

Факултет инжењерских наука Универзитета у Крагујевцу



Назив студијског програма: Електротехника и рачунарство

Ниво студија: Мастер студије

Предмет: Методе формирања и обраде дигиталне слике

Број индекса: 437/2019

Ненад М. Пантелић

**Препознавање карактеристичних
положаја шаке у циљу контроле
извршавања покретом
Мастер рад**

Комисија за преглед и одбрану:

1. доц. др Маријана Гавриловић Божовић -
ментор

2. _____

3. _____

Датум
одбране: _____

Оцена: _____

У оквиру овог мастер рада кандидат треба да проучи методе које се користе за обраду дигиталне слике, а у циљу издвајања и препознавања објекта од интереса. Рад ће се састојати од теоријског дела, где ће кандидат дати преглед резултата из области и изложити основне кораке неопходне у обради слике ради локализације и издвајања региона са објектом од интереса, као корак ка аутоматском препознавању карактеристичних положаја шаке. Допринос кандидата ће бити и у самостално написаним функцијама које ће вршити поправку квалитета слике и обављати операције неопходне за детекцију и издвајање карактеристичних положаја шаке (папир, камен, маказе).

Препоручена литература:

[1] Миодраг Поповић, Дигитална обрада слике, Академска мисао, 2006. година

[2] R. C. Gonzales, R. E. Woods, Digital Image Processing, Second edition, Prentice Hall, 2002. година

Крагујевац, 12.03.2020.

Ментор:

др Маријана Гавриловић Божовић, доцент

ПРИЛОГ 6: Пријава мастер рада

	Универзитет у Крагујевцу Факултет инжењерских наука	 
--	--	---

Мастер студије: **Електротехника и рачунарство**

Студијско подручје (усмерење): /

Име и презиме: **Ненад Пантелић**

Број индекса: 437/2019

ПРИЈАВА МАСТЕР РАДА

Тема рада:

Препознавање карактеристичних положаја шаке у циљу
контроле извршавања покретом

Задатак: У оквиру мастер рада кандидат ће проучавати методе које се користе за обраду дигиталне слике, а у циљу издвајања и препознавања објеката од интереса. Рад ће се састојати од теоријског дела, где ће кандидат дати преглед резултата из области и изложити основне кораке неопходне у обради слике ради локализације и издвајања региона са објектом од интереса, као корак ка аутоматском препознавању карактеристичних положаја шаке. Допринос кандидата ће бити и у самостално написаним функцијама које ће вршити поправку квалитета слике и обављати операције неопходне за детекцију и издвајање карактеристичних положаја шаке (папир, камен, маказе).

Ментор:

Крагујевац 12.03.2020.

др Маријана Гавриловић Божовић, доцент

Маријана Гавриловић Божовић
ФАКУЛТЕТ ИНЖЕЊЕРСКИХ НАУКА
КРАГУЈЕВАЦ

Садржај

1	Увод	6
1.1	Положаји шаке на сликама	8
2	Препознавање карактеристичних положаја шаке са слике	10
2.1	Учитавање слике и прилагођавање димензија	11
2.2	Претпроцесирање слике	12
2.3	Проналазак контуре и параметара контуре	16
2.4	Одређивање корисних тачака шаке	20
2.4.1	Одређивање корене тачке шаке	20
2.4.2	Одређивање блиских тачака (кластера) тачака	21
2.4.3	Одређивање центроиде кластера	23
2.4.4	Одређивање долина између прстију	25
2.4.5	Уклањање непотребних тачака - лажних прстију	25
2.5	Одређивање положаја који шака формира	30
2.6	Резултати алгоритма	34
3	Контрола извршавања покретом	47
3.1	Снимање позадине	48
3.2	Обрада фрејма	50
3.3	Детекција шаке и одређивање карактеристичног положаја	52
3.4	Контрола игре на основу положаја шаке	58
4	Закључак	61
	Литература	62

Abstract

The main purpose of this thesis is to develop a program for **palm localization** in images, as well as recognition and distinguishing the characteristic **gestures** of the palm. By default, the palm can form the shape of **paper**, **rock** and **scissors** from the so-called game. The algorithm that determines which gesture of the hand is formed is based primarily on contour recognition, but also on geometric estimates. Another goal of the project work is the implementation of **automated movement control** based on the recognized gesture of the hand. In this case, it is assumed that the recognition is performed based on the video, which brings a new dimension of the problem. The algorithm used for these purposes is based on the segmentation of an object of interest, that is, distinguishing the hand from the background.

Keywords: image, palm, palm gesture recognition, paper-rock-scissors, geometric algorithms, movement control, a region of interest, object segmentation.

Резиме

Основни циљ овог мастер рада је развој програма за **локализацију шаке** на сликама, као и препознавање, односно разликовање карактеристичних **положаја шаке** на истим. Подразумевано, шака може формирати положај **папира**, **камена** или **маказа** из истоимене игре. Алгоритам који одређује који положај шака формира се првенствено заснива на препознавању контура, али и геометријским проценама конвексности шаке, дужина прстију и неких карактеристичних димензија. Други циљ пројектног рада је имплементација **аутоматизоване контроле покрета** на основу препознатог положаја шаке. У овом случају се подразумева да се препознавање врши на основу видео снимка, што представља нову димензију проблема. Алгоритам коришћен у ове сврхе се заснива на сегментацији објекта од значаја, односно разликовању шаке од позадине. **Кључне речи:** слика, шака, препознавање положаја шаке, папир-камен-маказе, геометријски алгоритми, контрола покрета, објекат од интереса, сегментација објекта.

1 Увод

Проблеми везани за сегментацију објеката од интереса на сликама и видео снимцима су веома популарни данас. Продор дисциплина попут рачунарске визије и неких области вештачке интелигенције, као што су машинско учење и дубоко учење допринео је развоју сложених апликација и алата намењених решавању проблема препознавања објеката са слике као и одређивању неких њихових карактеристика - локација, боја, облик, димензије, тип...

Слика као податак се математички моделира као матрица, где сваки елемент у матрици носи информацију од значаја - пре свега боју. Квалитет слике са којом се ради значајно зависи од параметара попут квалитета уређаја који је коришћен при њеном снимању, осветљености, угла под којим се неки објекат снима, вештина фотографа или сниматеља. Сви ови параметри у великој мери одређују колико је слика као податак корисна.

Често долазимо у ситуацију да слике које користимо нису најбољег квалитета, па их је потребно обрадити пре даље употребе, а све то у циљу отклањања шума и поправљања контраста слике. Овде се пре свега мисли на методе филтрирања, морфолошке операције, одређивање хистограма и других карактеристика интензитета пиксела [1, 2].

Оно што представља нови ниво изазова у односу на препознавање објеката на слици је препознавање и праћење објеката на видео записима. Осим проблема динамике кретања објекта (промена положаја, угла, облика), проблем представља и извршење у реалном времену, где је потребно користи ефикасне алгоритме који су захвални по питању употребе рачунарских ресурса (процесорског времена, меморије и интеракције са диском). С обзиром да то често и није могуће, прибегава се коришћењу хардверски напреднијих архитектура, попут графичких и векторских процесора, мултирачунара (кластера) и сл.

Данас, рачунарска визија представља једну од дисциплина са којом је дубоко учење у тесној спрези. Интелигентни системи за подршку одлучивању играју главну улогу у локализацији и препознавању објеката са слике или видео записа, а све у циљу опонашања лудске способности, односно имитирању спреге визуалног и нервног система код човека. Неки од популарних проблема данашњице у којима обрада слике и рачунарска визија играју кључну улогу су: препознавања лица и емоција на лицу, детекција биомаса на снимцима магнетне резонанце, СТ скенера, ултразвучним снимцима, обрада сателитских снимака, видео надзор, означавање личности на сликама са друштвених мрежа, аутоматизована вожња и сл [2].

Концепти изнесени у овом раду се пре свега базирају на обради слике и геометријским проценама у циљу сегментације објекта од интереса, односно проналажењу карактеристичних својстава шаке како би се одредио положај који шака формира. Друга димензија решења подразумевала би коришћење неких алгоритама машинског и дубоког учења.

Пројекат је реализован у програмском језику Python, уз употребу модула *OpenCV* и *numpy*. Пројекат је конципиран као пројекат отвореног кода (*open-source*) без лиценце и нема ограничења у његовом коришћењу, доради и преправци.

1.1 Положаји шаке на сликама

Као што је већ наглашено, основни циљ овог рада је развој апликације (програма) за препознавање карактеристичних положаја шаке и то положаја папира, камена и маказа, по узору на истоимену игру. Основни скуп података који је коришћен при развоју и тестирању саме апликације састоји се од 39 слика (по 13 слика за сваки положај). Сlike су снимљене мобилним телефоном и њихова резолуција је прилагођена на 600x800 пиксела. Поред ових слика коришћене су и друге слике, углавном преузете са интернета.

Оно што је још битно навести је како изгледају стандардни облици положаја које тражимо. На слици 1 дат је приказ стандардних положаја шаке при формирању ових знакова.



Слика 1: Пример стандардних положаја шаке при формирању знака папира, камена и маказа (слева на десно).¹

Претпоставка је да се на свакој слици налази шака која формира неки од наведених положаја. У идеалном случају шака је вертикално постављена тако да

¹слика преузета са адресе: https://images-na.ssl-images-amazon.com/images/I/71VqHrKuHWL._AC_SL1500_.jpg

јој се прсти налазе у горњем делу слике, а почетак шаке у доњем, тако да је шака паралелна апсцисној оси, као што можемо видети на примеру формираног положаја папира на слици 2.



Слика 2: Пример идеалног положаја шаке (формиран папир) - слика из тест скупа

Свакако да је осим идеалног, програм тестиран и на положајима шаке који представљају неки вид варијације датих идеалних положаја - издвојени прсти (палац пре свега), шака постављена под оштрим углом, шака постављена хоризонтално... За различите положаје је потребно вршити одређене корекције параметара, пре свега оних који су везани за углове и растојања појединих тачака, о чему ће више бити речи у наставку рада.

2 Препознавање карактеристичних положаја шаке са слике

Укратко, ако се поступак одређивања карактеристичних положаја шаке подели у три фазе, добијају се следећи кораци:

1. лоцирати објекат од интереса, у овом случају шаку;
2. пронаћи прсте и корисне параметре који их одређују;
3. одредити који положај је формиран;

Да би објекат од интереса био лоциран, потребно је пронаћи шаку на слици, односно пронаћи контуру шаке. Пре свега, треба увести претпоставку да је шака доминантан (највећи) објекат на слици. У супротном, алгоритам може да не да жељене резултате. Даље, подразумева се да је потребна само спољна контура шаке, без унутрашњих контура које могу да праве проблем, па се оне идентификују као шум. Под унутрашњим контурама подразумевају се контуре ноктију, бразда и ожиљака на руци, длаке, младежи и друге кожане израслине.

Следећи корак подразумева проналазак прстију и других параметара који их одређују. Људска шака без аномалија има пет прстију тако да је подразумевано средњи прст најдужи. Према томе, треба водити рачуна о елиминисању “лажних прстију”. Такође, оно што је корисно при одређивању положаја који шака формира је постојање удубљења између прстију која се формирају када су прсти растављени. У даљем тексту, ова удубљења ће бити називана “долинама”.

Трећи корак представља одређивање положаја који шака формира. У овом делу кључну улогу играју карактеристике шаке, односно прстију. Њихова дужина, растојање између прстију, као и карактеристике самих положаја: да ли постоје долине између прстију, колико прстију формира неки положај, колика је разлика у висини између највишег (најдужег) учљивог прста и оног најнижег (најкраћег) и сл.

Према томе, поступак одређивања положаја шаке је нешто компликованији, па ће у даљем раду детаљније бити објашњен у више корака и то:

1. учитавање слике и прилагођавање димензија слике
2. претпроцесирање слике како би се отклонио шум и слика трансформисала у формат погодан за наредне кораке алгоритма
3. проналазак контуре и значајних параметара контуре са слике
4. одређивање корисних тачака објекта од интереса
5. одређивање положаја који шака формира
6. приказ добијених резултата

2.1 Учитавање слике и прилагођавање димензија

Пућања улазне слике се задаје као параметар командне линије, као и већина подесивих параметара. Слика се учитава каква јесте, без иницијалне модификације.

Димензије слике је могуће прилагодити кроз два параметра:

- **takeUpperHalf** - логички параметар који дефинише да ли треба узети само горњу половину слике. Уколико је овај параметар тачан, подразумевано се узимају у обзир прве две трећине слике, али је то свакако могуће променити уносом из командне линије. Овај параметар је користан када је висина задате слике велика и представља проблем приказа исте. Такође, овај параметар у неким случајевима може утицати и на квалитет проналаска неких тачака од интереса.
- **rotationAngle** - нумерички параметар који дефинише угао ротације слике. Шака на слици може бити ротирана за -90 или 90 степени у односу на подразумевани положај што представља проблем у даљем одређивању карактеристичних тачака и углова. Из тог разлога је неопходно ротирати улазну слику тако да шака буде усправно постављена, заузимајући тако идеалан положај. При ротацији слике за -90 или 90 степени, мењају се и ширина и висина слике (мењају места), па је потребно извршити одсецање делова слике који су празни (математички гледано, с обзиром да је иницијално ширина мања од висине, услед транспоновања матрице, добијају се нула редови на почетку и на крају слике) [3]. Битно је напоменути да овај параметар има смисла само ако му је уопштена вредност облика $k\frac{\pi}{2}$, $k \in \mathbb{Z}$.

2.2 Претпроцесирање слике

Опис поступака обраде слике биће објашњен на примеру слике 2 из уводног дела овог рада. Поступци обраде слике, филтрирање и уклањање непотребних детаља су шематски приказани на слици 3.



