



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
У НОВОМ САДУ




Никола Стојановић

ПРИМЕНА *OpenCV* БИБЛИОТЕКЕ ЗА ИМПЛЕМЕНТАЦИЈУ КОНТРОЛЕРА ВИДЕО ИГРЕ

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2018

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА	Лист/Листова:

(Податке уноси предметни наставник - ментор)

Врста студија:	<input type="checkbox"/> Основне академске студије <input type="checkbox"/> Основне струковне студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	Проф. др Мирослав Поповић

Студент:	Никола Стојановић	Број индекса:	RA 13/2014
Област:	Агентске технологије		
Ментор:	Проф. др Милан Видаковић		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; - литература 			

НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

Примена *OpenCV* библиотеке за имплементацију контролера видео игре

ТЕКСТ ЗАДАТКА:

Задатак рада представља развој контролера већ постојеће видео игре. Контролер ће бити реализован у Јава програмском језику уз употребу *OpenCV* библиотеке. Спецификација ће бити написана употребом *UML*-а.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ Студента; ☐ Ментора

САДРЖАЈ

1. Увод.....	2
2. Опис коришћених технологија	3
2.1 Промена формата слике	5
2.2 Морфолошке операције	6
2.3 Поједностављивање контура	7
2.4 Филтери за ублажавање слике.....	10
2.5 Гаусов филтер	11
2.6 Кенијев детектор ивица.....	13
2.7 Проналажење контура.....	15
2.8 Цртање контура.....	16
2.9 Одређивање површине контура	17
2.10 Одређивање просечног аритметичког одступања	17
3. Спецификација апликације	19
3.1 Дијаграм случајева коришћења	19
3.2 Дијаграм класа	20
3.3 Дијаграм секвенци.....	21
4. Опис имплементације	22
5. Закључак	33
6.КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА.....	35
7.KEY WORDS DOCUMENTATION.....	37

1.Увод

Задатак рада представља развој контролера постојеће видео игре која се састоји из два дела. Први део представља серверску апликацију, док други део представља клијентску апликацију. За успешно коришћење, потребно је остварити конекцију са серверском апликацијом и потом камером пронаћи оквир за гађање на серверској апликацији.

Рад је подељен у пет поглавља.

Прво поглавље представља увод у рад.

У другом поглављу је описана *OpenCV* библиотека као и њене функције које су примењене у раду.

Треће поглавље садржи спецификацију апликације, са описима дијаграма случајева коришћења, дијаграма класа, као и дијаграма секвенце. За опис дијаграма употребљен је обједињени језик за моделовање (*Unified Modeling Language - UML*).

У четвртом поглављу је описана имплементација апликације. Опис имплементације садржи релеватне слике и делове кода.

У петом поглављу изнет је закључак рада.

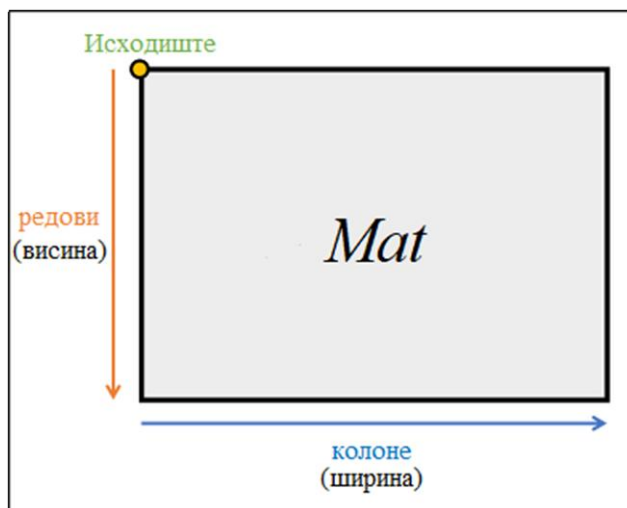
Рад је закључен навођењем коришћене литературе, као и кратком биографијом аутора.

2.Опис коришћених технологија

OpenCV (*Open Source Computer Vision Library*) је једна од најпознатијих библиотека отвореног кода која се користи за решавање проблема из области компјутерске визије. Она садржи на стотине унапред припремљених функција из те области. *OpenCV* пружа подршку за широк спектар оперативних система и платформи, као што су *GNU/Linux*, *Windows*, *OS X*, *Android*, *iOS*, као и подршку за више програмских језика (*C++*, *Java*, *Python*). [1]

Уз помоћ *OpenCV* библиотеке могуће је извршити много различитих операција над сликом, од цртања основних дводимензионалних геометријских облика (линија, круг, елипса) до сложених морфолошких операција.

У *OpenCV* библиотеци дигитална слика је представљена матрицом, односно *Mat* објектом.



Слика 2.1 *Mat* класа [11]

Број колона одговара ширини слике, док број редова одговара њезиној висини. Исходиште слике је у горњем левом углу, што је и видљиво на слици 2.1.

54	58	255	8	0		
45	0	78	51	100	74	
85	47	34	185	207	21	36
22	20	148	52	24	147	123
52	36	250	74	214	278	41
	158	0	78	51	247	255
		72	74	136	251	74

OpenCV може на више начина приказати slike. Конвенционални запис:

где $j \in \mathbb{N}$:

- U – неозначени (позитивни) цели бројеви
- S – означени цели бројеви
- F – реални бројеви

- CV_8UC1 – осмобитна неозначена слика са једним каналом,
- CV_8U – осмобитна слика са једним каналом која користи целе неозначене бројеве од 0 до 255.,

- CV_8S – осмобитна слика са једним каналом која користи целе означене бројеве од -128 до 127.,
- CV_16U – шеснаестобитна слика користи целе неозначене бројеве од 0 до 65.535.,
- CV_16S – шеснаестобитна слика користи целе означене бројеве од -32.768 до 32.767.,
- CV_32S – тридесетдвобитни цели бројеви од -2,147,483,648 до 2,147,483,647,
- CV_32F – тридесетдвобитни реални бројеви од -FLT_MAX до FLT_MAX и обухватају бесконачне (INF) вредности и вредности које нису бројеви (NAN) и
- CV_64F – шездесетчетверобитни реални бројеви од -DBL_MAX до DBL_MAX и обухватају бесконачне (INF) вредности и вредности које нису бројеви (NAN) [3]

2.1 Промена формата слике

У *OpenCV* библиотеци промена формата слике је имплементирана методом *cvtColor* чија декларација гласи:

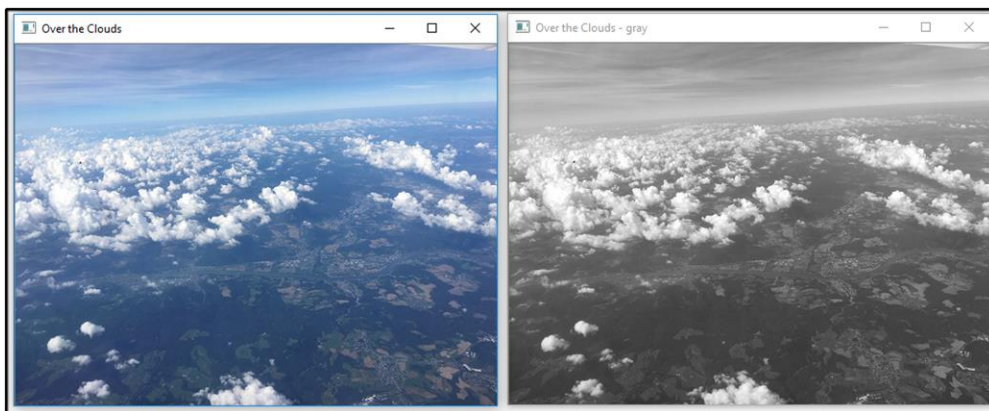
```
public static void cvtColor(Mat src, Mat dst, int type)
```

Листинг 2.1 Метода *cvtColor*

Где је:

- src – улазна слика,
- dst – излазна слика и
- type – тип излазне слике, постоји више вредности, вредност која је коришћена у овом раду је *COLOR_BGR2GRAY*, за претварање *BGR* слике у монохроматску (*grayscale*).

Пример претварања слике из *BGR* формата у монохроматску се види на слици 2.3.



