Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Електротехника и рачунарство

Ниво студија: Мастер академске студије

Предмет: Методе формирања и обраде дигиталне слике

Број индекса: 444/2019

Софија М. Анђелковић

Предобрада слика QR кодова у циљу ефикаснијег препознавања

Мастер рад

Комисија за преглед и одбрану:

1. **Др Маријана Гавриловић Божовић, доцент - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог мастер рада кандидаткиња ће проучавати методе које се користе за обраду дигиталне слике, а у циљу поправке квалитета слика QR кодова. Рад ће се састојати од теоријског дела, где ће кандидаткиња дати преглед резултата из области и изложити основне кораке неопходне у предобради слике ради уклањања евентуалних недостатака услед лошег оптичког система, осветљаја итд. Допринос кандидаткиње ће бити и у самостално написаним функцијама које ће вршити поправку квалитета слике као корак пре препознавања кода са слике.

Препоручена литература:

[1] R. E. Woods. R. C. Gonzalez, Digital Image Processing, 4th edition, Pearson, 2018.

[2] R. E. Woods. R. C. Gonzalez, S. L. Eddins, Digital Image Processing Using MATLAB, 3rd edition, Gatesmark, 2020.

[3] М. В. Поповић, Дигитална обрада слике, Микро књига, 2006.

Крагујевац, 12. 03. 2020.

Ментор:

Др Маријана Гавриловић Божовић, доцент

_____

Садржај

1. Увод 6
2. Дигитална обрада слике 7
3. QR код 8
3.1. Елементи QR кода 9
4. Предобрада слика QR кодова 14
4.1. Бинаризација 15
4.1.1. Основни алгоритам за тражење прага 15
4.2. Локализација 16
4.2.1. Поплава алгоритам 19
4.2.2. Момент слике 20
4.3. Геометријско исправљање 24
4.3.1. Афине трансформације 24
4.4. Формирање бинарне матрице 27
5. Имплементација 28
6. Закључак 41
7. Литература 42
8. Прилог: Кодови 44

Резиме рада:

У оквиру рада је теоријски описан и имплементиран процес предобраде дигиталних слика QR кодова ради ефикаснијег препознавања. Процес предобраде се састоји од неколико кључних фаза: процеса бинаризације, локализације QR кода помоћу позиционих образаца и геометријског исправљања применом Афиних трансформација. Последња фаза предобраде чини формирање бинарне матрице, како би рачунар могао да прочита податке кодиране у оквиру кода. Имплементација је рађена помоћу програмског језика Python3 и његових библиотека NumPy и OpenCV. Целокупна имплементација је приказана на примерима.

Кључне речи:

QR код, кодирање, дигитална слика, предобрада дигиталне слике, бинаризација, локализација, позициони обрасци, Афине трансформације, геометријско исправљање, Python3

Summary of work:

This thesis theoretically describes and implements the process of pre-processing digital images of QR codes for more efficient recognition. The processing consists of several key phases: the binarization process, the localization of the QR code using finder patterns, and the geometric correction using Affine transformations. The last phase of pre-processing is the formation of a binary matrix so that the computer can read the encrypted data encoded in the code. The implementation was done using the Python3 programming language and its NumPy and OpenCV libraries. The entire implementation is shown through the examples.

Key words:

QR code, encoding, digital image, image pre-processing, binarization, localization, finder patterns, Affine transformation, geometric correction, Python3

1. Увод

Живимо у свету који је сваким даном све више дигитализован. Осим што све око нас постаје дигитално, наша свакодневница постаје све бржа. Такво окружење је направило идеалну подлогу за масовну употребу QR кодова у свим животним сферама. Иако је патентиран пре скоро 30 година, за потребе аутомобилске индустрије, тек је у послењих пар година доживео експанзију и нашао примену у складу са својим капацитетима. Данас их користимо кад хоћемо да се конектујемо на бежичну WiFi мрежу без да укуцавамо лозинку или на аеродрому уместо папирне авионске карте. Када хоћемо да одемо на неки веб сајт, без да укуцавамо целу адресу или ако хоћемо да видимо мени у ресторану. Нема потребе да носимо са собом карту за концерт или музеј, довољно је да имамо генерисан QR код карте на нашем телефону.