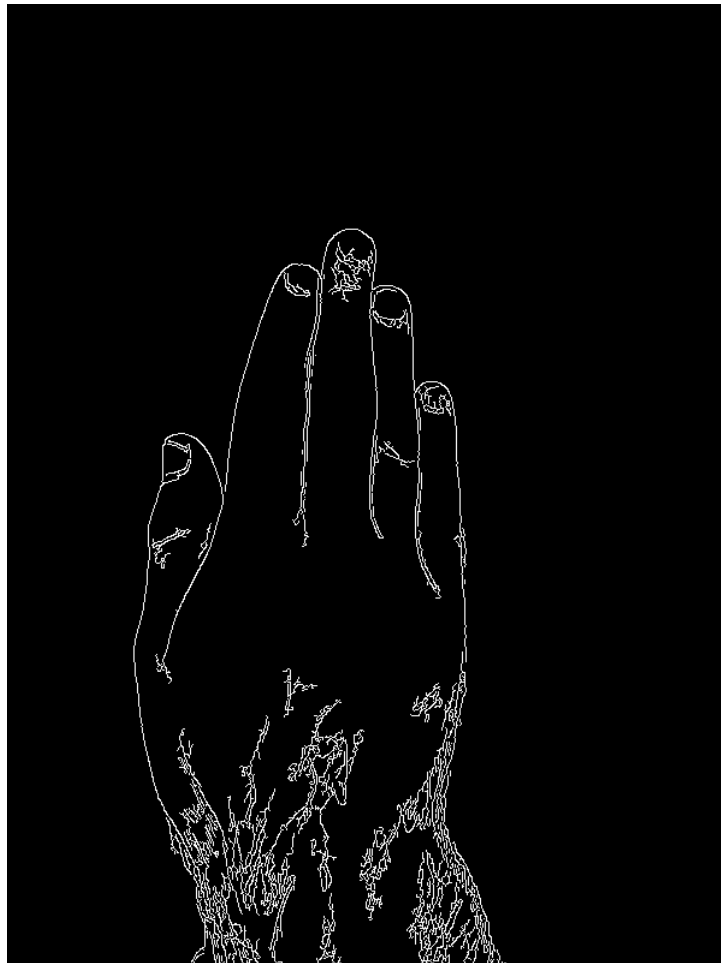
Слика 3: Ток претпроцесирања слике

Након учитавања слике и извршења евентуалних модификација у димензијама слике, слика се преводи у **grayscale** формат. Овај колор модел (енг. *color space*) одликује то да је сваки пиксел описан једним бајтом, односно интензитет сваког пиксела се креће у опсегу $[0, 255]$. Овакве слике се називају и *single-channel* слике. Рад са сликама у RGB (*red-green-blue*) колор моделу је углавном компликован, прави проблеме при детекцији ивица и контура, а уз то је и рачунски захтевнији, па се стога слика преводи у grayscale модел. Након конверзије колор модела добија се слика 4 [1, 2, 4].



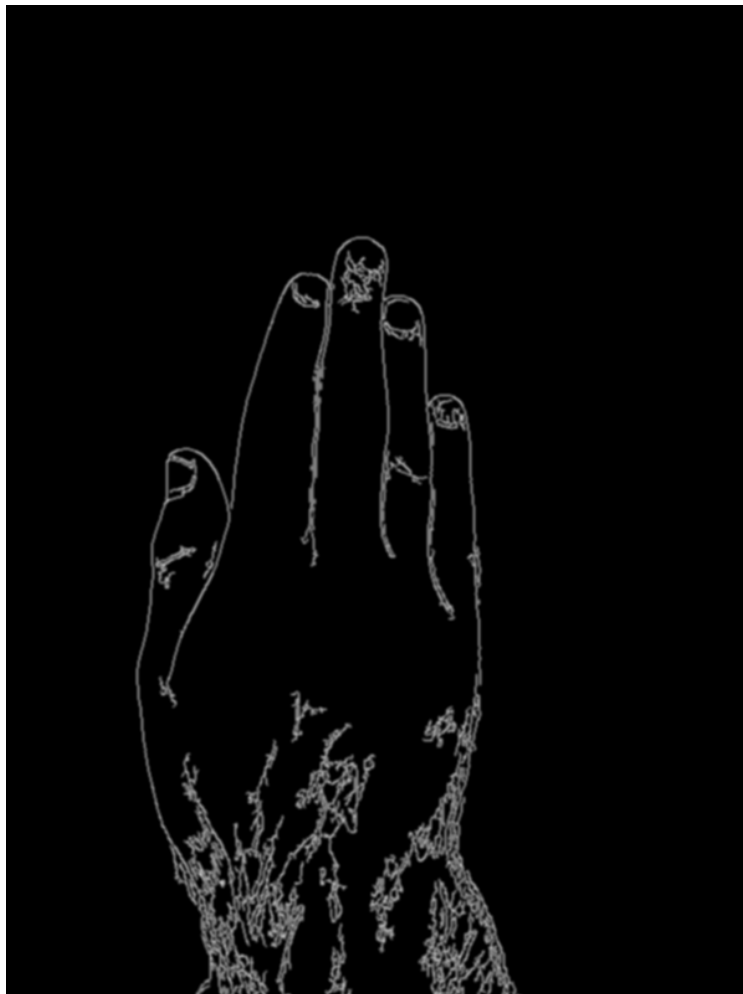
Слика 4: Улазна слика конвертована у grayscale формат

Следећи корак је детекција ивица. За детекцију ивица у овом раду је коришћен **Canny edge** алгоритам који представља један од најпопуларнијих алгоритама за ову намену, развијен средином 80-их година прошлог века. Алгоритам се заснива на методама одређивања локалних максимума суседства за сваки пиксел, коришћењем градијентне методе, а затим утврђивањем прагова помоћу којих се потврђују ивични пиксели, односно одбацују они који то нису. Ови прагови се могу подесити у оквиру апликације као аргументи који се задају из командне линије. Након детекције ивица добија се слика 5 [5].



Слика 5: Детектоване ивице

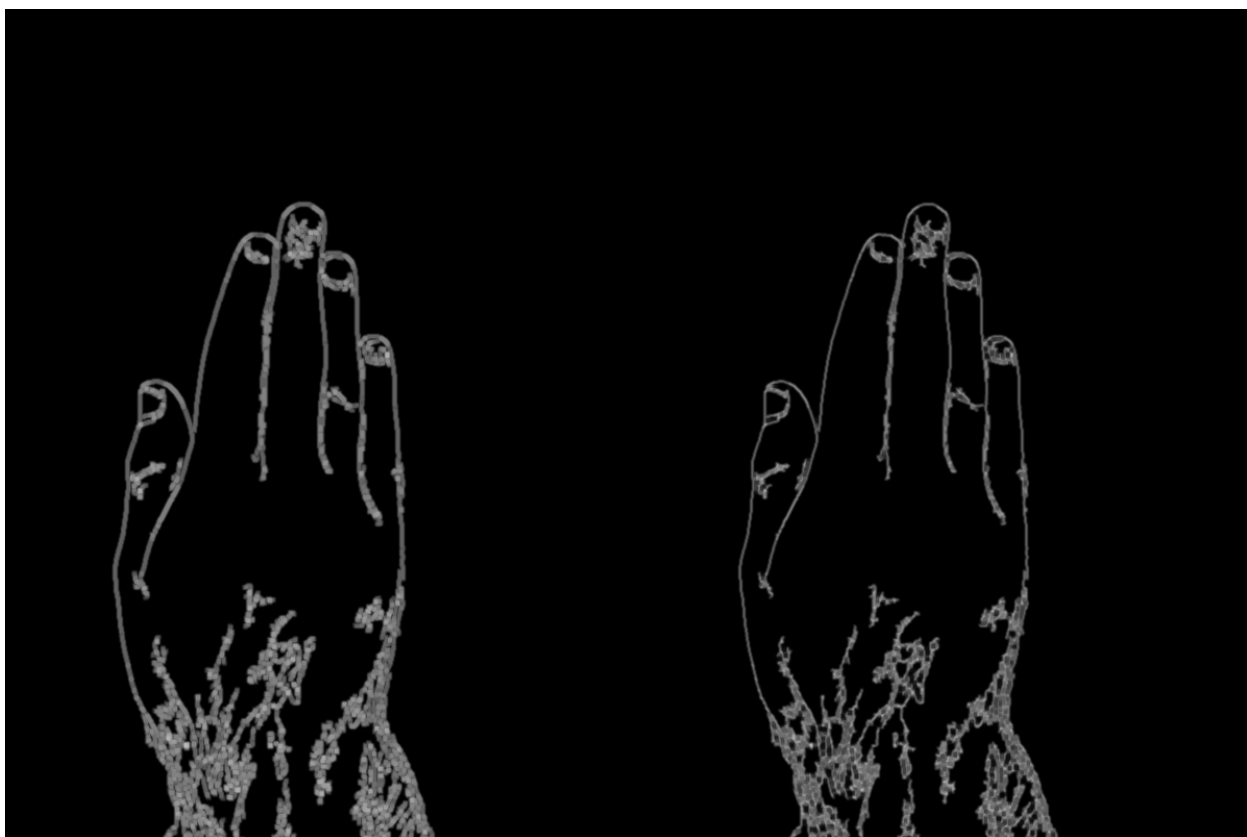
Као што се може видети са слике 5, уз спољну ивицу имамо и унутрашње ивице - нокти, прегини прстију, длаке... Из тог разлога је потребно исфилтрирати пронађене ивице. Прво ћемо применити Гаусов филтар. Идеја Гаусовог филтра је просторно усредњавање интензитета пиксела применом Гаусове функције, тако што за сваки пиксел посматра њему суседне пикселе уоквирене кернелом филтра. Овај процес се још назива и “углачавање” или “замућивање”, јер слика стварно бива замућена услед смањења контраста, односно замућења ивица [6].



Слика 6: Резултат примене Гаусовог филтра

Када имамо слику на којој постоје “шупљине” у ивицама, тачкасти шум (со и бибер шум), повезаност два објекта танким ивичним елементима услед шума или сенки и сл. прибегавамо примени морфолошких операција. Под морфолошким операцијама се подразумевају ерозија и дилатација и операције у којима ове две операције учествују (отварање и затварање). Ерозија, аналогно истоименом геолошком процесу по ком је и добила назив, представља “спирање” ивичних пиксела. Упростијено, ова операција се заснива на конволуцији употребом кернела. Ако на пример, примењујемо ову операцију на бинарној слици, кернел се креће кроз слику и уколико сви пиксели под кернелом имају вредност 1, оригинални централни пиксел из кернела бива сачуван (не филтрира се). У супротном пиксел се одбацује, односно његова вредност се поставља на 0. На тај начин се ивица стањује, односно поједини ивични пиксели, зависно од величине кернела се губе. С друге стране, операција дилатације спроводи конволуцију, с тим што се централни пиксел под кернелом чува уколико бар један пиксел има вредност 1. На тај начин се ивице подебљавају (попуњавају уколико постоје

шупљине у њима). Примена ерозије, а затим дилатације се још назива операцијом отварања. Код операције затварања, редослед је обрнут, прво се примењује дилатација, па ерозија [1, 2, 7]. Ако на слици добијеној у претходном кораку применимо операцију дилатације, па ерозије (затварање), добијамо резултат са слике 7.



(а) након примене дилатације

(б) након примене ерозије

Слика 7: Резултат примене операције затварања

Филтрирање и морфолошке операције су готово стандардни поступци претпроцесирања који претходе поступцима проналаска контура или ивица на слици. Филтрирање је нарочито корисно у ситуацијама када је потребно извршити би-наризацију слике због ефекта усредњавања, односно поправке контраста. У ивицама које имају шупљине или су растављене, услед операције затварања формирају се спојене и истањене ивице што касније резултује проналаском спојених контура. Према томе, поједине ивице бивају исфилтриране, а спољна ивица јасно дефинисана и њена конвексности је израженија. С обзиром да у следећем кораку, програм проналази контуре (пре свега ону спољну), без ове операције, контура би била испрекидана на појединим местима и при спајању крајњих конвексних тачака би се добила маска која не покрива целу шаку.

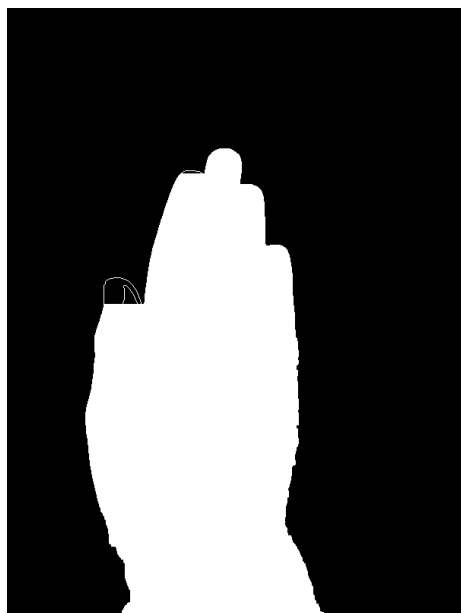
2.3 Проналазак контуре и параметара контуре

Након обраде слике потребно је локализовати област шаке. Под тим се подразумева проналазак спољне контуре шаке, чиме дефинишемо област шаке. Контуре можемо дефинисати као границе неког облика са истим интензитетом. На слици 8, дат је приказ свих пронађених контура на показној слици. За проналазак контура у раду је коришћена функција **findContours** из библиотеке **OpenCV** [8].

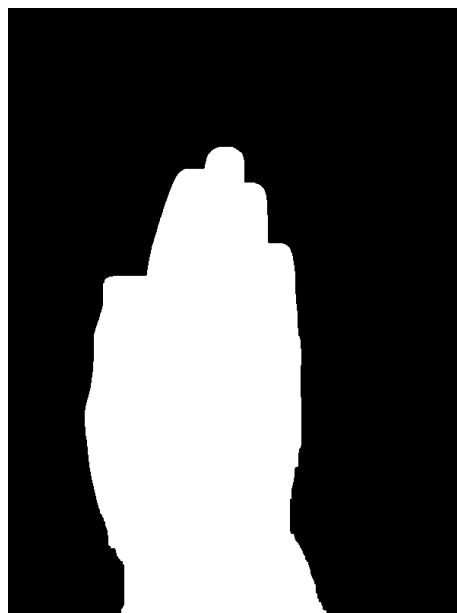


Слика 8: Контуре шаке

Видимо да осим спољне контуре која нас интересује, постоје и неке унутрашње контуре које нису од користи. Како би ово решили, могуће је одабрати највећу контуру, чиме добијамо контуру која окружује спољашњост шаке. Друга могућност коју имамо је превођење слике у бинарни облик како би шаку дефинисали као белу масу на црној позадини - с обзиром да је шака (њена унутрашњост) у том случају бела, не постоји могућност да унутрашње контуре буду пронађене. Превођење слике у бинарни облик подразумева да на потпуно црној слици обележимо крајње тачке шаке, које извучемо из свих пронађених контура, а затим формирамо конвексни полигон који обојимо у бело (слика 9а).