Слика 2.3 Слика у боји и монохроматска слика [13]

2.2 Морфолошке операције

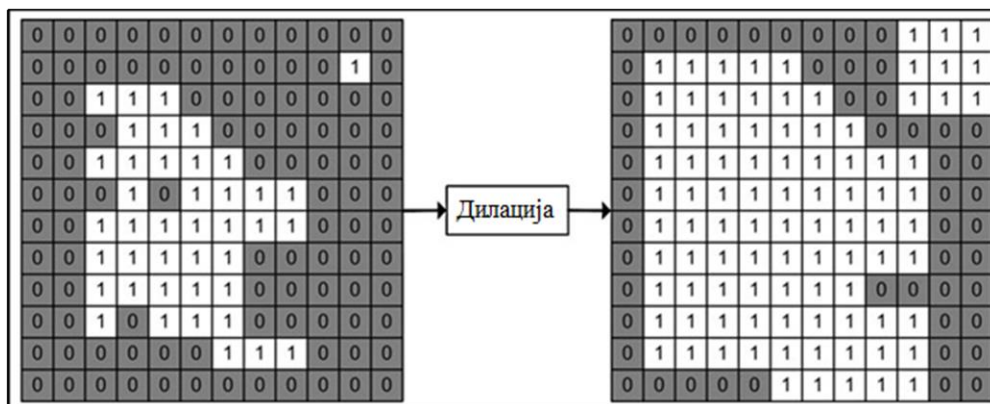
Морфолошке операције над сликом су оне операције које мењају облик и структуру објеката унутар слике. Тако на пример, оштри углови правоугаоника се могу отупити, и мале округле контуре се могу уклонити из слике, а у исто време очувати веће округле контуре.[8]

Користе се за уклањање шума на слици, за изолацију елемената од интереса као и спајање објеката на слици.

Основне морфолошке операције у *OpenCV* библиотеци су дилација и ерозија које се примењују на бинарним сликама.

Обе операције раде тако што упоређују конволуциони кернел наспрам слике која се обрађује. Овај основни елемент може бити правоугаоник (*CV_SHAPE_RECT*), елипса (*CV_SHAPE_ELLIPSE*) или крст (*CV_SHAPE_CROSS*). Основни елемент садржи своју упоришну тачку (*anchor point*) који се често налази у његовом центру. Приликом ерозије основним елементом се пролази преко слике те се посматра околина упоришне тачке. Уколико су сви пиксели у њеној околини који су обухваћени основним елементом 1, тада ће и централни пиксел бити 1, у супротном ће бити 0. Ово доводи до „смањивања“ белих региона на бинарној слици.

Операција супротна ерозији је дилација. Код дилације је довољно да је барем један пиксел 1, да би тренутни пиксел постао 1, у супротном ће бити 0. Ефект дилације се може видети на слици 2.4, где су бели региони улазне матрице проширени. [3, 7]



Слика 2.4 Пример дилације [6]

У овом раду је коришћена дилација.

Имплементација дилације у *OpenCV* бибиотеци:

```
public static void dilate(Mat src, Mat dst, Mat kernel);
```

Листинг 2.2 Метода *dilate*

Где је:

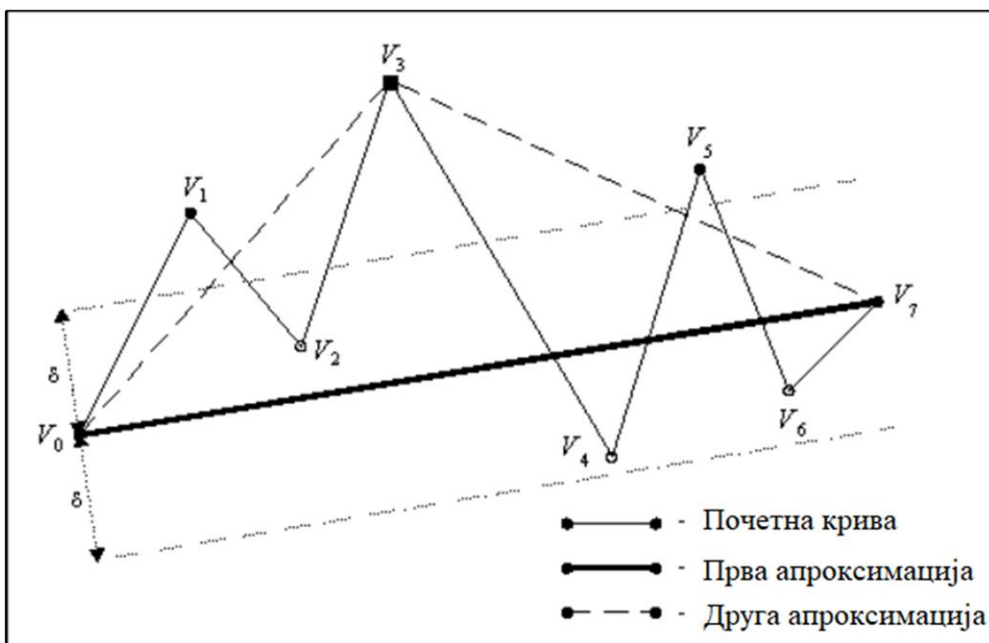
- src – улазна слика,
- dst – излазна слика и
- kernel – структурни елемент

2.3 Поједностављивање контура

Поједностављивање контура је вршено употребом Рамер-Даглас-Пекеровог алгоритма. Основна идеја Рамер-Даглас-Пекер алгоритма је да се на основу датог полигона пронађе сличан полигон са мање тачака.

Алгоритам ради тако што за дату толеранцију чува тачке које су изван те толеранције, а одбацује оне које су унутар. Поједностављена контура се састоји од тачака које су чиниле полазну контуру.

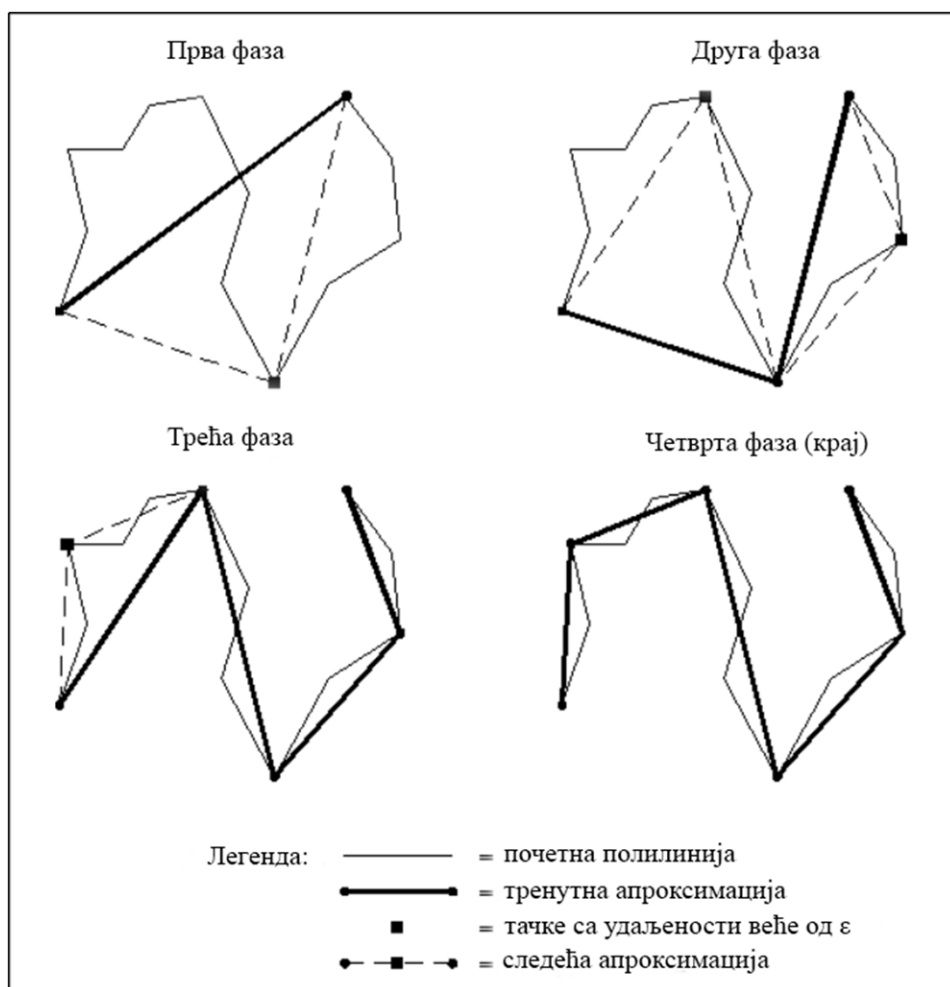
Кроз почетну и крајњу тачку се провлачи права која представља почетну грубу апроксимацију полигона. Потом се рачунају удаљености између преосталих тачака и праве. Ако су све ове удаљености мање од ε , онда је апроксимација добра, крајње тачке се сачувају, док се преостале одбацују. Међутим, ако је било која од ових удаљености већа од ε , онда апроксимација није добра. У овоме случају узима се најудаљенија тачка као нови врх полигона и тиме дели оригиналну апроксимацију на два дела. Ова процедура се понавља рекурзивно за сваку полилинију. Ако су у било ком тренутку удаљености преосталих тачака мање од ε , онда се оне одбацују. Процес се наставља док све тачке нису елиминисане. [5]



Слика 2.5 Две апроксимације Рамер-Даглас-Пекеровог алгоритма [5]

На слици 2.5 могу се видети почетне две апроксимације Рамер-Даглас-Пекеровог алгоритма. Прва апроксимација је увек иста и она представља праву између почетне и крајње тачке полилиније. Најудаљенија тачка од тренутне праве је V_3 и њена удаљеност је већа од ε , те се ова тачка додаје у апроксимацију, делећи почетну полилинију на две праве.