Приликом читања (скенирања) кода и формирања његове дигиталне репрезентације може доћи до његовог оштећења који се манифестује појавом геометријских дисторзија, нехомогене осветљености и присуства шума. На степен оштећења утиче квалитет камере којом се врши читање, али и амбијентално осветљење. Напретком технологије, камере на мобилним телефонима, које се обично користе за скенирање кодова су поприлично усавршене, али и даље остаје проблем неравномерног и лошег осветљења. Зато је претходна обрада слике пресудан корак у поступку препознавања и читања QR кода. Резултати предобраде слике имају витални утицај на стопу препознавања кода. Како би се код успешно прочитао, претходна обрада је неопходна.

У оквиру овог рада биће теоријски описан и имплементиран један од могућих приступа предобради слика QR кодова. Комплетан процес састојаће се од неколико кључних фаза које ће бити реализоване кроз неколико мањих корака. Свака фаза ће теоријски бити детаљно објашњена.

Циљ рада је направити низ математичких функција чијом применом на слику QR кода се побољшава квалитет слике и самим тим утиче на ефикасност препознавања. Читав процес представља међукорак који се налази између формирања дигиталне слике и декодирања QR кода.

Имплементација ће бити реализована помоћу програмског језика ***Python3*** и његових библиотека ***NumPy*** и **OpenCV**. У оквиру рада, биће објашњен сваки корак имплементације и биће приказан на неколико примера. Осим тога, рад ће садржати и осврт на проблеме на које је аутор наишао у току имплементације, као и предлоге могућих решења.

2. Дигитална обрада слике

Дигитална обрада слике је доживела импресиван раст у протеклим деценијама, како у погледу теоријског развоја, тако и у примени. Иако је издвојена као посебно научно поље, веома је зависна од технолошких напредака у пољима оптике, сензора и електронике. Поменути раст се огледа у великом броју радова објављених у међународним научним часописима, као и у великом броју специјализованих књига о дигиталној обради слике и рачунарском виду [1].

Појам обраде дигиталне слике се односи на процесе коришћења рачунарских алгоритама у циљу побољшања квалитета дигиталне слике или издвајање неке корисне информације. Поменути процеси у основи представљају обраду сигнала, где је улазни сигнал слика, а излаз може да буде слика или нека карактеристика повезана са том сликом.

На квалитет дигиталне слике највише утичу два фактора. Први фактор се односи на ограничења физичког медија да уопште формира слику доброг квалитета. Последњих година, напретком у пољима оптике и сензора, фотоапарати и камере на мобилним телефонима су поприлично усавршени, па су дигиталне слике које формирају све квалитетније. Без обзира на квалитет физичког медија, још увек је главни проблем препознавања објекта на слици када је позадина сложена и када је неуједначен интензитет светлосног окружења [2]. И то представља други фактор због кога је посебно неопходна обрада слике. Неравномерно осветљење приликом настанка дигиталне слике утицаће на то да се у оквиру слике појаве шумови, одређене геометријске дисторзије, сене, мрље итд.

С обзиром да у реалним условима не може да се гарантује да ће осветљење да буде идеално, од тренутка настанка до коришћења дигиталне слике потребно је да слика прође кроз процес обраде, како би се повећао квалитет. Исто важи и за слике QR кодова о којима је реч у оквиру овог рада. У циљу успешног читања кода, неопходно је да се уклони што више недостатака. Сам код има у оквиру себе имплементиран алгоритам за исправљање грешака који исправља грешке до одређене границе [3]. Утицај лошег осветљења често наруши квалитет слике толико да имплементиран алгоритам није довољан, па је потребно користити неке сложеније математичке методе за побољшавање квалитета у виду предобраде, како би се код успешно прочитао.

3. QR код

Код са брзим одговором, скраћено QR код (*енгл*. Quick Response Code) је врста матричног симбола који је развила јапанска корпорација Denso Wave 1994. године за потребе аутомобилске индустрије [4]. QR код је дводимензионална верзија бар-кода која може скенирањем мобилним уређајем готово тренутно да пренесе широк спектар информација. Њихова основна одлика је веома једноставан и брз запис података, али и читање истих [3].