(а) конвексни полигон шаке



(б) бинарна маска шаке

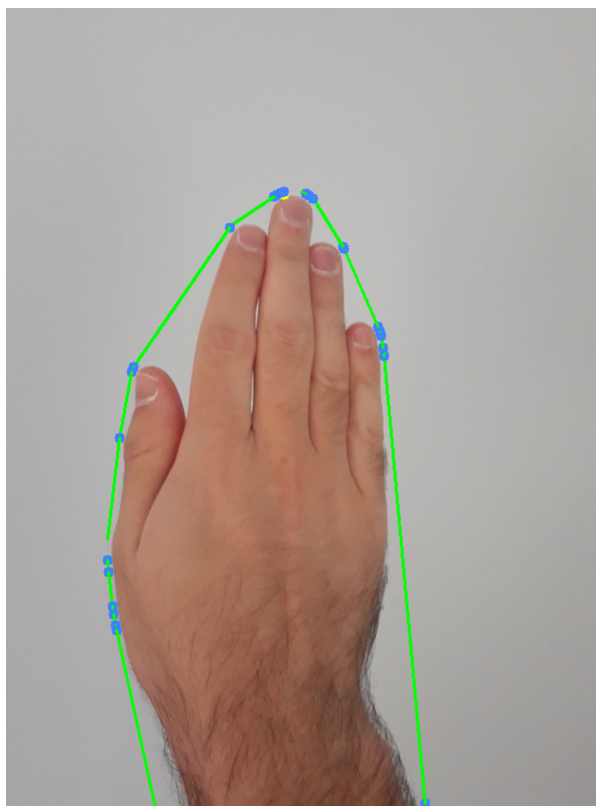
Слика 9: Бинаризација шаке

С обзиром да видимо да бела маса не покрива у потпуности шаку, потребно је морфолошким операцијама (затварањем пре свега), додатно процесирати слику (слика 9б). Након тога се спољна контура проналази врло лако, што се може видети са слике 10.



Слика 10: Спољна контура шаке

Конвексност контуре је једна од карактеристика која нас интересује. Посматрајући геометријске карактеристике шаке лако долазимо да закључка да постоје области где постоје **конвексни дефекти** (конкавни предели контуре), попут облина прстију, долина између прстију, заобљених делова шаке и сл. Сваки дефект можемо дефинисати са три тачке, две крајње тачке где дефект почиње и завршава се, као и оном тачком где је дефект најизраженији. Примера ради, код долина између прстију, две крајње тачке би се налазиле на врховима прстију док би се трећа тачка налазила у самој долини (најдубља тачка у долини). Уз то, свака контура је описана низом (листом) тачака, којих може бити више или мање (зависно од параметара које смо проследили функцији која тражи контуре). Претпоставка је да врхови прстију, без обзира да ли су испружени или не, формирају неки конвексни дефект. Из тог разлога су издвојене тачке (оне почетне) које учествују у формирању дефекта. Функција која проналази дефекте, један дефект може посматрати као више узастопних малих дефеката, па су те тачке често густо груписане формирајући тако кластере [8]. На слици 11 приказане су те тачке (у даљем тексту ове тачке ће бити називане **дефектним тачкама**). Битно је напоменути да се граничне тачке контуре, односно тачка од које креће контура, као и тачка у којој се контура завршава, такође сматрају дефектним. Осим тога, на слици 11 приказан је и конвексни омотач контуре (енг. *convex hull*).



Слика 11: Конвексни омотач контуре са дефектним тачкама

Као што се може видети са претходне слике, неке дефектне тачке одређују прсте. Међутим, постоји доста тачака које нам нису од користи - оне тачке које се налазе на ободу шаке или дуж прстију. Такође, кластери тачака нам нису потребни, довољна је једна тачка која представља цео кластер. У следећем одељку следи објашњење алгоритма који проналази прсте филтрирајући дефектне тачке.

2.4 Одређивање корисних тачака шаке

Ова фаза алгоритма се састоји из следећих корака:

- пронаћи корену тачку шаке
- пронаћи блиске тачке - кластере густо концентрисаних тачака
- пронаћи центроиду кластера
- пронаћи долине тачака
- уклонити непотребне тачке - лажне прсте

2.4.1 Одређивање корене тачке шаке

Корена тачка шаке представља тачку која се налази на дну шаке, изнад ручног зглоба. Одлика OpenCV је да се замишљени координатни почетак налази у горњем левом углу слике, односно апсциса расте на доле. Математички гледано, уз претпоставку да корена тачка заузима идеалан положај, то би значило да корена тачка има највећу вредност апсцисе од свих дефектних тачака. Корену тачку налазимо користећи дефектне тачке и то оне које се налазе на почетку и на крају контуре. Према томе потребно је узети тачке са најмањом и највећом вредношћу ординате (с обзиром да контура прати облик шаке, то су крајња лева и крајња десна тачка шаке) и одредити средину између те две тачке - корена тачка се налази на половини линије која спаја крајње тачке. Уз то, корена тачка, како је претпостављено, има највећу вредност апсцисе, па је потребно пронаћи максималну вредност апсцисе међу свим дефектним тачкама.

```
1 def findPalmRoot(points):
2     if len(points) < 2:
3         raise Exception("Fatal error. Not enough points for palm
4         root localization.")
5     y = max(points, key=lambda point: point[1])[1] # ymax
6     pointsSortedByX = sorted(points, key=lambda point: point[0])
7     x = (pointsSortedByX[0][0] + pointsSortedByX[-1][0]) // 2
8     return x, y
```

Листинг 1: одређивање корене тачке шаке

2.4.2 Одређивање блиских тачака (кластера) тачака

Циљ овог корака је локализација кластера и њихова замена једном тачком чије геометријско место довољно добро представља тај кластер. Како ни шака, ни прсти нису савршено праве и равне површине, јављају се мини дефекти, па тако тачке које их одређују формирају кластере. Кластери су најчешћи на глатким површинама јагодица прстију, закривљеним деловима шаке, али и дуж прстију.



Слика 12: Дефектне тачке

Алгоритам који детектује кластере тачака своди се на рачунање еуклидског растојања између суседних дефектних тачака. Уколико су узастопне дефектне тачке довољно близу једна друге, сматрамо да припадају истом кластеру. Исе-чак кода који проналази кластере, односно групише густо концентрисане тачке у кластере, дат је у листингу 2.

```
1 def findClosePoints(points):  
2     closePoints = {}  
3     pointNo = 0  
4     batch = []  
5     distThreshold = ImageParam.consecutivePointsDistThreshold
```

```
6     coordsOffset = ImageParam.consecutivePointsCoordOffset
7     for i in range(len(points) - 1):
8         dist = distanceBetweenPoints(points[i], points[i + 1])
9         x1, y1 = points[i]
10        x2, y2 = points[i + 1]
11        if dist < distThreshold or (abs(x1 - x2) <= coordsOffset
12        and abs(y1 - y2) <= coordsOffset):
13            batch.append(points[i])
14        else:
15            if batch:
16                batch.append(points[i])
17                closePoints[pointNo] = batch
18            else:
19                closePoints[pointNo] = [points[i]]
20            batch = []
21            pointNo += 1
22    return closePoints
```

Листинг 2: додела тачака одговарајућем кластеру

Да би две тачке биле додељене истој групи (кластеру) потребно је да важи један од следећа два услова:

1. растојање између две суседне дефектне тачке је мање од неког прага - **ImageParam.consecutivePointsDistThreshold**. Овај праг се може подесити уносом са командне линије. Његова подразумевана вредност је 30.0.
2. из једне дефектне тачке се формира квадрат, тако да је та тачка у центру квадрата странице **2 * ImageParam.consecutivePointsCoordOffset**. Уколико се следећа дефектна тачка налази унутар овог квадрата, тачке припадају истом кластеру.

Резултат груписања тачака у кластере се може видети на слици 13, где су тачке из истог кластера обележене истом бојом.



Слика 13: Кластери дефектних тачака

2.4.3 Одређивање центроиде кластера

Након што су формиран кластери тачака, одређују се центроиде тих кластера. Центроиду ћемо окарактерисати као централну тачку чија позиција довољно добро апроксимира све тачке из кластера. Испробана су два приступа при одређивању позиције центроида: медијана координата тачака у кластеру и њихова средња вредност. Оба приступа дају приближно добре резултате.

```
1 def findClosePoints(points):
2     closePoints = {}
3     pointNo = 0
4     batch = []
5     distThreshold = ImageParam.consecutivePointsDistThreshold
6     coordsOffset = ImageParam.consecutivePointsCoordOffset
7     for i in range(len(points) - 1):
8         dist = distanceBetweenPoints(points[i], points[i + 1])
9         x1, y1 = points[i]
10        x2, y2 = points[i + 1]
11        if dist < distThreshold or (abs(x1 - x2) <= coordsOffset
12        and abs(y1 - y2) <= coordsOffset):
13            batch.append(points[i])
```

```
13     else:
14         if batch:
15             batch.append(points[i])
16             closePoints[pointNo] = batch
17         else:
18             closePoints[pointNo] = [points[i]]
19         batch = []
20         pointNo += 1
21     return closePoints
22
```

Листинг 3: формирање центроида кластера

Након што су одређене центроиде кластера, све тачке из кластера се занемарују и на даље се користе само центроиде (слика 14).



Слика 14: Центроиде кластера

2.4.4 Одређивање долина између прстију

Још један од битних корака при одређивању корисних тачака шаке јесте налажење долина између прстију. Долина је карактеристичан пример дефекта. Као дефект, долину формирају три тачке и то тако да су две тачке (почетна и крајња тачка дефекта) заправо врхови прстију (или се налазе на ободу прста), док је трећа тачка у понору долине. Оно што је још карактеристично је то да је угао у понору долине који формирају ове тачке када се споје у троугао, оштар. На слици 15 црвеном бојом је учртана тачка понора долине.

```
1
2 def findFingerValleys(defectPoints):
3     valleys = []
4     for defect in defectPoints:
5         start, end, far = defect
6         a = distanceBetweenPoints(start, end)
7         b = distanceBetweenPoints(start, far)
8         c = distanceBetweenPoints(end, far)
9         angle = calculateAngle(a, b, c)
10        if angle <= pi / 2 and (start[1] < far[1] and end[1] < far
11        [1]):
12            valleys.append((start, end, far))
13
14    return valleys
```

Листинг 4: одређивање долина прстију

Постојање долина је нарочито карактеристично за положај маказа, што се и може видети са слике 16.

2.4.5 Уклањање непотребних тачака - лажних прстију

Људска шака без морфолошких аномалија има пет прстију. Након свих описаних корака алгоритам најчешће даје више од пет прстију, односно налази више од пет корисних тачака. Те прсте, односно тачке, називаћемо лажним прстима. Како би лажни прсти били уклоњени спроводе се следећи поступци:

- уклањају се оне тачке које са кореном тачком, а у односу на праву која пролази кроз корену тачку и паралелна је ординатној оси, граде мањи угао - тај гранични угао који се проверава је подесив, а његова подразумевана вредност је ± 15 степени. Угао који се формира, обележимо га са θ , се рачуна као нагиб праве која пролази кроз корену тачку и тачку коју испитујемо. Нагиб се добија преко једначине 1, а затим сам угао добијамо користећи једначину 2. Програмске функције које рачунају угао су дате у листингу 5.

$$k = \frac{y_1 - y_2}{x_2 - x_1} \quad (1)$$

$$\theta = \arctan k \quad (2)$$



Слика 15: Долина између прстију када шака формира положај папира

```
1 def calculateLineSlope(firstPoint, secondPoint):
2     c = distanceBetweenPoints(firstPoint, secondPoint)
3     b = abs(firstPoint[0] - secondPoint[0])
4     a = abs(firstPoint[1] - secondPoint[1])
5     angle = calculateAngle(a, b, c)
6     return angle * 180 // pi
7
8 def calculateAngle(a: float, b: float, c: float) -> float:
9     """
10     Calculate angle (in radians) between edges b and c of
11     triangle by cosine theorem.
12     :param a: triangle edge
13     :param b: triangle edge
14     :param c: triangle edge
15     :return: angle between edges b and c
16     """
17     return acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c))
```

Листинг 5: рачунање угла између тачака



Слика 16: Долина између прстију када шака формира положај маказа

Коначно, део кода који филтрира тачке које заклапају мањи угао (дефинисан параметром `ImageParam.cutoffAngles`), се може наћи у листингу 6.

```
1 possibleFingers = []
2 angleOffset = ImageParam.angleOffset
3 cutoffAngles = ImageParam.cutoffAngles
4 for point in fingers:
5     angle = calculateLineSlope(rootPoint, point)
6     angle += angleOffset
7     if cutoffAngles[0] <= angle <= cutoffAngles[1]:
8         continue
9     possibleFingers.append(point)
10
```

Листинг 6: филтрирање тачака према углу

Такође, оно што је битно напоменути је утицај променљиве **`angleOffset`** који представља угаони параметар који се задаје код слика где шака не заузима идеалан положај, већ има неку угаону компоненту, односно где је рука померена за неки оштар угао у позитивном или негативном смеру (десно или лево) у односу на идеалан положај.

- уклањају се **колинеарне тачке**. Зависно од положаја који шака заузима, али и од карактеристика шаке са слике, често се дешава да се пронађе више дефектних тачака које су позициониране дуж неког прста или по ободу шаке између корене тачке и прстију. Такве тачке се сматрају колинеарним, с обзиром да прате правац простирања прста или бокова шака. Да би се такве тачке уклониле, посматрају се сваке две узастопне тачке - посматра се угао нагиба праве коју чине те две тачке. Ако је тај угао из неког дефинисаног опсега (подразумевано између 75 и 105 степени), подразумевамо да те две тачке припадају истом прсту или су лоциране на ободу шаке. Прелаз с једног на други прст, рецимо кажипрст на средњи прст, је оштрији, односно угао је мањи, па према томе те тачке не бивају филтриране. Уколико су две тачке колинеарне, уклањамо ону “нижу” тачку - тачку која има већу вредност апсцисе, с обзиром да је координатни почетак код OpenCV позициониран у горњем левом углу. На тај начин се остављају само вршине колинеарне тачке што најчешће представља врхове прстију. Када шака заузима положај који одступа од идеалног, где шака заузима дијагонални положај (када је њен положај отклоњен од идеалног за неки оштар угао, у позитивном или негативном смеру), потребно је анулирати дејство тог нагиба. У том случају, угао се задаје са командне линије као аргумент, а подразумевана вредност тог аргумента (**angleOffset**) је 0 степени.

```
1 def removeCollinearFingers(fingers):
2     fingerSet = set(fingers)
3     angleOffset = ImageParam.angleOffset
4     for i in range(len(fingers) - 1):
5         angle = (calculateLineSlope(fingers[i], fingers[i + 1])
6 + angleOffset) % 180
7         angleBounds = ImageParam.angleBounds
8         if angleBounds[0] <= angle <= angleBounds[1]:
9             if fingers[i][1] <= fingers[i + 1][1]:
10                 fingerToRemove = fingers[i + 1]
11             else:
12                 fingerToRemove = fingers[i]
13             if fingerToRemove in fingerSet:
14                 fingerSet.remove(fingerToRemove)
15     return list(fingerSet)
```

Листинг 7: уклањање колинеарних тачака

Посматрајући слику 14, видимо да три дефектне тачке с леве стране уз палац представљају колинеарне тачке.

- ако и након свих претходних операција, постоји више од пет дефектних тачака, прибегавамо уклањању оних тачака које су најближе кореној тачки, тако да остане 5 тачака које сматрамо прстима.