На слици 2.6 се види апроксимације почетне криве у четири фазе:



Слика 2.6 Апроксимација криве у четири фазе [5]

Метода *approxPolyDP* представља *OpenCV* имплементацију Рамер-Даглас-Пекер алгорита. Њена декларација гласи:

```
public static void approxPolyDP(MatOfPoint2f curve,
MatOfPoint2f approxCurve, double epsilon, boolean closed)
```

Листинг 2.3 Метода *approxPolyDP*

Где је:

- *curve* – улазна крива,
- *approxCurve* – излазна крива,
- *epsilon* – дозвољено одступање излазне криве од почетне (у пикселима) и
- *closed* – индикатор да ли је контура затворена или није [3]

2.4 Филтери за ублажавање слике

Филтери за ублажавање слике (*smoothing* или *blurring*) се обично користе за смањење шума на сликама или припрему слике за даљу обраду.

Уобичајени филтери за ублажавање слике су линеарни, при чему се вредности излазних пиксела ($g(i, j)$) израчунавају помоћу тежинске суме улазних вредности пиксела ($f(i + k, j + l)$):

$$g(i, j) = \sum_{k, l} f(i + k, j + l) h(k, l)$$

Функција $h(k, l)$ се назива кернел (*kernel*) и даје вредности тежинских коефицијената филтера. Кернел се може посматрати као прозор који са својим тежинским коефицијентима пролази кроз слику и мења вредности канала у пикселима који су преклопљени његовом чворном (*anchor*) тачком у складу са тим вредностима.

Правоугаони (*box blur*) филтер је линеарни филтер који даје излазну слику у којој је вредност канала сваког пиксела једнака аритметичкој средини вредности свих суседних пиксела кернела. [1]

Функција *blur* омогућава примену праоугаоног филтера на слику. Њена декларација гласи:

```
public static void blur(Mat src, Mat dst, Size ksize)
```

Листинг 2.4 Метода *blur*

Где је:

- *src* – улазна слика,
- *dst* – излазна слика, исте величине и типа попут *src* и
- *ksize* – величина кернела [2]

Слика 2.7 приказује ефект *blur* методе.

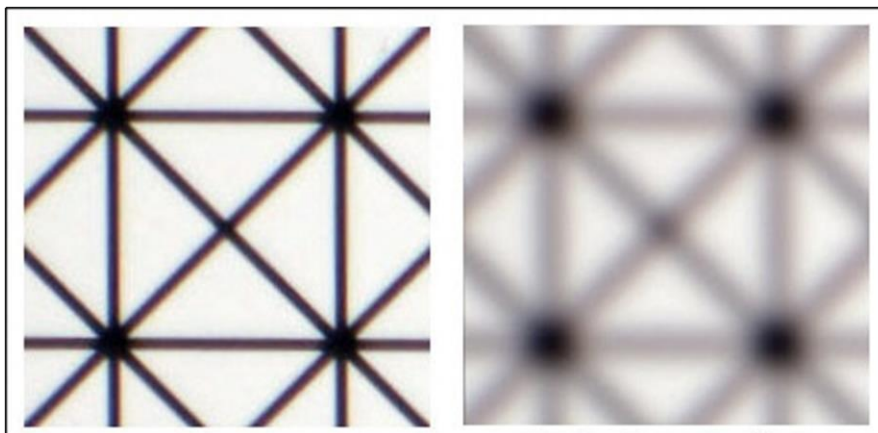


Слика 2.7 Оригинална слика и слика настала применом *blur* методе[2]

2.5 Гаусов филтер

Постоји више филтера за уклањање шума на слици. Гаусов филтер ради на принципу конволуције сваке тачке у улазном низу пиксела са кернелом који узима вредност у складу са Гаусовом расподелом (најчешће унапред израчунатим). Средишњи пиксел Гаусовог кернела има највећу вредност тежинског коефицијента (с обзиром на то да има и највећу вредност у складу са Гаусовом расподелом), док околни пиксели имају све мање вредности тежинских коефицијената како се њихово растојање од централног пиксела повећава. [1]

Слика 2.8 представља поређење оригиналне слике и слике настале применом Гаусовог филтера.



Слика 2.8 Оригинална слика и слика настала применом Гаусовог филтера[2]

Гаусов филтер се на сликама примењује помоћу функције *GaussianBlur* декларисане са:

```
public static void GaussianBlur(Mat src, Mat dst, Size ksize,  
double sigmaX [, double sigmaY])
```

Листинг 2.5 Метода *GaussianBlur*

Параметри *GaussianBlur* функције су:

- *src* – улазна слика. Може садржавати било који број канала који се обрађују независно, али би њихова дубина требала да буде CV_8U, CV_16U, CV_16S, CV_32F или CV_64F,
- *dst* – излазна слика исте величине и типа попут *src*,
- *ksize* – димензије кернела. Висина и ширина се могу разликовати али обе вредности морају бити позитивне и непарне. Уколико су обе вредности иницијализоване на нулу, онда ће се вредност израчунати из сигма параметра,
- *sigmaX* – стандардна девијација Гаусовог кернела у правцу *x*-осе и

- σ_Y – стандардна девијација Гаусовог кернела у правцу у-осе, ако је σ_Y параметар једнак нули, додељује му се вредност σ_X , а ако су σ_X и σ_Y једнаки нули, њихова вредност се рачуна према параметру $ksize$. [3]

2.6 Кенијев детектор ивица

Кенијев детектор користи алгоритам од пет корака за детекцију разних ивица:

1. Примена Гаусовог филтера за уклањање шума
2. Проналажење градијента интензитета осветљености на слици
3. Примена техника не-максималног потискивања (*non-maximum suppression*) како би се ивице слике стањиле и уклонили сви пиксели за које се сматра да нису део ивице
4. Примена дуплог *thresholding* филтера како би се одредиле могуће ивице на слици.
5. Проналажење ивица по принципу хистерезисне криве: финализација детекције ивица која се постиже потискивањем свих осталих ивица које су слабе или нису довољно повезане са јаким ивицама. [1]

Резултат примене Кенијевог детектора је видљиво на слици 2.9.



Слика 2.9 Монохроматска слика и слика настала употребом Кенијевог детектора[12]

Функција *Canny* омогућава примену Кенијевог филтера. Њена декларација гласи:

```
public static void Canny(Mat image, Mat edges, double
threshold1, double threshold2, int apertureSize, boolean
L2gradient)
```

Листинг 2.6 Метода *Canny*

Параметри *Canny* функције су:

- *image* – улазна слика,
- *edges* – излазна матрица ивица; има исту величину као улазна слика *image*,
- *threshold1* – вредност првог прага у хистерезисној процедури (вредности пиксела мање од ове ће бити игнорисане),
- *threshold2* – вредност другог прага у хистерезисној процедури (вредности веће од ове ће се сматрати за ивице). Такође, вршиће се провера да ли има тачака унутар границе *threshold1* – *threshold2* а које додирују ивице. Оне тачке које додирују ивице ће бити додане у коначну матрицу,
- *apertureSize* – величина отвора за Собелов оператор и
- *L2gradient* – вредност која одређује да ли ће се користити прецизнија (*L2*) норма за израчунавање амплитуде градијента

слике (*L2gradient=true*) или је подразумевана норма довољна (*L2gradient=false*) [1, 2]

2.7 Проналажење контура

За проналажење контура на бинарној слици се користи функција *findContours*. Њена декларација гласи:

```
public static void findContours(Mat image, List<MatOfPoint>  
contours, Mat hierarchy, int mode, int method)
```

Листинг 2.7 Метода *findContours*

Параметри *findContours* функције су:

- *image* – улазна 8-битна једноканална слика. Уколико ово није бинарна слика, сви пиксели који имају ненулте вредности ће се третирали као јединице, док пиксели са нулама задржавају своје вредности. Функција мења слику и врши екстракцију контура у исто време,
- *contours* – листа контура које су представљене *MatOfPoint* класом,
- *hierarchy* – вектор који садржи информације о топологији слике. За сваку контуру чува информације о претходној и следећој контури на истом нивоу, прво дете и првог родитеља те контуре,
- *mode* – начин проналажења контура. Могуће вредности:
 - *RETR_CCOMP* – проналази све контуре и организује их у хијерахију на два нивоа (унутрашње и вањске). Дакле, свака контура ће или представљати облик објекта, или облик објекта унутар другог објекта.
 - *RETR_EXTERNAL* – проналази само екстремне спољне контуре,
 - *RETR_LIST* – проналази све контуре без успостављања било каквих хијерархијских односа и

- RETR_TREE – проналази све контуре и реконструише пуну хијерархију угнеждених контура
- method – метод апроксимације контура. Могуће вредности:
 - CHAIN_APPROX_NONE – проналази апсолутно све тачке контуре. То јест, било које две тачке (x_1, y_1) и (x_2, y_2) за које вреди $\max(\text{abs}(x_1 - x_2), \text{abs}(y_2 - y_1)) = 1$.
 - CHAIN_APPROX_SIMPLE – врши компресију хоризонталних, вертикалних и дијагоналних сегмената и оставља само њихове крајње тачке. На пример, правоугаона контура се памти само помоћу 4 тачке и
 - CHAIN_APPROX_TC89, CHAIN_APPROX_TC89_KCOS – користи једну од две верзије алгоритма Тех-Чин ланчане апроксимације. [1, 2]

2.8 Цртање контура

Цртање контура се врши помоћу *drawContours* методе, чија декларација гласи:

```
public static void drawContours(Mat image, List<MatOfPoint>
contours, int contourIdx, Scalar color)
```

Листинг 2.8 Метода *drawContours*

Параметри су:

- image – излазна слика на којој ће бити исцртане контуре,
- contours – листа контура која је добијена позивањем *findContours* методе,
- contourIdx – индекс контуре која се треба исцртати, уколико је вредност негативна, све контуре ће бити нацртане и
- color – боја контуре. [2]

2.9 Одређивање површине контура

Површина контуре се рачуна употребом Гринове формуле[9]:

$$\oint_L P dx + Q dy = \iint_{\sigma} (Q_x - P_y) dx dy$$

Односно:

$$A = \frac{1}{2} \oint_c x dy - y dx$$

Декларација функције која одређује површину контуре се налази у листингу 2.9:

```
public double contourArea(Mat contour)
```

Листинг 2.9 Метода *contourArea*

Где је:

- *contour* – контура чија ће површина бити израчуната

Функција враћа вредност површине. [2]

2.10 Одређивање просечног аритметичког одступања

Мера просечног одступања свих чланова серије од њихове аритметичке средине се добија сабирањем апсолутне вредности одступања свих узорака од њихове аритметичке средине и дељења са њиховим бројем.[10] Односно:

$$\bar{d} = \frac{1}{N} \sum |X_i - \bar{X}|$$

За $i = 1, 2, \dots, n$

Где су:

- X_i – вредност појединог узорка,
- \bar{X} – средња вредност и
- N – број узорака

Декларација функције која одређује стандардну девијацију и просечно аритметичко одступање у *OpenCV* гласи:

```
public static void meanStdDev(Mat src, MatOfDouble mean,  
MatOfDouble stddev)
```

Листинг 2.10 Метода *meanStdDev*

Где је:

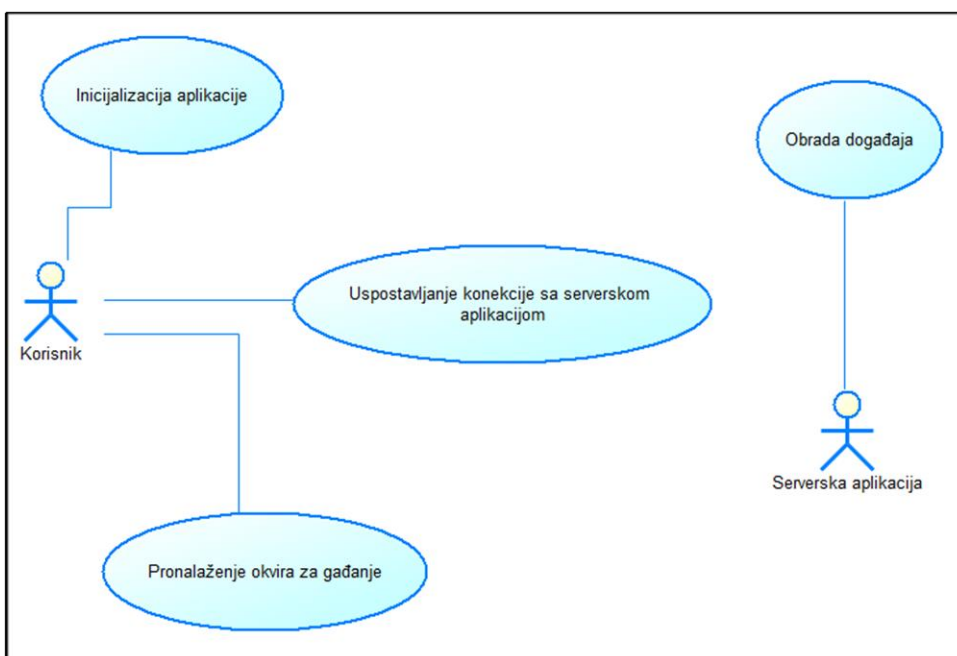
- *src* – улазна матрица,
- *mean* – излазна матрица која ће садржавати апсолутно средње одступање и
- *stddev* – излазна матрица која ће садржавати вредност стандардне девијације [2]

За потребе овог рада коришћено је само апсолутно средње одступање.

3. Спецификација апликације

3.1 Дијаграм случајева коришћења

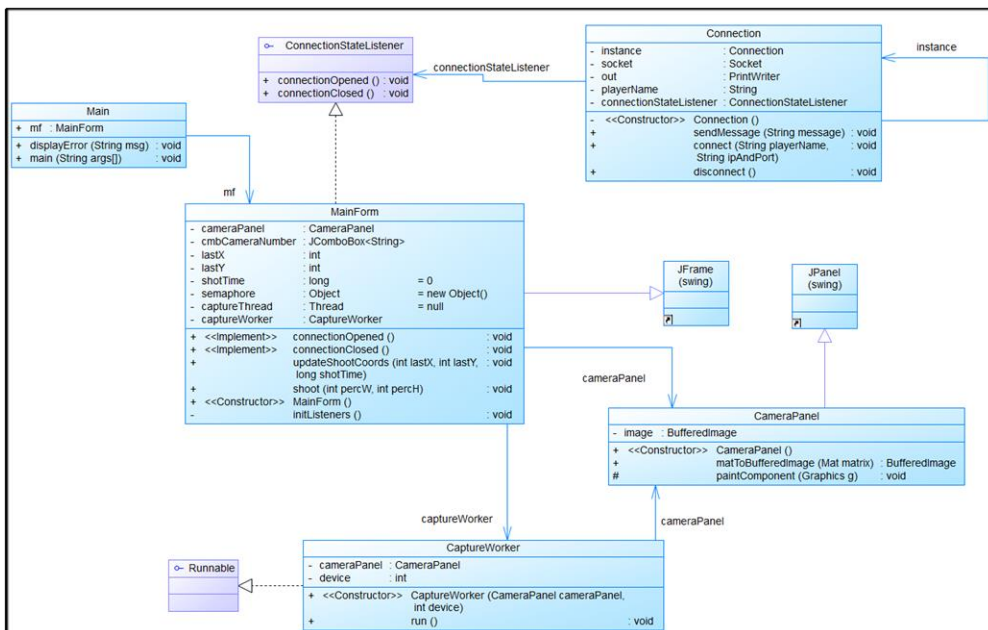
Као што се може видети са слике 3.1 дијаграм случајева коришћења садржи два учесника, корисника и серверску апликацију. Корисник чека иницијализацију апликације, након чега мора отворити конекцију ка серверској апликацији. После овога може пронаћи оквир за гађање. Серверска апликација само чека на догађај који треба обрадити.