Слика 1 – QR код сајта Факултета инжењерских наука Универзитета у Крагујевцу

Састоји се од црних квадрата распоређених у квадратну мрежу на белој позадини, коју уређај за обраду слике, као што је дигитална камера може прочитати. Дизајниран је тако да омогућава брзо деокодирање, а намена му је да врло једноставним поступком, скенирањем кода, приступимо информацијама које су кодиране у оквиру кода. Код може да садржи информације до 7089 цифара или 4296 знакова, укључујући знакове интерпункције и посебне знакове [4]. Што се више података додаје, то се величина кода повећава и структура постаје сложенија.

Првобитно је био намењен да помогне у производном процесу праћења возила и ауто делова у аутомобилској индустрији [4], али се убрзо проширио и на остале гране производне индустрије. Корпорација Denso Wave се одрекла права на патент, тако да је QR код постао апсолутно отвореног типа и доступан свима. До данас су QR кодови у великој мери надмашили основну употребу. Развој технике и технологије проширио је могућности QR кодова, као и његову примену у скоро свим областима живота и свакодневнице. У ери интернета и друштвених мрежа највише се користи за директно навођење до одређене веб локације или до профила на социјалним мрежама као што су Инстаграм и Фејсбук. Брзим скенирањем кода помоћу паметног телефона, можемо директно приступити неком сајту или профилу, без да трошимо време на укуцавање адресе или тражења профила. Осим тога, помоћу QR кода могуће је упутити позив, послати предефинисану поруку или мејл [3], додати контакт, конектовати се на WiFi без укуцавања лозинке, додати догађај у календар, прочитати геолокацију, скинути апликацију, чак и вршити плаћање.

У оквиру QR кода могуће је складиштити и PDF документ, слику у разним форматима, песму или било коју другу датотеку [4].

Уз раније поменуте предности, QR кодови имају један велики недостатак. Код је немогуће декодирати директно од стране корисника, пре него што се скенира мобилним уређајем. Поменути недостатак доводи до ситуације где једна легитимна функционалност може бити искоришћена од стране аутора злонамерних програма и зато се QR кодови могу сматрати несигурним. Нападач може направити QR код који садржи Интернет адресу која спроводи корисника до злонамерног програма, притом маскирајући га легитимним конктекстом, као, на пример, лажни оглас на неком популарном сајту [5].

3.1. Елементи QR кода

QR кодови нам се на први поглед могу чинити врло случајни, али се у суштини врло мало међусобно разликују. Иако изгледају као слике издељене у пикселе, сваки од тих квадрата (модула) заправо је маркер (образац) који има неку већу функцију. У QR коду постоји 7 различитих елемената.

1. Позициони обрасци

Сваки QR код се састоји од три позициона обрасца (*енгл*. Finder patterns) који су смештени у три угла и то горњи леви, горњи десни и доњи леви угао (*Слика 2*). Они омогућавају QR читачу да тачно препозна код, односно пронађе ивице кода и прочита модуле са подацима великом брзином, као и да пронађе смер пружања кода [6]. За позиционе обрацце је карактеристично да је однос црно-белих секвенци 1:1:3:1:1 из било ког правца [7]. Захваљујући томе QR код може се читати из било ког угла, што значајно побољшава ефикасност рада.

Слика 2 – Позициони обрасци

2. Тиха зона

Тиха зона (*енгл.* Quite zone) је од виталног значаја за QR код. Она помаже читачу QR кода да изолује код од његове околине [4]. Њен значај је сличан значају осталог белог простора у дизајну QR кода. По стандарду, тиха зона је широка четири модула [8], али у пракси то није увек случај. Тиха зона не би требало да садржи никакве обрасце или структуре који могу збунити читач кода [3].

Слика 3 – Тиха зона

3. Обрасци поравнања

Обрасци поравнања (*енгл.* Alignment patterns) су слични позиционим обрасцима. Разлика је у томе што су обрасци поравнања мањи и налазе се по целом коду. QR кодови који имају верзију 2 или више морају да имају бар један образац поравнања [3]. Поменути обрасци омогућавају QR читачу да исправи изобличења када је код савијен или закривљен [4]. Што више информација складишти код, то је код већи и захтева више образаца поравнања. У зависности од верзије кода јасно је дефинисано колико образаца поравнања је потребно и на којим позицијама се налазе. На слици 4 приказан је QR код верзије 2 који има један образац поравнања.

