$

Слично, можемо одабрати t_1, као слободан параметар са вредношћу један, чиме добијамо коначно решење:

$$
\begin{aligned}
t_1 &= 1\\
t_2 &= t_1(-(x_3-x_0)(y_2-y_0)(x_1-x_0)+(x_3-x_0)(y_1-y_0)(x_2-x_0)+\\
&\quad (x_2-x_0)(y_3-y_0)(x_1-x_0)-(x_2-x_0)(y_1-y_0)(x_3-x_0))/\\
&\quad ((x_1-x_0)((y_3-y_0)(x_1-x_0)-(y_1-y_0)(x_3-x_0)))\\
t_3 &= t_2((y_3-y_0)(x_1-x_0)-(x_3-x_0)(y_1-y_0))/\\
&\quad (-(x_2-x_0)(y_1-y_0)+(y_2-y_0)(x_1-x_0))\\
t_0 &= (-t_2(x_1-x_0)((y_3-y_0)(x_1-x_0)-(y_1-y_0)(x_3-x_0))\\
&\quad +t_3(x_1-x_0)((y_2-y_0)(x_1-x_0)-(y_1-y_0)(x_2-x_0))+\\
&\quad t_1((y_3-y_0)(x_1-x_0)(x_2-x_0)-(y_1-y_0)(x_3-x_0)(x_2-x_0)-\\
&\quad (y_2-y_0)(x_1-x_0)(x_3-x_0)+(y_1-y_0)(x_2-x_0)(x_3-x_0)))\\
&\quad /((y_3-y_0)(x_1-x_0)(x_2-x_1)-(y_1-y_0)(x_3-x_0)(x_2-x_1)-\\
&\quad (y_2-y_0)(x_1-x_0)(x_3-x_1)+(y_1-y_0)(x_2-x_0)(x_3-x_1))\\
b &= \frac{t_2(x_1-x_0)}{-(x_2-x_0)(y_1-y_0)+(y_2-y_0)(x_1-x_0)}\\
e &= \frac{t_2(x_1-x_0)-(x_2-x_0)t_1}{(y_2-y_0)(x_1-x_0)-(y_1-y_0)(x_2-x_0)}\\
h &= \frac{(t_2-t_0)(x_1-x_0)-(x_2-x_0)(t_1-t_0)}{(y_2-y_0)(x_1-x_0)-(y_1-y_0)(x_2-x_0)}\\
a &= \frac{-b(y_1-y_0)}{x_1-x_0}\\
d &= \frac{t_1-e(y_1-y_0)}{x_1-x_0}\\
g &= \frac{t_1-t_0-h(y_1-y_0)}{x_1-x_0}.
\end{aligned}
$$

Биографија аутора

Божица Сања Калинић (*Београд, 4. децембар 1996.* —) завршила је О.Ш. Јован Миодраговић 2011. године у Београду, основну музичку школу Јосип Славенски 2010. године, Шесту београдску гимназију 2015. године и Математички факултет у Београду 2019. године. Тренутно је на програму мастер студија на смеру рачунарство и инфорамтика. У академској години 2019/2020 је радила као сарадник у настави на Математичком факултету у Београду, а тренутно је на програму стипендирања у истраживачком институту РТ-РК.