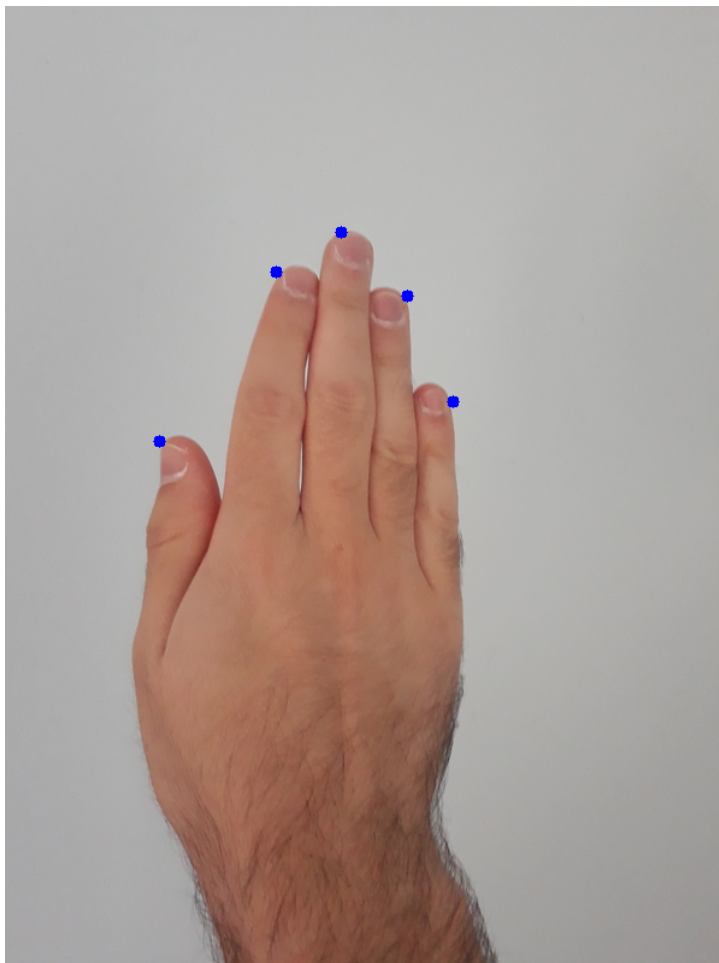
```
1 def removeFalseFingers(distances, fingersNo=5):
2     fingers = []
3     for finger, dist in sorted(distances.items(), key=lambda x:
4 x[1], reverse=True):
```



```
4     fingers.append(finger)
5     return fingers[:fingersNo]
6
```

Листинг 8: уклањање најближих лажних прстију

Након ових операција добија се пет тачака за које сматрамо да припадају врховима прстију.



Слика 17: Тачке које се сматрају прстима (врховима прстију)

2.5 Одређивање положаја који шака формира

Параметри који се узимају у обзир при одлучивању о положају шаке су:

1. колико прстију је јасно приказано на шаци
2. колико долина формира положај шаке
3. колике су дужине страна долина - колика су растојања од почетка долине до понора долине и од краја долине до понора долине
4. колике су висине прстију - за ову величину узећемо разлику у y координатама тачке прста и корене тачке.
5. колике су дужине пронађених прстију, где дужину прста процењујемо као растојање тачке прста од корене тачке шаке.
6. у случају да алгоритам проналази долине, посматрамо колике су дужине прстију који формирају долине.

Посматрајући наведене параметре, облици шаке дефинисани су на следећи начин:

- Уколико шака формира тачно једну долину, треба да размотримо да ли је у питању положај папира или маказа, пошто шака у положају камена не формира никакве долине, и обрнуто, уколико не постоји ниједна долина код шаке, сигурно није у питању положај маказа.
- Уколико уз једну долину, алгоритам проналази два прста, онда потврђујемо да је у питању положај маказа. С друге стране, ако уз једну долину алгоритам налази пет прстију, онда је у питању положај папира.
- Ако смо идентификовали једну долину, али не и два прста, друга могућност коју разматрамо да би потврдили да је у питању положај маказа, се заснива на висинама прстију и дужинама страна долине. Код маказа, два најдужа прста на руци (средњи прст и кажипрст) формирају долину. Друга карактеристика је та да су обе стране долине (растојања од врха прстију до понора) дугачке, и то приближно исте дужине - минимална дужина коју стране долина морају да имају да би задовољиле овај услов се дефинише као параметар који је могуће подесити из командне линије при покретању програма. Долине које се могу јавити код положаја папира су такве да постоје две могућности: 1) у питању је једна долина и њу најчешће формирају палац и кажипрст и у том случају имамо једну дугу страну долине (између кажипрста и понора долине), а друга страна је краћа и формира је палац са понором долине; 2) постоји више од једне долине, ако више прстију формира долине, тј. ако постоје размаци међу прстима.
- Број прстију код камена је варијабилан и алгоритам може наћи од једног до пет прстију, зависно од положаја шаке. Ипак, оно што највише дефинише камен као облик је карактеристика да нема долина. Облик папира може имати више долина, али такође не мора имати ниједну долину.

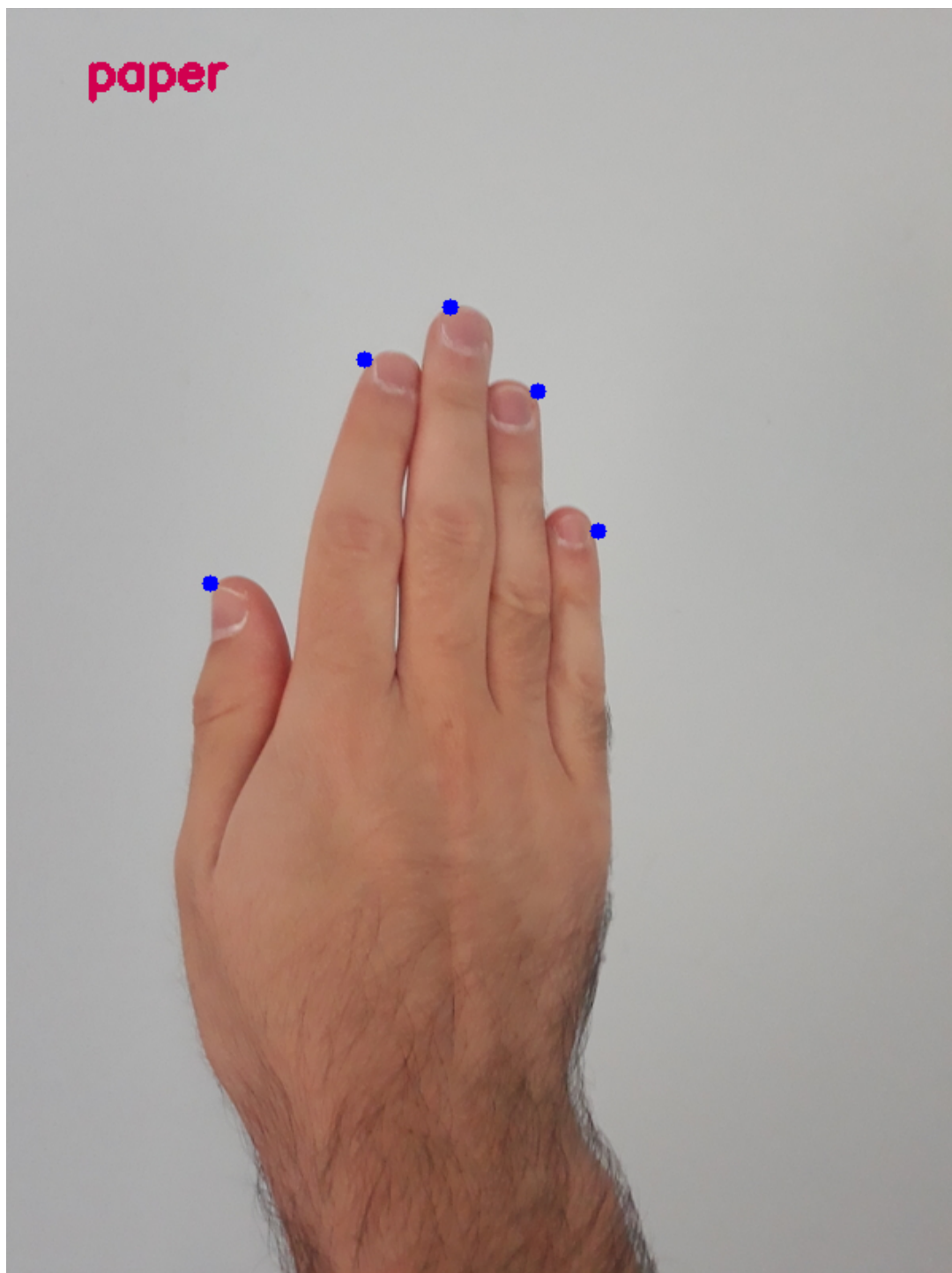
У случају када шака не формира долину, посматрају се дужине прстију. Код камена, прсти су савијени па су краћи него код папира где су прсти испружени. Тако, посматрамо разлику у дужини између најдужег и најкраћег прста, односно највишег и најнижег прста. Поред тога, узимамо три највиша прста и ако су сва три прста дужа од неког прага, знамо да је у питању папир, с обзиром да су код камена, како је већ наглашено, прсти савијени, па самим тим и краћи. Треба нагласити да се при процени дужине прстију, узимају дужине у односу на корену тачку шаке.

При одлучивању, неки од ових услова су искључиви и јасно одређују неки положај, док су неки асертивни, односно потврђују одлуку. Због тога функција која одређује који положај шака формира, враћа два резултата - положај који је пронађен и коментар колико је та одлука поуздана.

```
1 def determineGesture(fingers, rootPoint, rootToPointsDistances,
2   valleys):
3     """
4     :param fingers: list of finger points
5     :param rootPoint: root point of the palm
6     :param rootToPointsDistances: dictionary of distances between
7     root point and other points from fingers list
8     :param valleys: list of valleys
9     :return:
10    """
11    decisionCredibility = "uncertain"
12    foundGesture = None
13    maxDistance = max(rootToPointsDistances.values())
14    minDistance = min(rootToPointsDistances.values())
15    sortedFingersByHeight = sorted(fingers, key=lambda coord:
16    rootPoint[1] - coord[1], reverse=True)
17    if len(valleys) == 1:
18        if len(fingers) == 2:
19            foundGesture = "scissors"
20            decisionCredibility = "certain"
21        elif len(fingers) == 5:
22            foundGesture = "paper"
23            decisionCredibility = "certain"
24        else:
25            a = distanceBetweenPoints(valleys[0][0], valleys[0][2])
26            b = distanceBetweenPoints(valleys[0][1], valleys[0][2])
27            valley = valleys[0]
28            valleyTopsHeightCondition = abs(valley[0][1] - valley
29            [1][1]) < ImageParam.valleyTopPointsHeightDiff and \
30            valley[2][1] - max(valley
31            [0][1], valley[1][1]) > ImageParam.valleyDepthThreshold
32            valleyDepthCondition = valley[2][1] - valley[0][1] >
33            ImageParam.valleyDepthThreshold and \
34            valley[2][1] - valley[1][1] >
35            ImageParam.valleyDepthThreshold
36
37            fingerHeightsCondition = sortedFingersByHeight[0][1] -
38            sortedFingersByHeight[-1][
39            0] > ImageParam.fingersHeightDiff
40            if valleyTopsHeightCondition or (a >= ImageParam.
```

```
valleySidesLengthFactor * b) or \
33         fingerHeightsCondition or valleyDepthCondition:
34         foundGesture = "scissors"
35         decisionCredibility = "certain"
36     else:
37         foundGesture = "paper"
38         decisionCredibility = "certain"
39     else:
40         if len(fingers) < 5:
41             foundGesture = "rock"
42             decisionCredibility = "certain"
43         else:
44             # PAPER OR ROCK
45             minHeightFinger = min(fingers, key=lambda x: x[1])
46             maxHeightFinger = max(fingers, key=lambda x: x[1])
47
48             fingerLengths = [rootPoint[1] - finger[1] for finger in
sortedFingersByHeight[:3]]
49             minMaxFingersHeightCondition = maxHeightFinger[1] -
minHeightFinger[1] > ImageParam.fingersLengthDiff and \
50                                     maxDistance -
minDistance > ImageParam.minMaxFingerDist
51             longFingersCondition = all([finger >= ImageParam.
longestFingersHeightsThreshold for finger in fingerLengths])
52             if len(valleys) > 0:
53                 if len(fingers) == 5 or
minMaxFingersHeightCondition:
54                     foundGesture = "paper"
55                     if longFingersCondition:
56                         decisionCredibility = "certain"
57                     else:
58                         decisionCredibility = "uncertain - rock
properties"
59                 else:
60                     if minMaxFingersHeightCondition and
longFingersCondition:
61                         decisionCredibility = "uncertain - rock
properties"
62                     foundGesture = "paper"
63                 else:
64                     foundGesture = "rock"
65                     decisionCredibility = "uncertain - paper
properties"
66         return foundGesture, decisionCredibility
67
```

Листинг 9: одређивање положаја шаке



Слика 18: Резултат предвиђања положаја који шака формира

2.6 Резултати алгоритма

У овом поглављу биће приказани неки тест примери и резултати које алгоритам даје на тим примерима. Више пута је наглашено да је идеалан положај шаке најбољи и да у том случају није потребно подешавати параметре, а уз то алгоритам има одличну тачност. Ипак, алгоритам је тестиран на сликама где шака заузима различит положај. Неки од положаја који су тестирани су:

1. случај када шака заузима идеалан положај
2. случај када шака заузима хоризонтални положај - ротирана је за $+90$ или -90 степени у односу на идеалан (вертикални) положај
3. случај када се шака простира дуж дијагонале под неким оштрим углом, налево или надесно или када јој се виде бочне стране
4. случај када је шака окренута лицем и наличјем
5. случај када је неки прст издвојен (палац најчешће), нпр. код положаја папира или камена

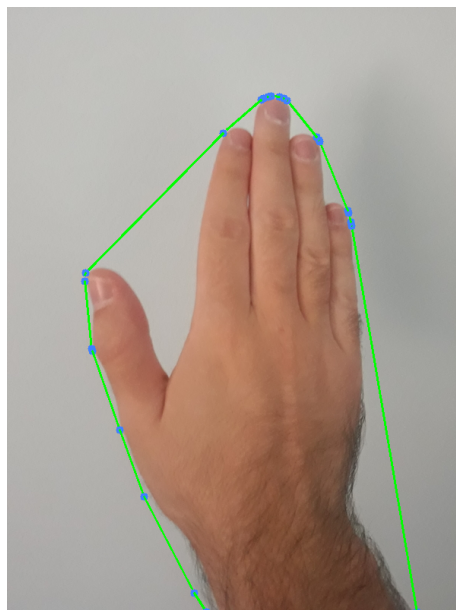
Када је шака ротирана хоризонтално, потребно је подесити угаони параметар **rotationAngle** тако да је његова вредност ± 90 степени. Када је шака постављена дијагонално, подешавањем параметра **angleOffset** коригује се угао за који је шака ротирана у односу на идеалан положај - овај параметар највише утиче на уклањање тачака које са кореном тачком граде неки мањи угао, али и колинеарних тачака. Код појединих случајева попут оног када је угао отклона већи од 90 степени, могуће је комбиновати ова два параметра тако да се слика ротира за 90 степени у позитивном смеру, а параметром **angleOffset** се зада угао којим треба поништити отклон ротиране слике у односу на идеалан положај.