Слика 3.1 Дијаграм случајева коришћења

3.2 Дијаграм класа

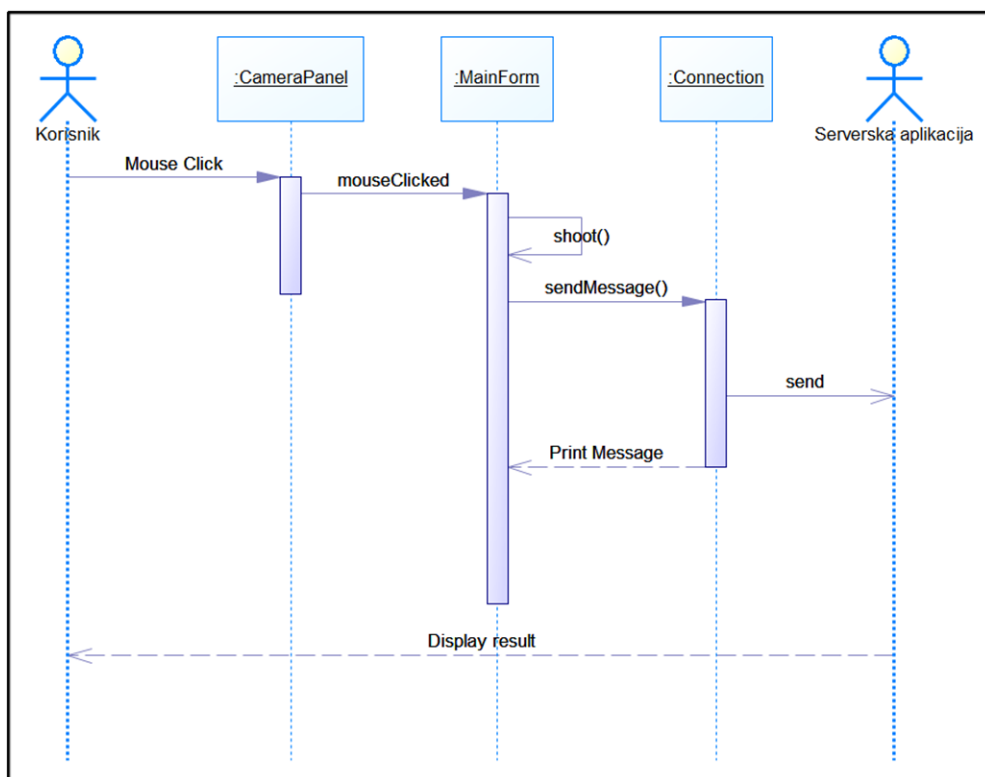
Дијаграм класа је приказан на слици 3.2. *MainForm* класа представља главни прозор апликације. *CaptureWorker* је класа која врши обраду фрејма и сву логику откривања тачака од интереса. Путем класе *Connection* се остварује веза са серверском апликацијом и шаљу поруке. Класа *CameraPanel* приказује коначни резултат морфолошких трансформација.



Слика 3.2 Дијаграм класа

3.3 Дијаграм секвенци

На слици 3.2. је дијаграм секвенци који описује процес пуцања. Класа *CameraPanel* обради корисников клик мишем тако што позове методу *shoot* класе *MainForm*. Надаље, ова метода прима координате пуцња и прослеђује их класи *Connection* методом *sendMessage()*. На овај начин порука *send* је послана серверској апликацији, која ће обрадити догађај. Иста порука ће такође бити исписана на стандардни излаз. Коначан резултат овог пуцња ће бити приказан на екрану чиме ће сервереска апликација обавестити корисника.



Слика 3.2 Дијаграм секвенци

4. Опис имплементације

Апликација је имплементирана у Јава програмском језику (верзија 1.8) уз употребу *OpenCV* библиотеке (верзија 3.4.1).

Помоћу *connect* методе класе *Connection* се остварује конекција са серверском апликацијом као што се може видети у листингу 4.1.

```
/**
 * IP:PORT
 */
public void connect(String playerName, String ipAndPort) {
    try {
        String[] ipPort = ipAndPort.trim().split(":");

        socket = new Socket(ipPort[0],
Integer.parseInt(ipPort[1]));
        out = new PrintWriter(socket.getOutputStream(),
true);

        this.playerName = playerName;

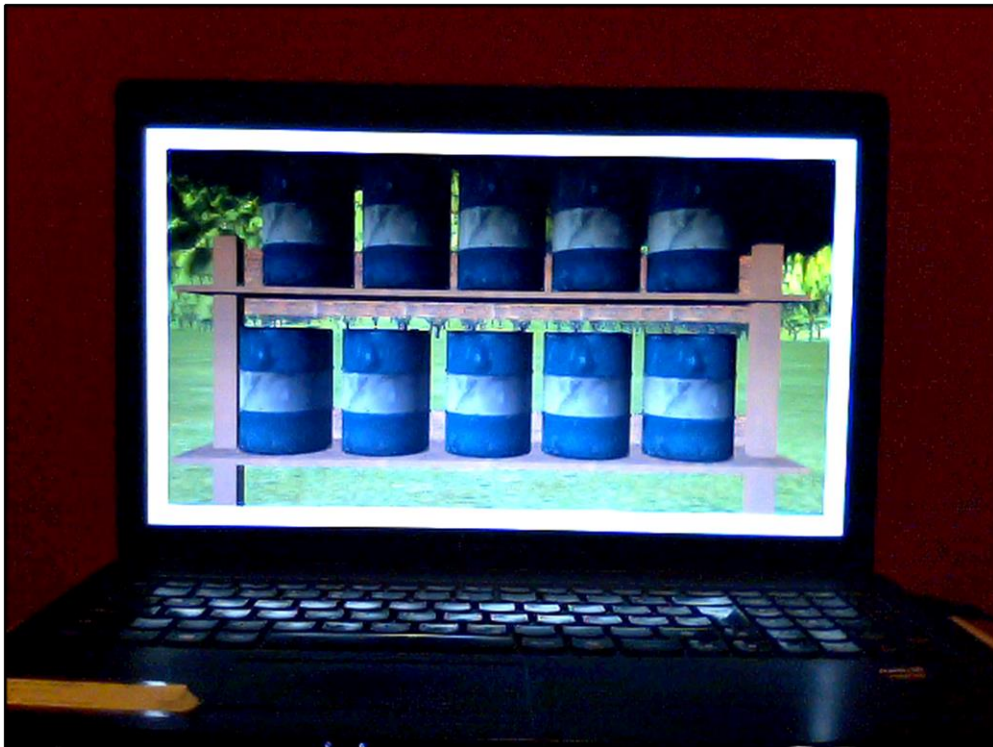
        sendMessage(playerName + "|HELLO");

        if (connectionStateListener != null) {
            connectionStateListener.connectionOpened();
        }

    } catch (Exception e1) {
        disconnect();
        throw new CameraShooterException("Can not connect
to server!", e1);
    }
}
```

Листинг 4.1 Део кода који омогућује спајање са серверском апликацијом

На слици 4.1 приказан је изглед видеа пре морфолошке обраде.



Слика 4.1 Видео пре примене морфолошких операција

Део кода за морфолошке операције је приказан у листингу 4.1.

```

Imgproc.GaussianBlur(copy, copy, new Size(3, 3), 0, 0);
Imgproc.cvtColor(copy, mGray, Imgproc.COLOR_BGR2GRAY);

//Mean, etc.
MatOfDouble mean = new MatOfDouble();
MatOfDouble stdDev = new MatOfDouble();
Core.meanStdDev(mGray, mean, stdDev);

Imgproc.Canny(mGray, mCanny, 0.66*mean.get(0, 0)[0],
1.33*mean.get(0, 0)[0], 3, true);

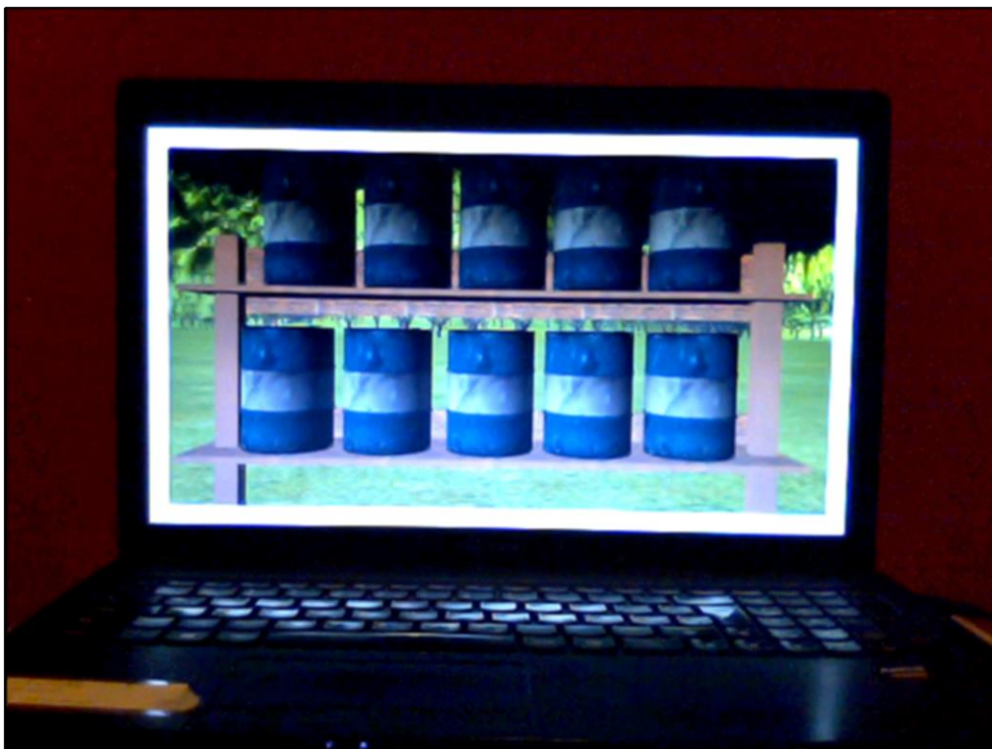
Imgproc.dilate(mCanny, mCanny,
Imgproc.getStructuringElement(Imgproc.MORPH_ELLIPSE,
new Size(5, 5)));

```

```
Imgproc.blur(mCanny, mCanny, new Size(3, 3));
```

Листинг 4.2 Део кода који врши морфолошке операције над сликом

Прво се позива Гаусов филтер за уклањање шума на изворној *BGR* слици, чији резултат је приказан на слици 4.2.



Слика 4.2 Видео пре примене морфолошких операција

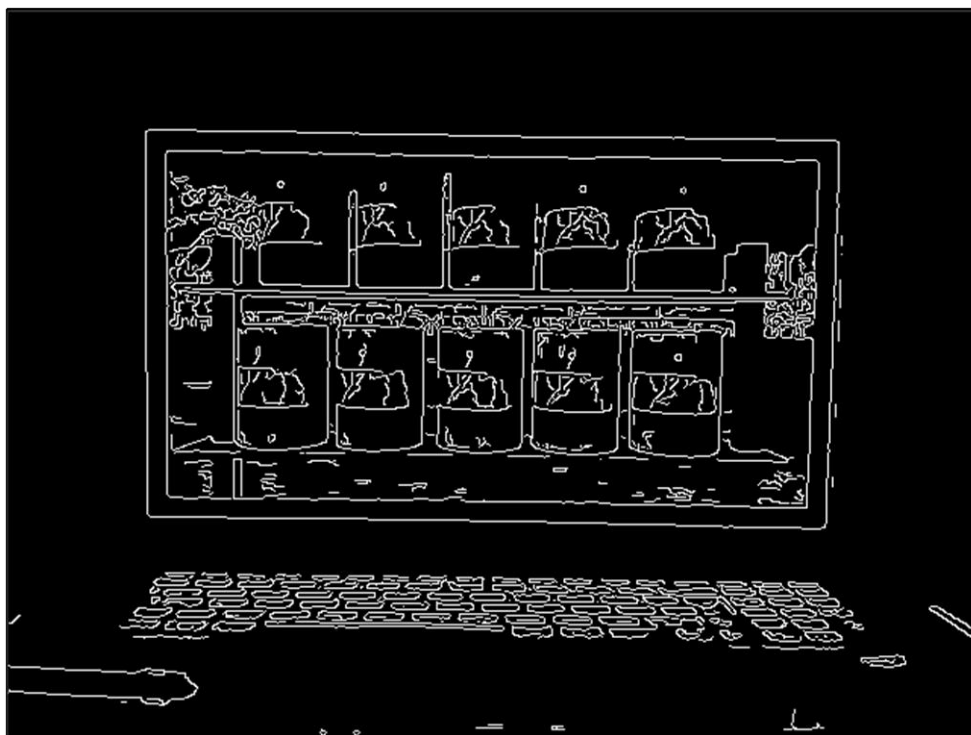
Слика се потом претвара у монохроматску, позивом методе *cvtColor*. Резултат позива наведене методе је видљив на слици 4.3.



Слика 4.3 Монохроматска слика

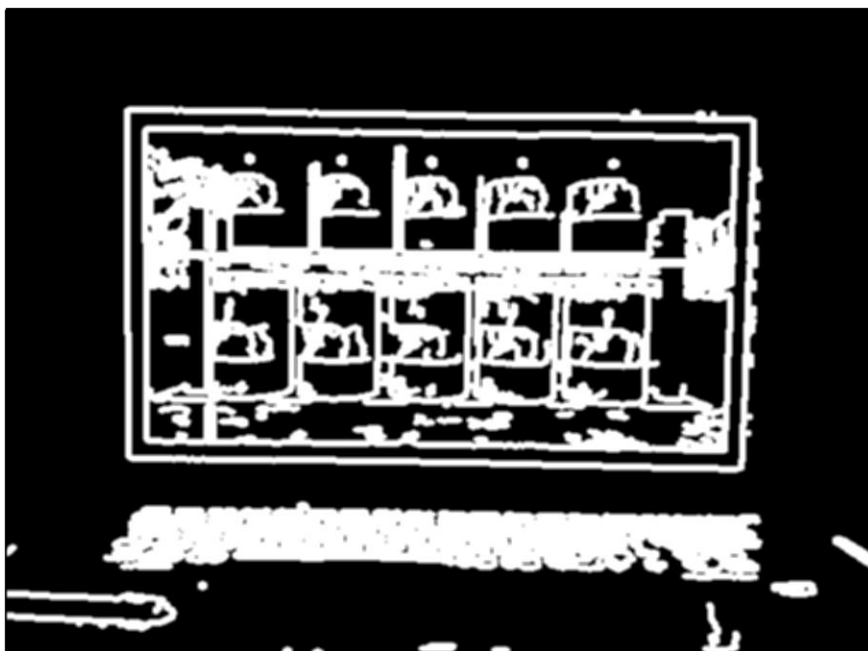
Позивом методе *meanStdDev* одређује се највеће средње одступање у монохроматској слици, које се надаље користи за одређивање доње и горње границе у хистерезисној процедури. Вредност највећег средњег одступања се помножи са 0.66 за добијање доње границе, односно са 1.33 за добијање горње границе за хистерезисну процедуру.

Након тога се одређују ивице у објекту помоћу Кенијевог детектора ивица. Резултат Кенијевог детектора је видљив на слици 4.4.

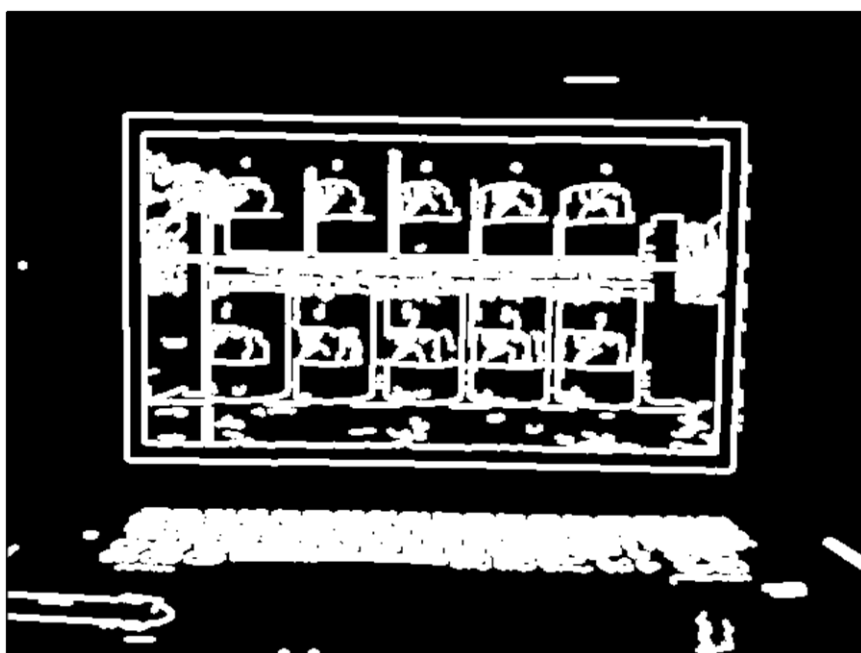


Слика 4.4 Ивице откривене помоћу Кенијевог филтера

Потом се врши дилација и позив *blur* методе на фрејму слике како би се уклонио шум и затвориле контуре. Резултати дилације су видљиви на слици 4.5 док се резултати замућења виде на слици 4.6.