С обзиром да димензије и облик људске шаке варирају зависно од пола, старосног доба, генетике, активности које особа обавља и сл., некада је потребно подесити и друге параметре, пре свега оне које представљају прагове дужина прстију, разлике у дужини између појединих прстију, дужина страна долина.... Поједини случајеви ипак захтевају детаљније подешавање параметара или се чак не могу ни решити, нпр. када је шака постављена тако да јој се прсти налазе у доњем десном углу, а корена тачка у горњем левом. За случајеве које није могуће решити овим алгоритмом потребне су додатне геометријске трансформације.

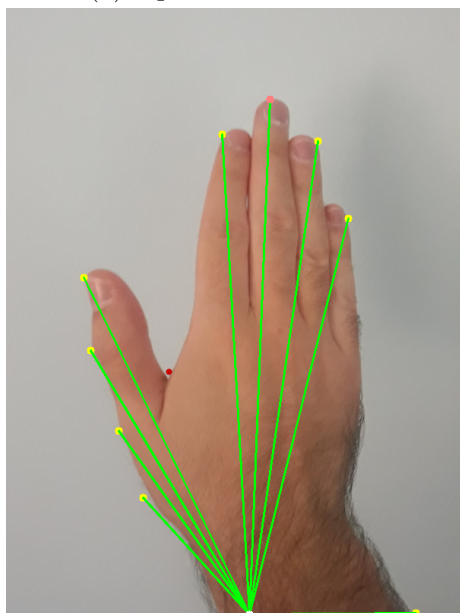
За тест скуп који је коришћен у овом раду алгоритам даје добре резултате. Од 39 слика из скупа, алгоритам тачно одређује положај за 38 слика што представља тачност од 97.44 %. У наставку следе тест примери и резултати које алгоритам даје. На првој слици матрице (горе лево) ће бити приказана оригинална слика, на другој (горе десно) следи приказ контурних дефектних тачака и конвексног омотача. На трећој (доле лево) слици биће приказане спојене дефектне тачке (жутом бојом), долине (тамно-црвеном бојом), најудаљенија дефектна тачка од корене (светло-црвеном бојом) и корена тачка (белом бојом). На последњој слици (доле десно) ће бити приказани прсти и положај који је алгоритам предвидео.



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

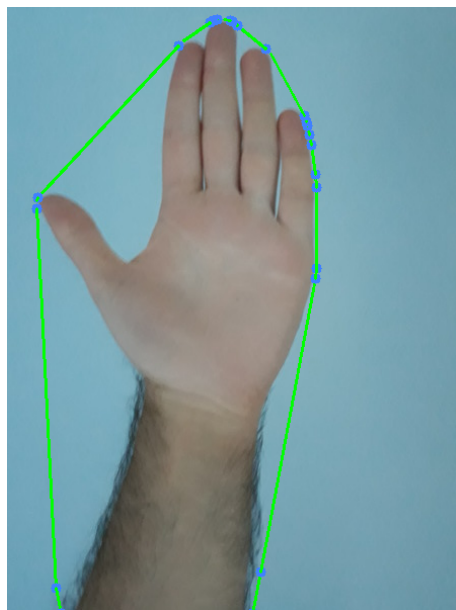


(г) Филтриране дефектне тачке и резултат

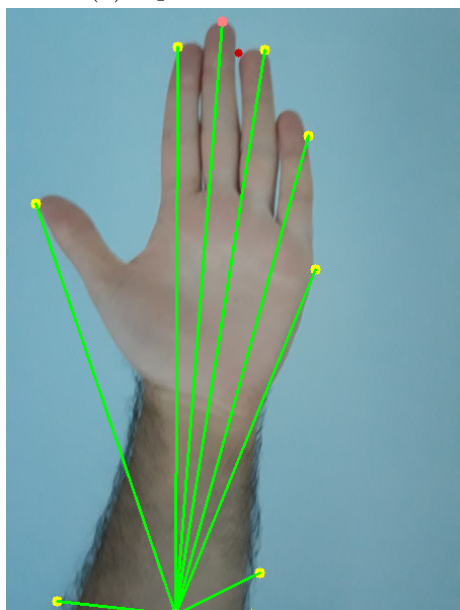
Слика 19: Пример положаја папира када је палац издвојен



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

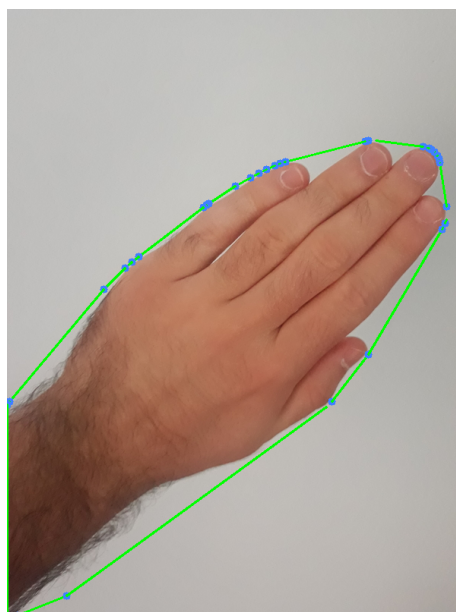


(г) Филтриране дефектне тачке и резултат

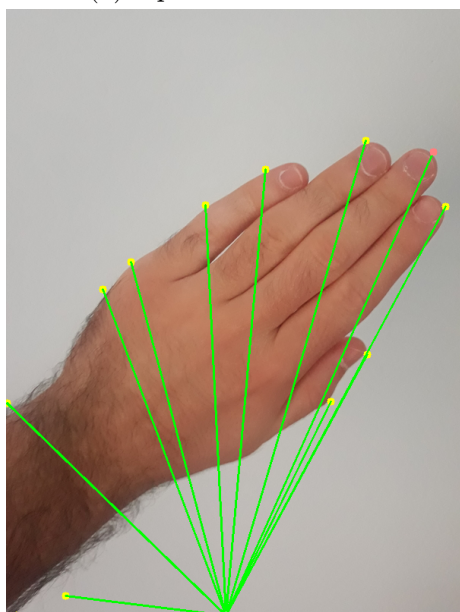
Слика 20: Пример положаја папира када је шака окренута лицем



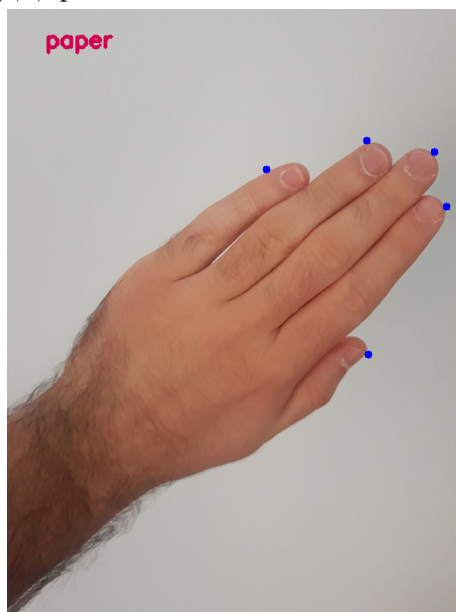
(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

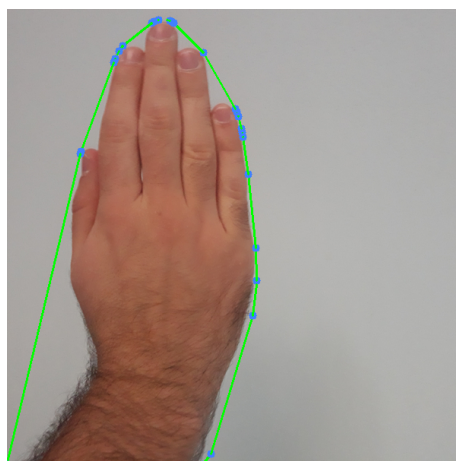


(г) Филтриране дефектне тачке и резултат

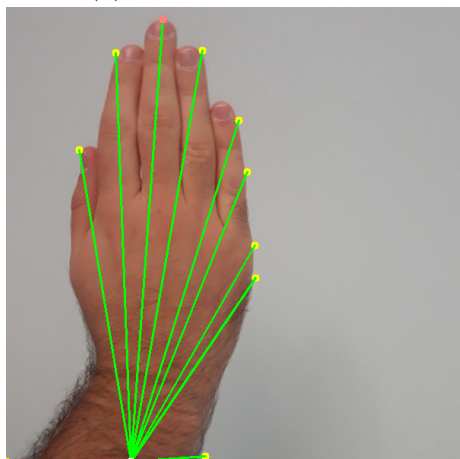
Слика 21: Пример положаја папира када је шака дијагонално позиционирана (45 степени)



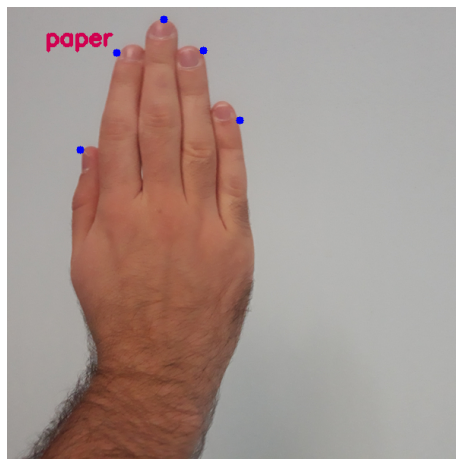
(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

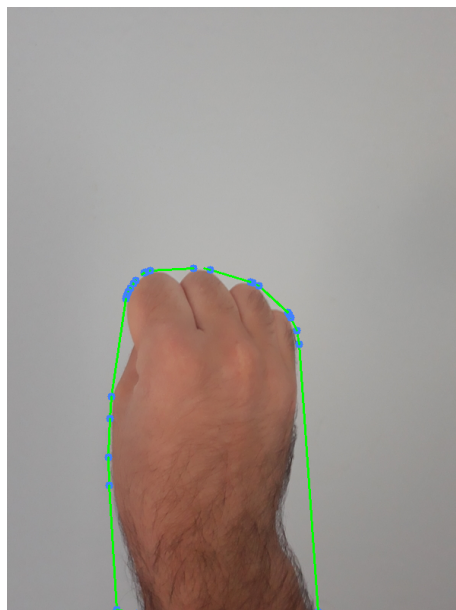


(г) Филтриране дефектне тачке и резултат

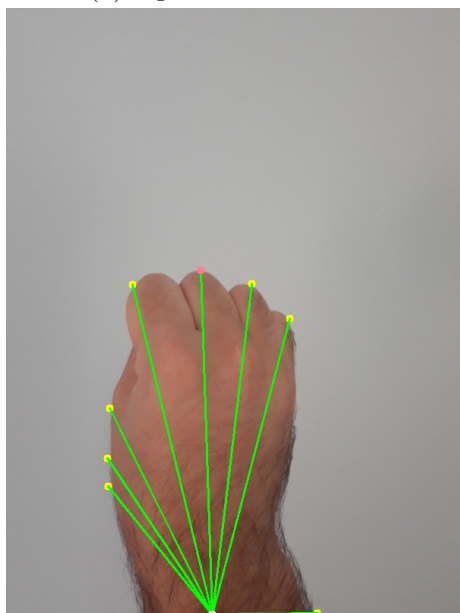
Слика 22: Пример положаја папира када је шака ротирана за 90 степени



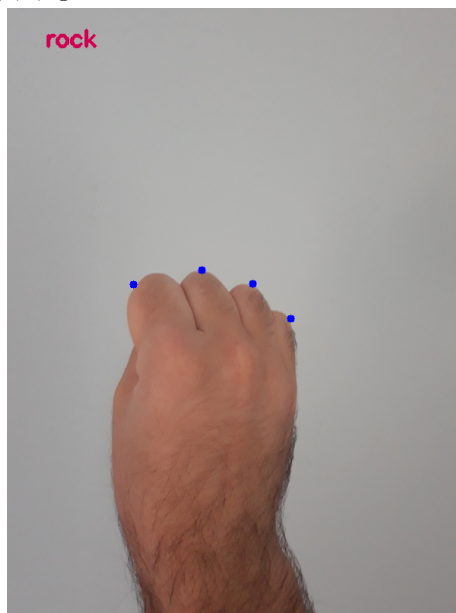
(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

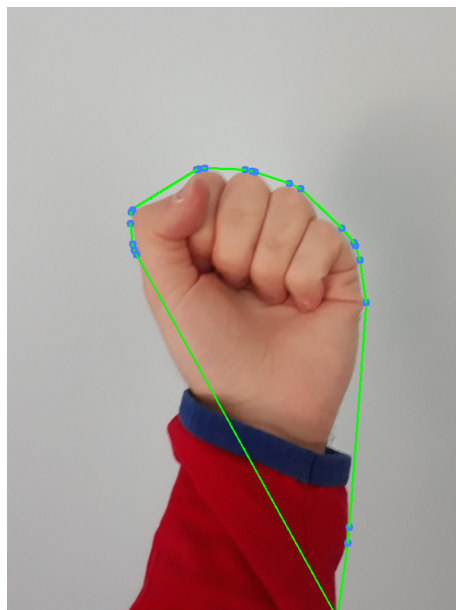


(г) Филтриране дефектне тачке и резултат

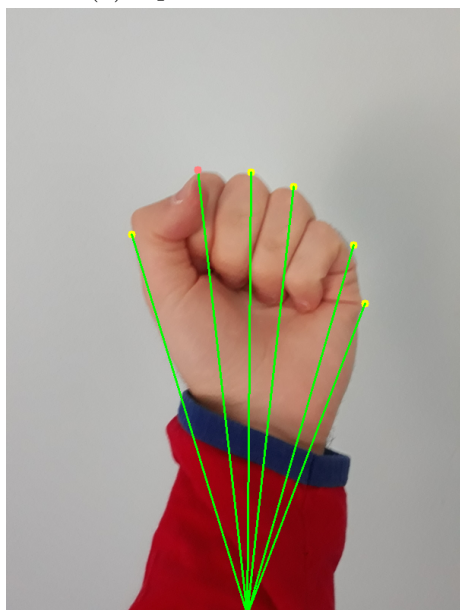
Слика 23: Пример положаја камена када је шака у идеалном положају