Слика 4.5 Ефекат дилације



Слика 4.6 Резултат *blur* методе

Проналажење највеће контуре је описано у листингу 4.2. Функција пролази кроз све контуре и рачуна њихову површину, при чему у привремену променљиву чува површину и индекс тренутно највеће контуре. Повратна вредност функције је индекс највеће контуре.

```
/**
 * Pronalazi indeks najvece konture
 *
 * @param contours
 * @return
 */
public static int getIndexOfLargest(List<MatOfPoint> contours)
{
    if (contours.size() < 1) {
        return -1;
    }
    double maxArea = 0;
    int index = 0;
    for (int i = 0; i < contours.size(); i++) {
        double area = Imgproc.contourArea(contours.get(i));
        if (maxArea < area) {
            maxArea = area;
            index = i;
        }
    }

    return index;
}
```

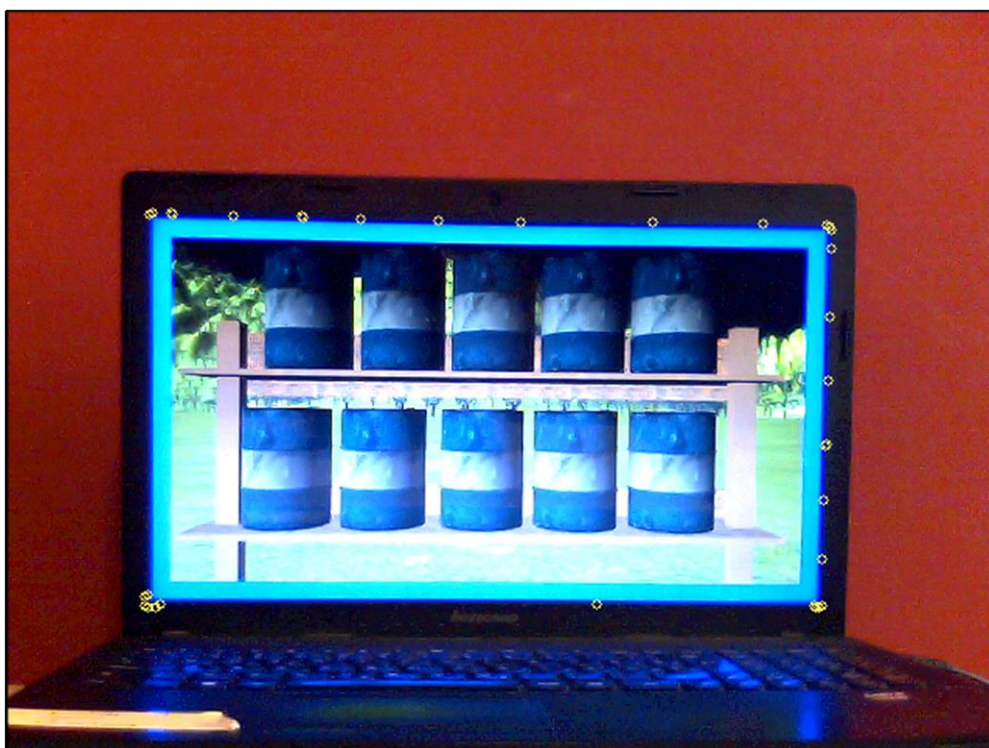
Листинг 4.3 Метода за проналажење највеће контуре

Једном када је пронађена највећа контура, приступа се њеном поједностављивању употребом *approxPolyDP* методе, што је приказано листингом 4.4. Од највеће контуре се прво направи погодан објекат класе *MatOfPoint2f*, као и низ тачака које чине ту контуру. Уколико низ који је добијен од највеће контуре има више од четири тачке, почиње се с позивима *approxPolyDP* методе у *while* петљи. Ова петља ће се извршавати док год је величина низа већа од четири. На тај начин се

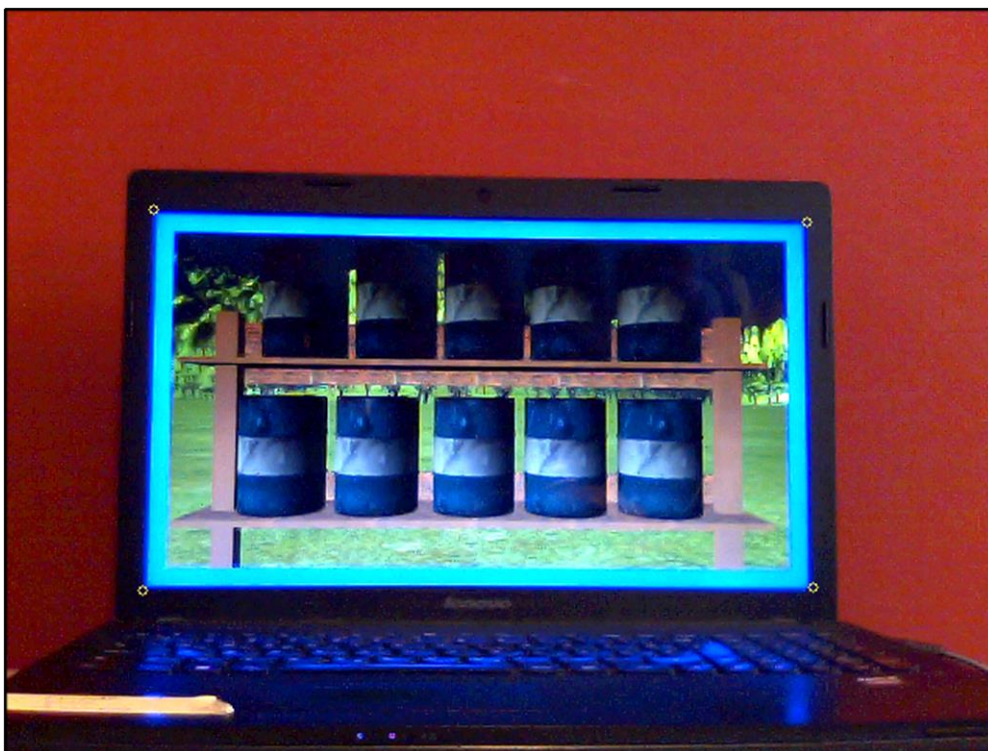
комплексна контура своди на четири тачке од интереса које су потребне за одређивање координата пучња. Број тачака је успешно редукован на овај начин, што се види поредећи слику пре редукције (слика 4.7) и слику после редукције (слика 4.8). На обе слике су означене тачке које чине контуру.

```
MatOfPoint largest = contours.get(index);  
  
MatOfPoint2f NewMtx = new MatOfPoint2f(largest.toArray());  
MatOfPoint2f simplified = new MatOfPoint2f();  
Point[] largestArray = largest.toArray();  
int flag = 1;  
while (largestArray.length > 4) {  
    Imgproc.approxPolyDP(NewMtx, simplified, flag, true);  
    largestArray = simplified.toArray();  
    flag += 2;  
}
```

Листинг 4.4 Употреба *approxPolyDP* методе



Слика 4.7 Тачке које чине контуру (пре редукције)



Слика 4.8 Тачке које чине контуру (након редукције)