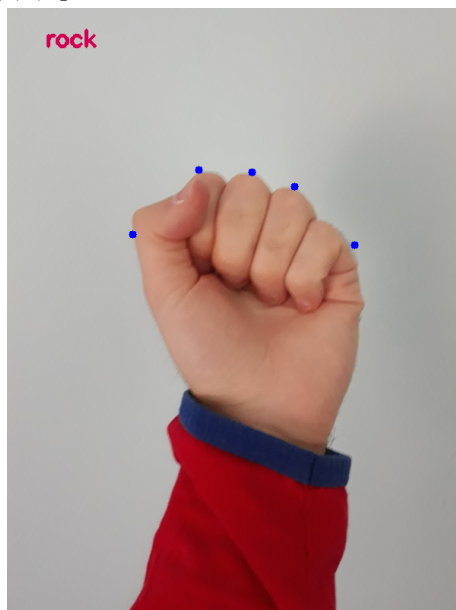
(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

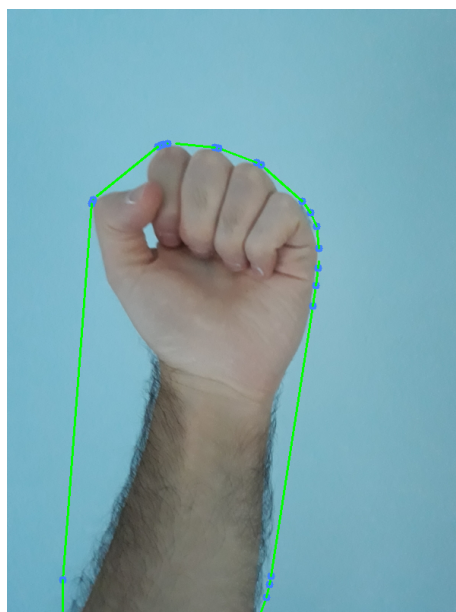


(г) Филтриране дефектне тачке и резултат

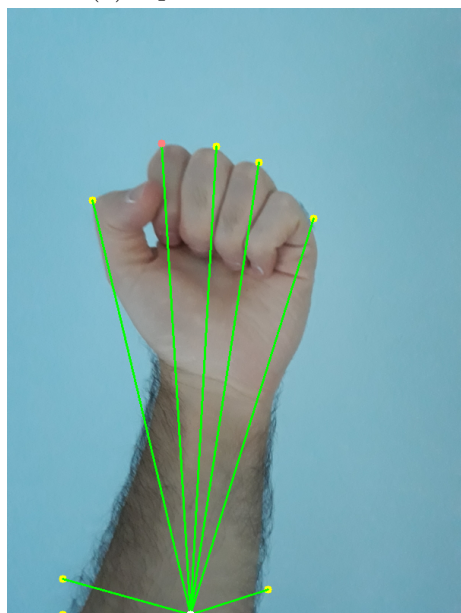
Слика 24: Пример положаја камена када је шака окренута лицем



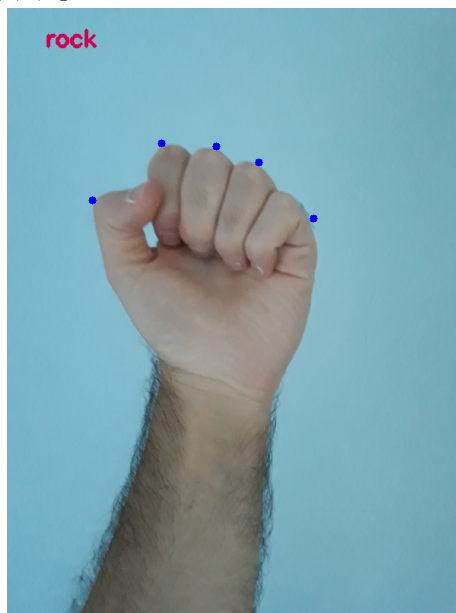
(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

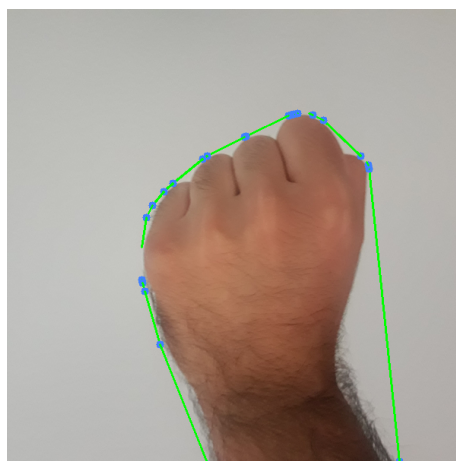


(г) Филтриране дефектне тачке и резултат

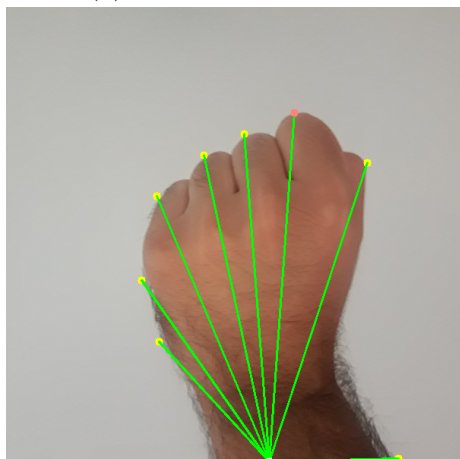
Слика 25: Пример положаја камена када је шака окренута лицем



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

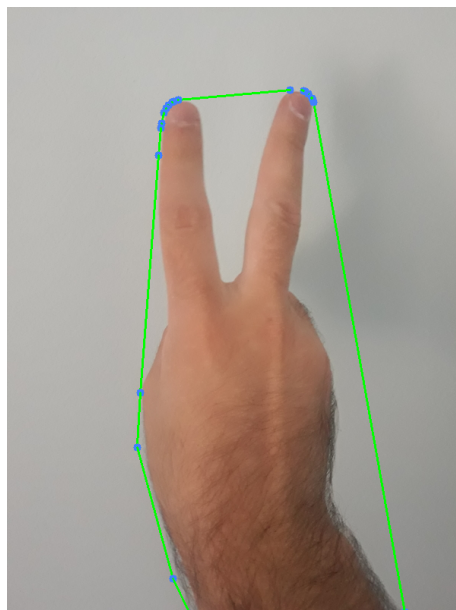


(г) Филтриране дефектне тачке и резултат

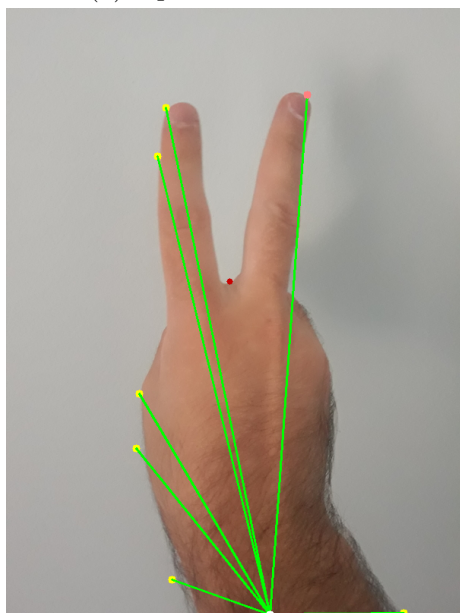
Слика 26: Пример положаја камена када је шака ротирана за -90 степени



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

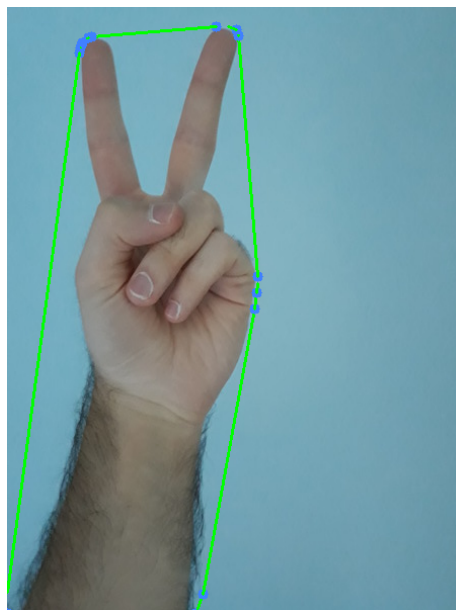


(г) Филтриране дефектне тачке и резултат

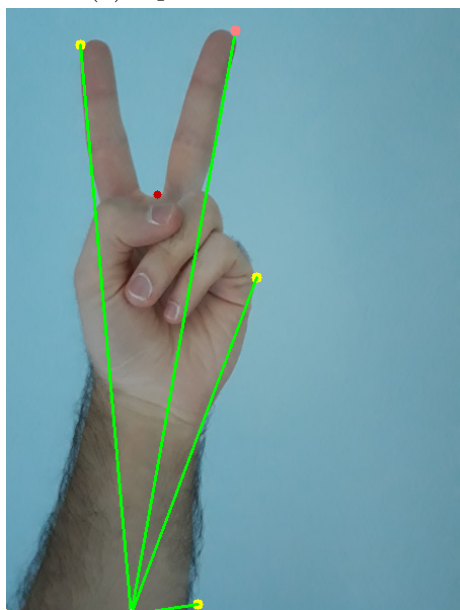
Слика 27: Пример положаја маказа када је шака у идеалном положају



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

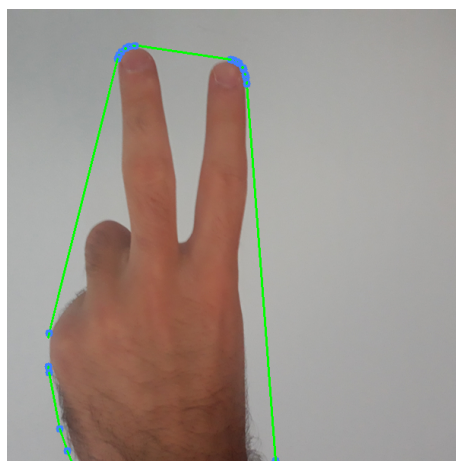


(г) Филтриране дефектне тачке и резултат

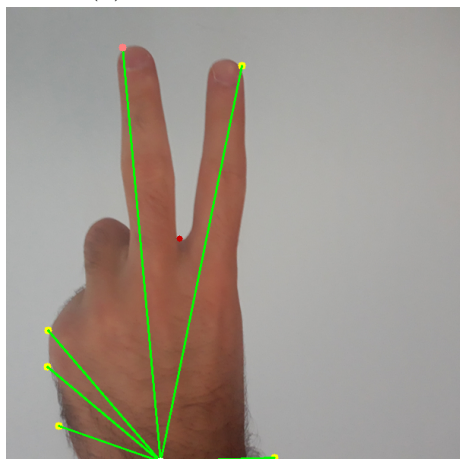
Слика 28: Пример положаја маказа када је шака окренута лицем



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке

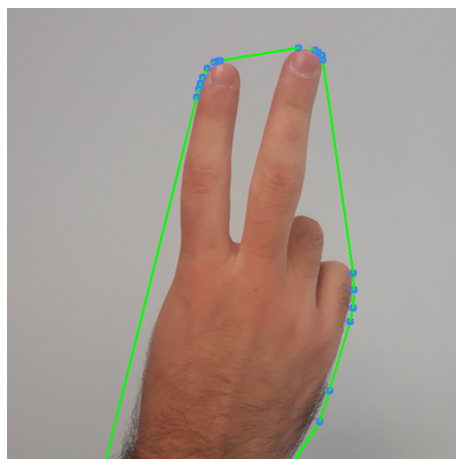


(г) Филтриране дефектне тачке и резултат

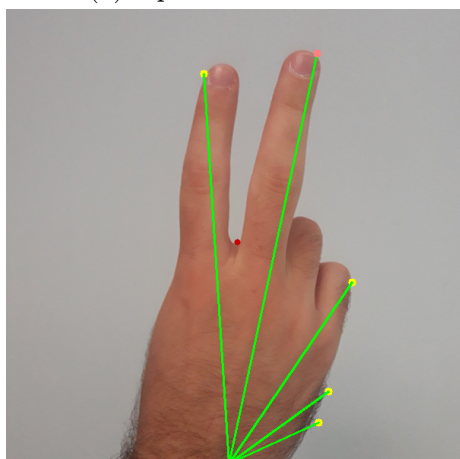
Слика 29: Пример положаја маказа када је шака ротирана за -90 степени



(а) Оригинална слика



(б) Дефектне тачке и конвексни омотач



(в) Дефектне и карактеристичне тачке



(г) Филтриране дефектне тачке и резултат

Слика 30: Пример положаја маказа када је потребна и ротација и прилагођавање нагиба

3 Контрола извршавања покретом

Други део овог мастера рада се односи на контролу извршавања покретом. Сажето, на основу карактеристичног положаја шаке, спроводи се одређена акција, односно контрола објекта од интереса.

Ова област је значајно постала актуелна развојем *Kinect* уређаја, али и дисциплина попут виртуалне стварности (енг. *virtual reality*) и допуњене стварности (енг. *augmented reality*). Тако се данас многе комерцијалне игре (виртуални тенис, голф...) свODE на интеракцију са играчем тако да играч покретом руком и/или телом управља играњем.

У овом раду је имплементиран програм као доказ концепта (енг. *proof-of-concept*) и функционише тако што са видео снимка који у реалном времену снима камера, сегментира шаку као објекат од интереса, препознаје карактеристичне покрете шаке и на основу тога спроводи управљачку акцију како би контролисао кретање актера у игри. Основна игра на којој је програм тестиран и даје најбоље резултате је Дино (*Chrome Dino*).