Уколико је највећа контура успешно поједностављена на четири тачке, врши се њихово сортирање по x -у, да би се затим могао одредити распоред, односно која тачка је горњи леви, горњи десни, доњи леви и доњи десни врх. Након тога се одређују координате центра (нишана), те се затим позива метода која врши ажурирање координате пуцња (*updateShootCoords*).

```
ArrayList<Point> sortedByX = PointUtils.sortPointsByX(new
ArrayList<>(Arrays.asList(largestArray)));

Point topLeft = (sortedByX.get(0).y < sortedByX.get(1).y) ?
sortedByX.get(0) : sortedByX.get(1);

Point topRight = (sortedByX.get(2).y < sortedByX.get(3).y) ?
sortedByX.get(2) : sortedByX.get(3);

Point bottomLeft = (sortedByX.get(0).y > sortedByX.get(1).y) ?
sortedByX.get(0) : sortedByX.get(1);

Point bottomRight = (sortedByX.get(2).y > sortedByX.get(3).y)
? sortedByX.get(2) : sortedByX.get(3);
```



```

// Nacrtaaj tacke koje su centri kontura
Imgproc.circle(webcam_image, topLeft, 3, RED, 2);
Imgproc.circle(webcam_image, topRight, 3, RED, 2);
Imgproc.circle(webcam_image, bottomLeft, 3, RED, 2);
Imgproc.circle(webcam_image, bottomRight, 3, RED, 2);

// Nacrtaaj linije koje spajaju tacke
Imgproc.line(webcam_image, topLeft, topRight, YELLOW, 2);
Imgproc.line(webcam_image, topLeft, bottomLeft, YELLOW, 2);
Imgproc.line(webcam_image, bottomLeft, bottomRight, YELLOW,
2);
Imgproc.line(webcam_image, bottomRight, topRight, YELLOW, 2);

PointUtils.getPerpendicularPoint(topLeft, topRight,
cameraCenter, perpPointTop);
Imgproc.circle(webcam_image, perpPointTop, 3, RED, 2);

PointUtils.getPerpendicularPoint(topLeft, bottomLeft,
cameraCenter, perpPointLeft);
Imgproc.circle(webcam_image, perpPointLeft, 3, RED, 2);

double topLineLen = PointUtils.pointsDistance(topLeft,
topRight);
double horDist = PointUtils.pointsDistance(topLeft,
perpPointTop);
int shootX = (int) ((horDist / topLineLen) * 10000);

double leftLineLen = PointUtils.pointsDistance(topLeft,
bottomLeft);
double vertDist = PointUtils.pointsDistance(topLeft,
perpPointLeft);
int shootY = (int) ((vertDist / leftLineLen) * 10000);

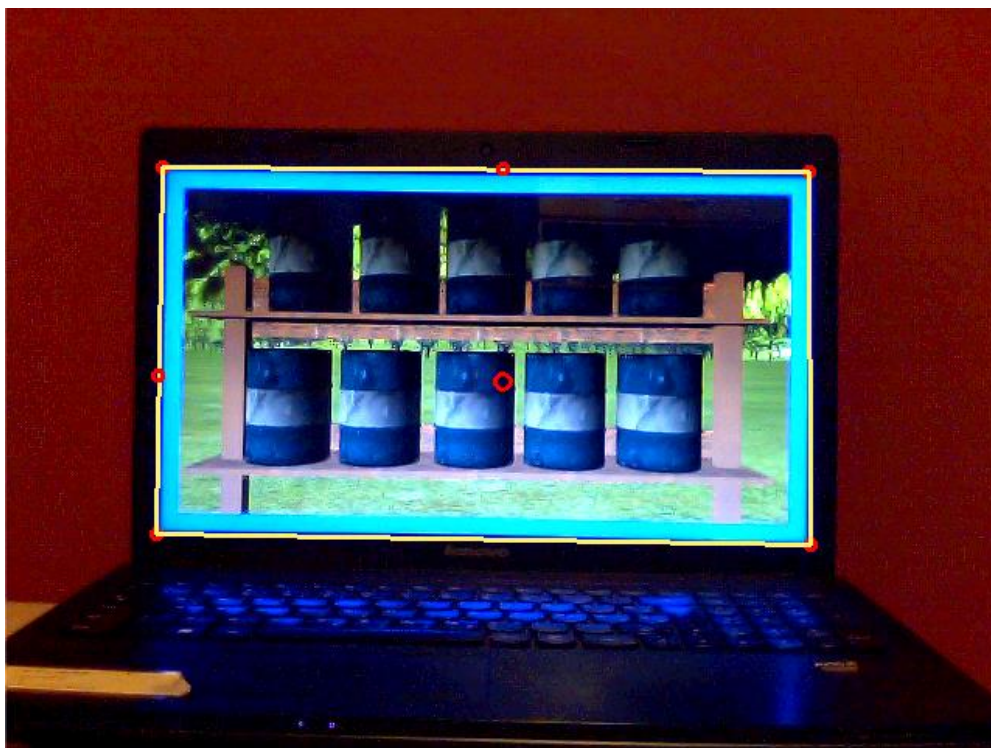
// Pucamo

```

```
Main.mf.updateShootCoords (shootX, shootY,  
System.currentTimeMillis ());
```

Листинг 4.5 Логика за проналажење тачака од интереса и пуцање

Коначни резултат може се видети на слици 4.9. Контура за гађање је успешно одређена.



Слика 4.9 Коначан резултат детекције

5.Закључак

Циљ рада је био реализовати контролер за видео игру који се базира на раду камере. Камера има улогу оружја, где се нишањењем помоћу камере постиже ефекат нишањења оружја. Повлачење ороза је имплементирано као клик мишем. Након детекције клика мишем, програм анализира слику и покушава да одреди где је нишањено. Координате пуцња се шаљу у игру преко мрежне конекције и игра одлучује да ли је постигнут погодак или не. Препознавање оквира се заснива на низу морфолошких операција, те је у потпуности аутоматизовано и више се не ослања на корисникову интеракцију. Ово препознавање најбоље ради у условима када постоји велики контраст између оквира и екрана.

Апликација је развијана у актуелној *OpenCV* библиотеци, тако да постоји добра основа за даљи развој.

ЛИТЕРАТУРА

- [1] Баљозовић Б. „Практикум из дигиталне обраде слике (уз коришћење OpenCV 3 библиотеке)“, Академска мисао, Београд 2017.
- [2] OpenCV
<https://docs.opencv.org/> (приступљено 19.9.2018.)
- [3] Baggio D. L. “OpenCV 3.0 Computer Vision with Java”, Packt Publishing Ltd., Livery Place, 2015
- [4] Visvalingam M. Whyatt J.D. “The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization”
- [5] http://geomalgorithms.com/a16-_decimate-1.html (приступљено 21.9.2018.)
- [6] <http://what-when-how.com/introduction-to-video-and-image-processing/morphology-introduction-to-video-and-image-processing-part-1/> (приступљено 22.9.2018.)
- [7] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html (приступљено 22.9.2018.)
- [8] Weeks A. R., *Fundamentals of Electronic Image Processing*, SPIE Optical Engineering Press, Bellingham, 2004
- [9] Стојаковић М. „Математичка анализа 2“, Факултет техничких наука, Универзитет у Новом Саду, 2017.
- [10] Прокоповић Б. „Математика са статистиком“, Алфаграф-НС, Нови Сад, 2017.
- [11] <http://enginius.tistory.com/505> (приступљено 30.9.2018.)
- [12] <https://www.bogotobogo.com> (приступљено 30.9.2018.)
- [13] <https://solarianprogrammer.com/2016/09/17/install-opencv-3-with-python-3-on-windows/> (приступљено 30.9.2018.)

6.КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	монографска публикација
Тип записа, ТЗ:	текстуални штампани
Врста рада, ВР:	дипломски-бечелор рад
Аутор, АУ:	Никола Стојановић
Ментор, МН:	Проф Др Милан Видаковић
Наслов рада, НР:	Примена <i>OpenCV</i> библиотеке за имплементацију контролера видео игре
Језик публикације, ЈП:	српски
Језик извода, ЈИ:	српски / енглески
Земља публиковања, ЗП:	Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2018
Издавач, ИЗ:	ауторски репринт
Место и адреса, МА:	Нови Сад, Факултет техничких наука,
Физички опис рада, ФО: (поглавља/страна/цитата/табела/слика/графика/прилога)	5 / 35 / 13 / 0 / 21/ 0 / 0
Научна област, НО:	Информатика
Научна дисциплина, НД:	Рачунарске науке
Предметна одредница/Кључне речи, ПО:	Java, OpenCV, Агентске технологије
УДК	

Чува се, ЧУ:		Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад	
Важна напомена, ВН:			
Извод, ИЗ:		Задатак рада представља развој контролера већ постојеће видео игре. Контролер ће бити реализован у Јава програмском језику уз употребу <i>OpenCV</i> библиотеке.	
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:	Др Валентин Пенца, доцент	
	Члан:	Др Синиша Николић, доцент	Потпис ментора
	Члан, ментор:	др Милан Видаковић, ред. проф., ФТН Нови Сад	

7.KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	monographic publication
Type of record, TR :	textual material
Contents code, CC :	BSc thesis
Author, AU :	Nikola Stojanović
Mentor, MN :	Prof Dr Milan Vidaković
Title, TI :	Application of OpenCV library for implementation of a video game controller
Language of text, LT :	Serbian
Language of abstract, LA :	Serbian / English
Country of publication, CP :	Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2018
Publisher, PB :	author's reprint
Publication place, PP :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	5 / 35 / 13 / 0 / 21 / 0 / 0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Science
Subject/Key words, S/KW :	Java, OpenCV, Agent technologies
UC	
Holding data, HD :	Library of the Faculty of Technical Sciences,
Note, N :	

Abstract, AB:		The thesis deals with the implementation of video game controller. Controller is implemented in Java programming language using OpenCV library for real-time computer vision.	
Accepted by the Scientific Board on, ASB:			
Defended on, DE:			
Defended Board, DB:	President:	Valentin Penca, PhD, Assistant professor	
	Member:	Siniša Nikolić, PhD, Assistant professor	Menthor's sign
	Member, Mentor:	Milan Vidaković, PhD, full prof., FTN Novi Sad	

Биографија

Никола Стојановић рођен је 3.10.1995. године у Сомбору. Завршио је основну школу „Др. Фрањо Туђман“ у Белом Манастиру. За време похађања гимназије „Бели Манастир“ је био полазник семинара астрономије у Истраживачкој станици Петница и учесник конференције „Корак у науку“ 2012. године. Гимназију је завршио 2014. године и исте године уписује се на Факултет техничких наука у Новом Саду, одсек Електротехника и рачунарство, смер Рачунарство и аутоматика. Школске 2016/17 опредељује се за област Примењених рачунарских наука и информатике.