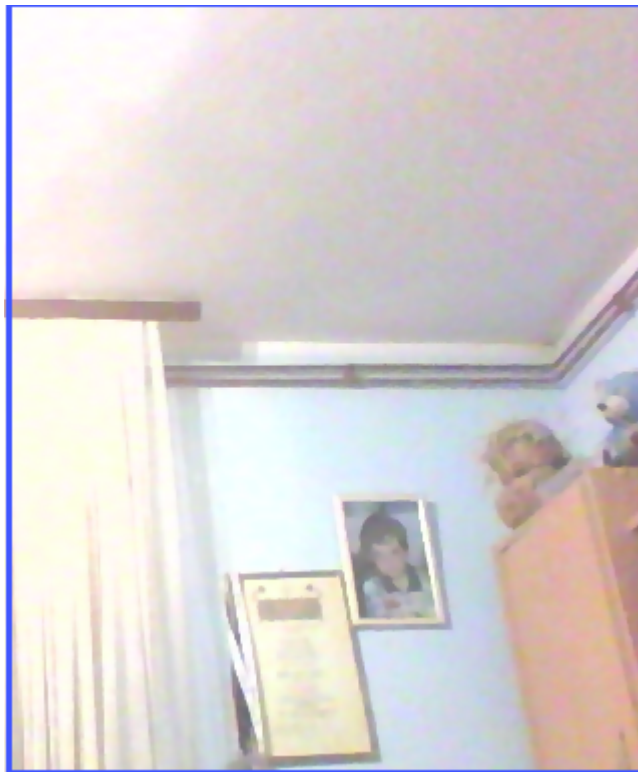
Ток извршавања контроле покретом изгледа овако:

1. снимити позадину без шаке
2. учитати и обрадити фрејм (енг. *frame*) са шаком
3. извршити препознавање шаке
4. одредити који карактеристичан положај је у питању
5. на основу одлуке извршити контролу кретања актера у игри

3.1 Снимање позадине

Извршење програма почиње снимањем кадра у ком се не налази шака. Циљ овог поступка је снимање позадине како би програм могао да прати шаку на тој позадини као објекат од интереса, без обзира који положај заузме и без обзира на то што ће се шака кретати, односно неће бити статична. Свакако да је јасно да позадина која се снима мора да буде иста као окружење у ком ће се појавити и шака, односно у питању је исти амбијент, јер у супротном ова акција нема ефекат [9].

Програм започиње са извршењем тако што уцрта правоугаоник у који ће бити постављена шака, па се тако и снима позадина тог дела амбијента (уоквиреног тим правоугаоником). Фрејм се прво филтрира применом билатералног филтра. Предност овог филтра у односу на Гаусов је та што овај филтар тежи да очува ивице при филтрирању. Гаусов филтар је просторни филтар који не обраћа пажњу на интензитет пиксела. С друге стране, билатерални филтар, који такође интерно примењује Гаусов филтар, води рачуна о интензитету пиксела, односно примењује Гаусов филтар тако да у рачунање нове вредности централног пиксела укључује само оне пикселе под кернелом који имају сличан интензитет као централни пиксел [6]. Даље, корисник притиском на тастер **b** са тастатуре потврђује снимање фрејма позадине. Позадина (односно маска позадине) се може преснимити, односно ресетовати у сваком тренутку притиском на тастер **r** са тастатуре. На даље, посматра се само онај део слике који је оивичен правоугаоником (слика 31).



Слика 31: Област од интереса посматраног фрејма

Када се позадина сними, она функционише као маска фрејма са шаком када се примени битовска конјункција. Упрошћено, алгоритам упореди фрејм без шаке (позадину) и фрејм са шаком, и циљ му је да пронађе у чему се разликују та два фрејма - да пронађе “разлику”. На тај начин сегментише објекат који није постојао код фрејма без шаке. Овај поступак је спроведен применом објекта **MOG 2 background subtractor** у оквиру OpenCV [9].

3.2 Обрада фрејма

Након што је позадина снимљена, шака може да се постави у кадар (плати правоугаоник). Пример таквог кадра је дат на слици 32.

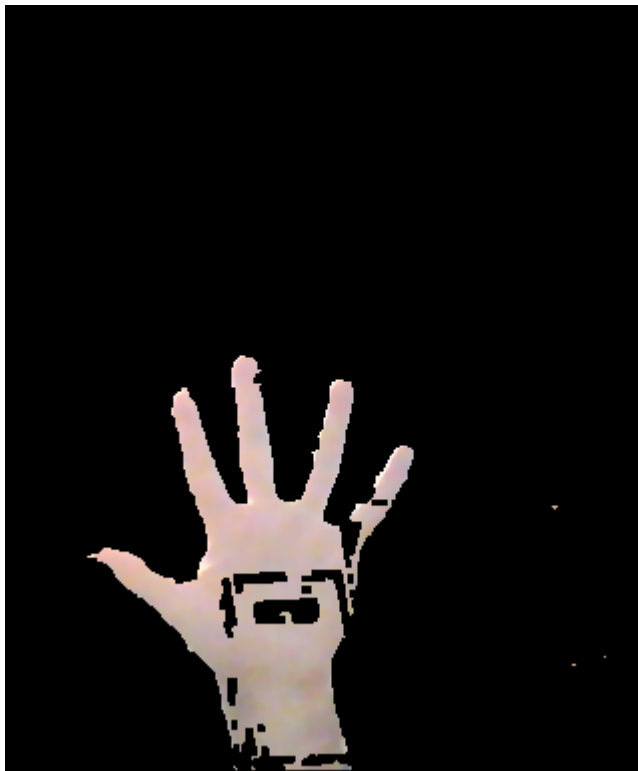


Слика 32: Фрејм са постављеном шаком

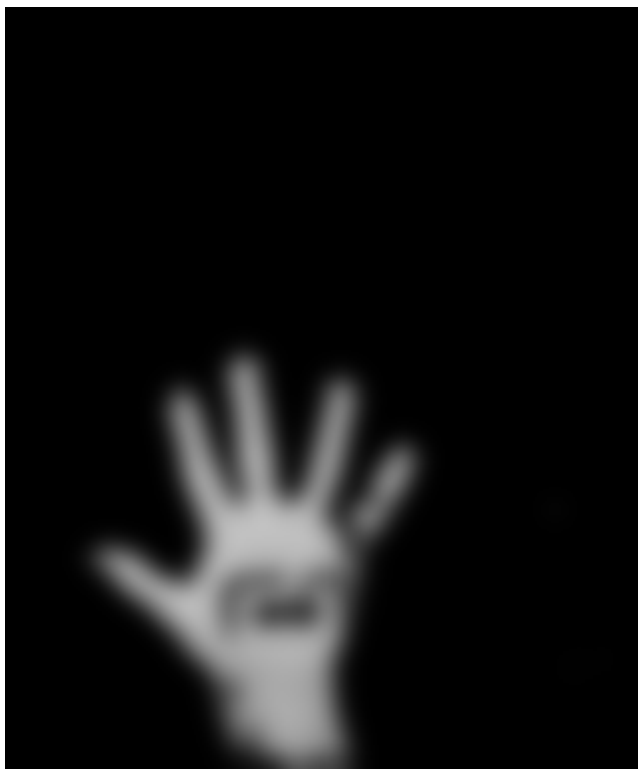
Наредни корак представља уклањање позадине применом *background subtractor*-а на начин који је већ објашњен. Када се позадина уклони, односно маскира са оригиналним фрејмом, то изгледа као на слици 33.

Као што се може видети, објекат је детектован, али је потребно отклонити шум који се јавља у унутрашњости шаке, а који је настао при маскирању фрејма маском позадине. Како слика у боји није погодна за рад, слику ћемо превести у *grayscale*, а затим додатно филтрирати применом Гаусовог филтра (слика 34) [4].

Након замућења слике, слику можемо превести у бинарни облик применом OpenCV **threshold** функције. Праг бинаризације је подложен промени и зависи од осветљења и позадине. У случајевима када желимо да детектујемо ивице или конуре, овакав формат слике је најпогоднији због уклоњеног шума и непотребних детаља у унутрашњости објекта (слика 35) [10].



Слика 33: Шака након уклањања позадине



Слика 34: Изглед фрејма након филтрирања



Слика 35: Изглед фрејма након бинаризације

3.3 Детекција шаке и одређивање карактеристичног положаја

Након бинаризације је добијена јако добра слика за рад, без шума и детаља који би правили сметње при детекцији шаке. Под детекцијом шаке подразумева се проналажење контуре коју шака формира, а поред тога одређује се и конвексни омотач контуре као и долине између прстију као и у првом делу рада (коришћен исти алгоритам). Сви пронађени параметри се уцртавају ради лакшег праћења рада алгоритма, како би резултат алгоритма и детекције био очигледан.

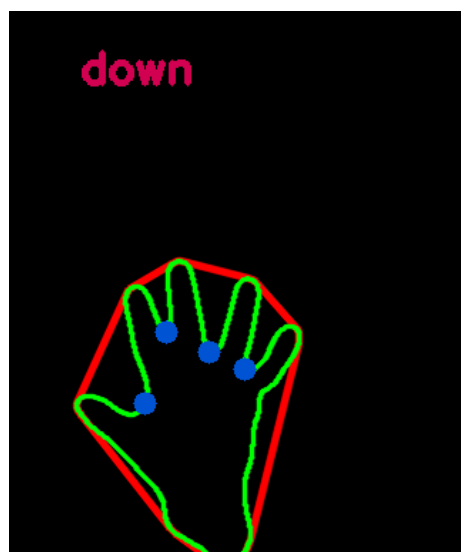
Када се пронађе контура, а пре свега број долина, одређује се који положај је у питању. С обзиром да је овај део рада конципиран као доказ концепта, ствари су мало упрошћене. Према томе, разликујемо три различите ситуације (положаја) у односу на број пронађених долина:

1. није пронађена ниједна долина
2. пронађена је тачно једна долина
3. пронађено је више од једне долине

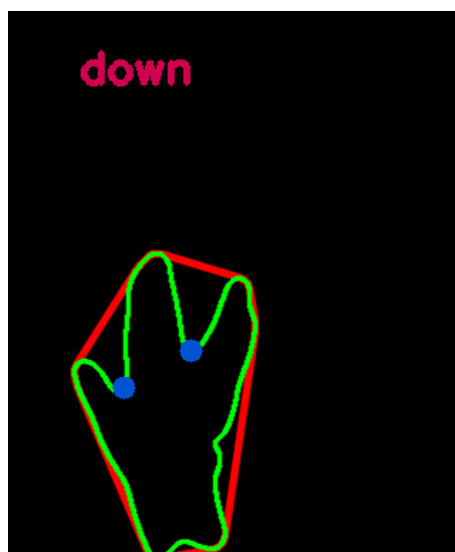
Како је циљ овог програма контрола игре путем покрета шаке, дефинисаћемо које команде се изводе у зависности од детектованог положаја. Према томе, ако није пронађена ниједна долина, сматраћемо да нема никакве команде, односно то је неактиван (енг. *idle*) потез. Уколико је пронађена једна долина, спроводи се команда скока, еквивалентна притиску горње стрелице на тастури. У случају

када је детектовано више од једне долине, изводи се чучањ, односно савијање, што је еквивалентно држању доње стрелице на тастатури.

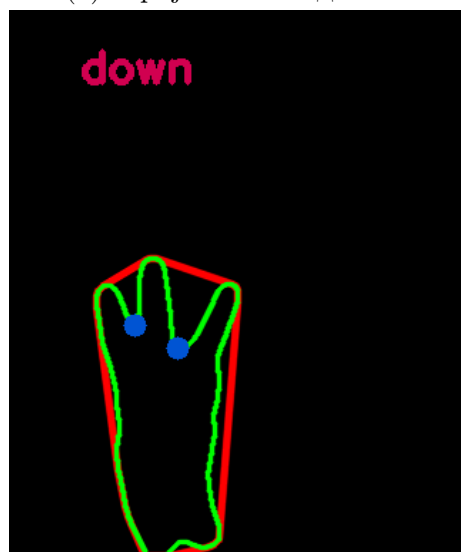
Према дефинисаним правилима, лако се може уочити да покрети нису једнозначно одређени, што значи да велики број покрета задовољава услов броја долина. Докле год шака формира дефинисани број долина за неку команду, покрет се препознаје као важећи и акција се спроводи. У наставку следи приказ неких варијација положаја који задовољавају сваки од три дефинисана услова на основу броја долина које шака формира.



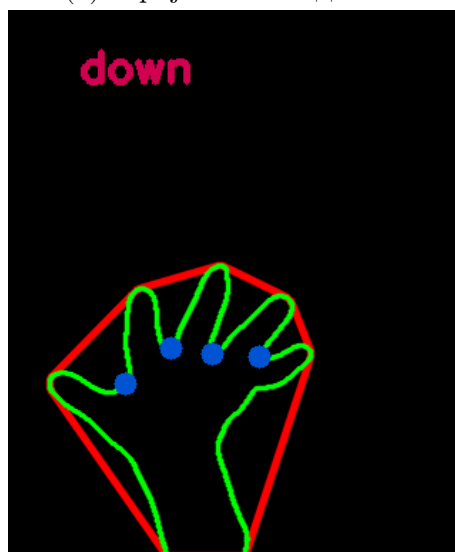
(а) варијанта 1 - 4 долине



(б) варијанта 2 - 2 долине



(в) варијанта 3 - 2 долине

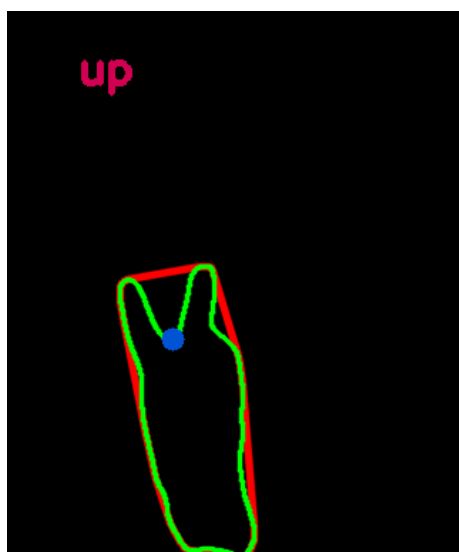


(г) варијанта 4 - 4 долине

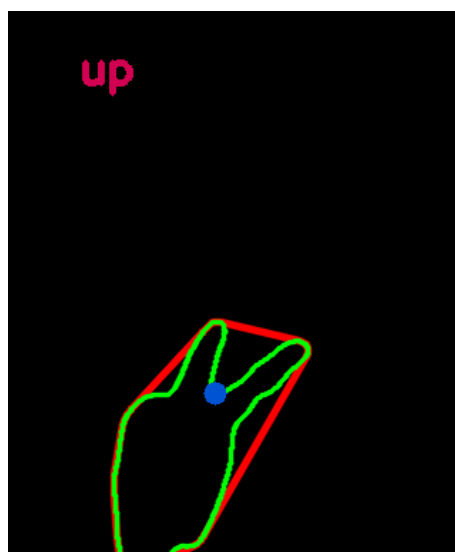
Слика 36: Пример положаја шаке када је пронађено више од 1 долине



Слика 37: Пример положаја шаке када није пронађена долина



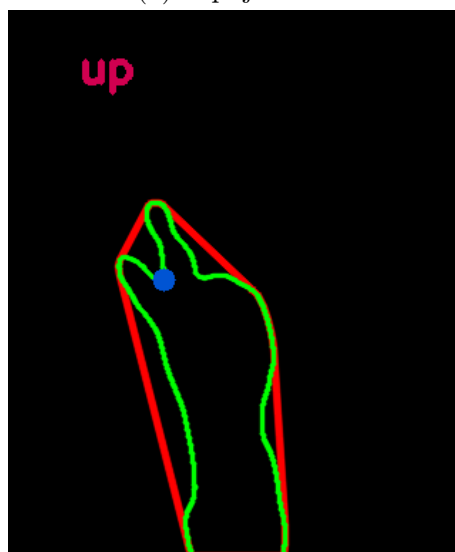
(а) варијанта 1



(б) варијанта 2



(в) варијанта 3



(г) варијанта 4

Слика 38: Пример положаја шаке када је пронађена 1 долина



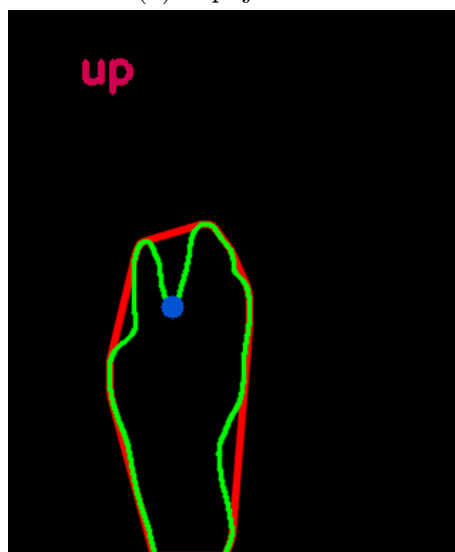
(а) варијанта 5



(б) варијанта 6



(в) варијанта 7



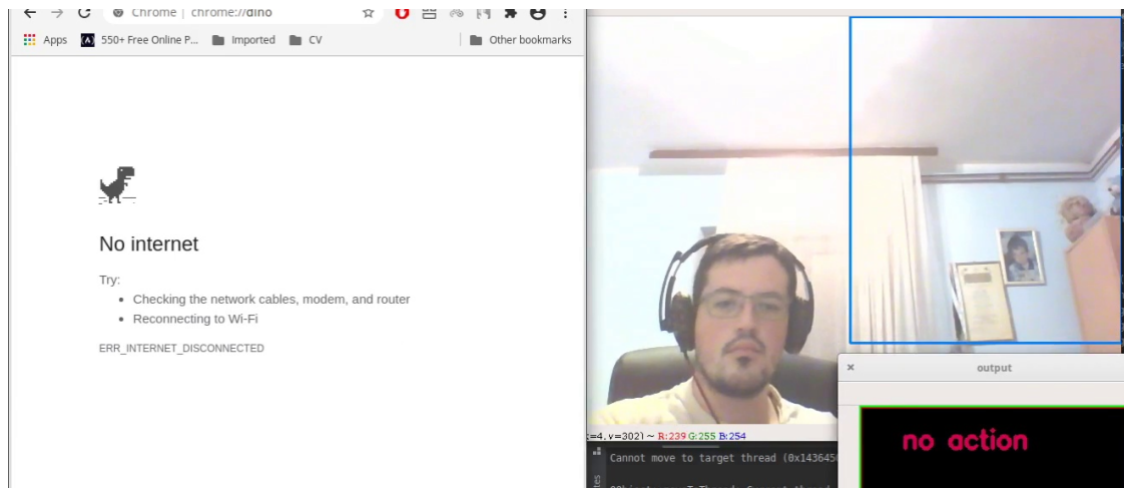
(г) варијанта 8

Слика 39: Пример положаја шаке када је пронађена 1 долина

3.4 Контрола игре на основу положаја шаке

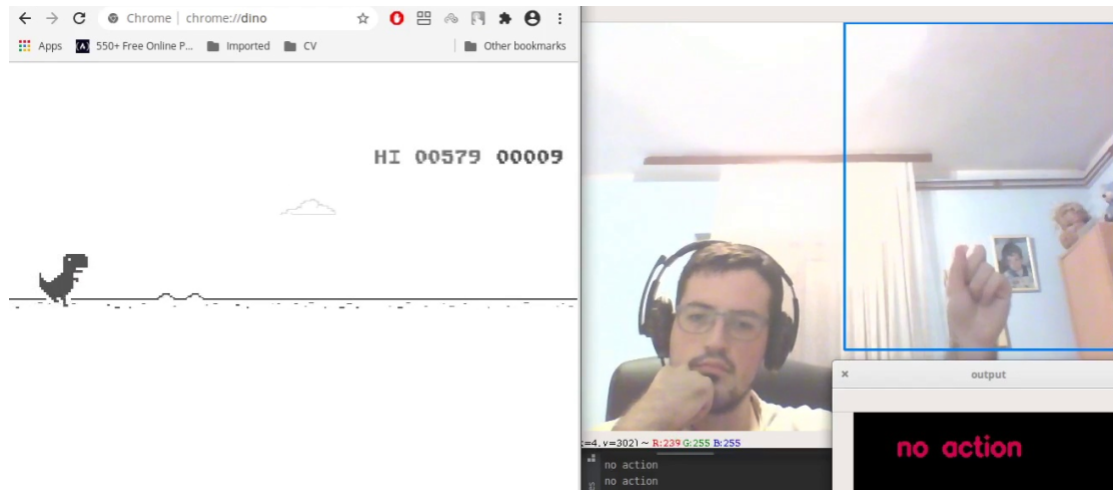
Након што је детектован и одређен положај шаке, спроводи се жељена акција над актером игре. Контрола карактера игре *Chrome Dino* је пожељна за демонстрацију и доказ концепта управљања игром на основу покрета шаке. Ова игра је врло једноставна и заснива се на свега два покрета - скоку и чучњу. Игра је доступна и *online* и *offline* у оквиру Chrome претраживача или на следећој интернет адреси: `chrome://dino/`.

Главни карактер игре је Дино (тираносаурус рекс) који се аутоматски креће (без притиска на стрелице) и циљ је да пређе што већу дистанцу и да се при томе не судари са неким од објеката или ликова (кактуси или други карактери, такође диносауруси). Неки објекти и карактери су статични или се крећу по тлу и потребно их је прескочити, док неки карактери лете и потребно је савити се при проласку поред њих. Јасно је да при скоку треба формирати положај шаке са једном долином међу прстима, док је за савијање (карактер се креће у савијеном положају), потребно формирати положај шаке тако да се уочава више од једне долине. Основни положај на који се не шаље никакав управљачки сигнал је било који положај без долина. Након што се детектује жељени положај шаке, програм шаље контролни сигнал, имитирајући притисак одређеног тастера на тастатури. За ове сврхе коришћене су функције из модула **pyautogui** [11]. На сликама 40, 41, 42 и 43 су дата четири различита случаја игре.

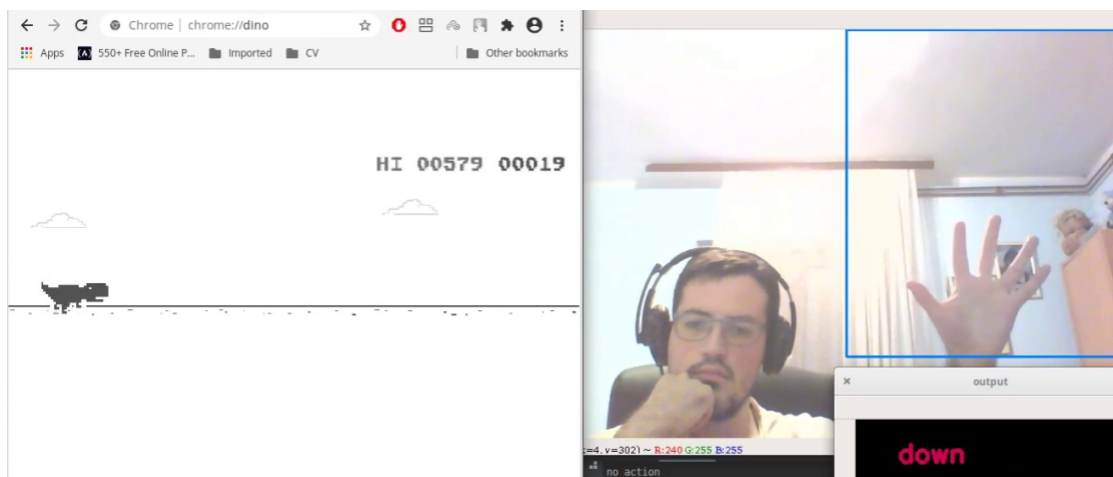


Слика 40: Почетак игре

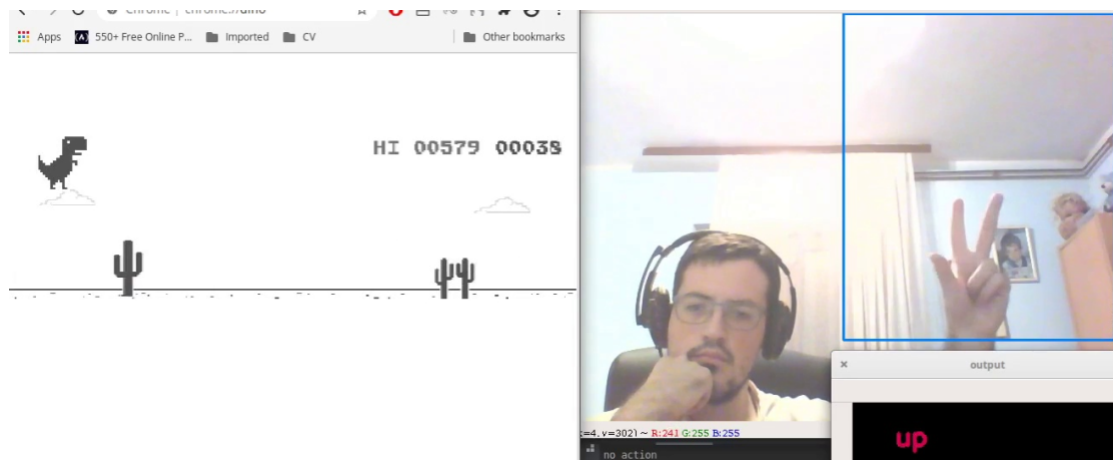
Препознавање карактеристичних положаја шаке у циљу контроле извршавања покретом



Слика 41: Подразумевани положај шаке, без акције



Слика 42: Положај шаке који изазива савијање, чучањ



Слика 43: Положај шаке који изазива скок

Осим игре Дино, контрола игре је тестирана и на другим играма попут: Лепршава птица (енг. *Flappy bird*), Сурф у метроу (енг. *Subway surf*), Трк бесне баке (енг. *Angry granny run*)...

Најбољи резултати контроле игре се постижу код игре Дино. Остале игре или имају комплексније покрете, доста су динамичније (играчи се брже крећу), за-снимају се на клику миша или су превише инертне.

Неки проблеми који се могу јавити код контроле игара на основу препознавања покрета шаком су:

- погрешно препознавање покрета изазива погрешну команду
- прелази између два положаја шаке - нпр. ако је циљ формирати покрет са више од једне долине, може се десити да се препозна једна долина при промени положаја шаке из положаја без долина у положај са више од једне долине
- ситуација када више узастопних фрејмова представља исти положај - да ли треба спровести један покрет, више узастопних покрета или дужи покрет (са задршком)

4 Закључак

У овом раду су демонстриране неке методе обраде слике, њихов значај и улога у трансформисању улазних података (слика). Иако се доминантни алгоритми код сегментације објеката, препознавању облика и праћењу објеката заснивају на методама дубоког учења, овај рад се пре свега базира на геометријским трансформацијама слика и објеката од интереса и препознавању облика на основу просторних карактеристика шаке - растојањима између прстију, дужинама прстију, неким угловима које шака и прсти формирају. Свакако да из тог разлога овај приступ није потпуно робустан и поуздан, јер се у великој мери ослања на параметре који се подешавају у зависности од квалитета слике, позиције и положаја шаке, али и на анатомска својства шаке. С друге стране предности овог приступа су једноставност коришћеног алгоритма, подесивост виталних параметара, могућност надоградње алгоритма с обзиром на модуларност кода....

Следећи корак у развоју ове апликације је тестирање алгоритма на сликама на којима се налазе шаке са различитом физиономијом: кратки прсти, веома дуги прсти, дебљи прсти, женска шака, шака са израженим ноктима....Осим тога, предлог је да се имплементира сасвим нови алгоритам који би се заснивао на алгоритмима дубоког учења. На тај начин би се добио сасвим нови приступ у решавању овог проблема где би систем мање био зависан од подешавања параметара. Што се тиче аспеката апликације који се тичу корисничког доживљаја и квалитета овог производа као софтвера, наредни кораци би подразумевали развој графичког корисничког интерфејса (енг. *graphical user interface*), где би се параметри уносили кроз графичко окружење уместо кроз командну линију. Још један потез који би употпунио кориснички доживљај ове апликације, нарочито оног дела са праћењем шаке и предикцијом положаја који шака формира, је развој верзије за мобилне уређаје (телефоне и таблете), с обзиром да ова апликација можда има већу употребну вредност ако се користи на мобилним телефонима и таблетима.

Што се тиче употребе овог софтвера у пракси, треба указати на употребу неких аспеката ове апликације у биомедицинском инжењерству и неким истраживачким сферама претклиничке и клиничке дијагностике - испитивање карактеристика шаке (здраве, са малформитетима, шака које су претрпеле повреде или преломе и сл.). Други вид употребе ове апликације би се огледао у имплементацији неке врсте учила знаковних језика (намењеном глувим и наглувим особама, глувонемима и особама са сметњама у слуху). Још неке могућности употребе ове апликације се огледају у развоју игара за вежбање моторике код деце и спортиста, али и виртуелном игрању забавних игара што је већ и наглашено у другом делу рада.

Литература

- [1] Миодраг Поповић. *Дигитална обрада слике*. Академска мисао, 2006.
- [2] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2 edition, 2002.
- [3] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Geometric transformations. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html#geometric-transformations. (приступљено октобра 2020.).
- [4] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Converting colorspace. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces. (приступљено октобра 2020.).
- [5] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Canny edge detection. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html#canny. (приступљено октобра 2020.).
- [6] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Smoothing images. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering. (приступљено октобра 2020.).
- [7] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Morphological operations. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html#morphological-ops. (приступљено октобра 2020.).
- [8] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Contours in opencv. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_table_of_contents_contours/py_table_of_contents_contours.html. (приступљено октобра 2020.).
- [9] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Background subtraction. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html. (приступљено октобра 2020.).
- [10] Alexander Mordvintsev and Abid K. Opencv-python tutorials: Image thresholding. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding. (приступљено октобра 2020.).

- [11] Al Sweigart Revision. Pyautogui docs: Keyboard control functions. <https://pyautogui.readthedocs.io/en/latest/keyboard.html>. (приступљено октобра 2020.).