Слика 4 – Образац поравнања

4. Временски обрасци

Временски образац (*енгл.* Timing pattern) чине наизменично поређани црни и бели модули који помажу QR читачу да утврди колико је велика матрица података у коду, тако што открива положај сваког модула у коду. На основу временских образаца читач може тачно да конфигурише мрежу података [4]. Поменути образац се налази између позиционих образаца, па самим тим помаже у лоцирању позиционих образаца.

Слика 5 – Временски обрасци

5. Информације о верзији

Постоји 40 различитих верзија QR кода [4]. Што је већа верзија то је код већи и садржи више података. Верзија 1 се састоји од 21 x 21 модула, док се верзија 40 састоји од 177 x 177 модула. Повећањем верзије величина кода расте за 4 модула по страни [3]. Најчешће су верзије од 1 до 7 [4]. Обележени делови на слици испод представљају комбинацију црних и белих модула на основу којих QR читач има информацију о верзији кода.

Слика 6 – Информације о верзији

6. Информације о формату

Приликом читања QR кода, прво се читају информације о формату. Узорци формата садрже информацију о томе који се ниво користи за исправљање грешака (4 опције), као и информацију о броју обрасца маске (7 опција), који су неопходни за декодирање кода [3].

Слика 7 – Информације о формату

7. Подаци и модули за исправљање грешака

Централни део QR кода се састоји од блокова за податке и блокова за исправљање грешака. Подаци се састоје од низа редова и колона. Свака ћелија се чува као бинарни број (1 или 0).

Слика 8 – Подаци и модули за исправљање грешака

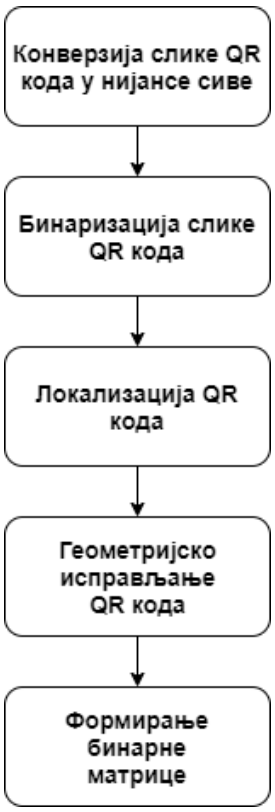
Подаци су кодирани унутар кода и имају могућност корекције грешака. Корекција грешака се обавља помоћу Reed-Solomon алгоритма, који омогућава обнављање оштећених података чак до 30%. Захваљујући корекцији грешака, код може бити делимично оштећен, а да се може у потпуности очитати. QR кодови пружају могућност избора једног од четири нивоа за исправљање грешака (L, M, Q, H)) [3]. Повећањем нивоа за исправљање грешака, смањује се расположиви капацитет за податке кода. Код одабира нивоа исправљања грешака, морају се у обзир узети различити фактори, као што су оперативно окружење и величина QR кода. Најчешће се бира ниво M, помоћу ког се успешно исправљају грешке када је оштећење кода до 15%.

Ниво L	Приближно 7% кодних речи може исправити
Ниво M	Приближно 15% кодних речи може исправити
Ниво Q	Приближно 25% кодних речи може исправити
Ниво H	Приближно 30% кодних речи може исправити

Табела 1 – Нивои исправљања грешака [3]

4. Предобрада слика QR кодова

Предобрада слика QR кодова састоји се од неколико главних фаза, које су имплементиране помоћу више мањих корака. Дијаграм тока обраде приказан је на слици 9.



Слика 9 – Дијаграм тока обраде

Први корак је конверзија слике кода у нијансе сиве, што представља основни корак када је у питању обрада дигиталних слика. Следећи корак је процес бинаризације који се састоји из два корака, један је примена адаптивног алгоритма за тражење вредности прага, док је други сама примена бинаризације на слику коришћењем пронађеног прага. Након тога следе кораци који се односе на локализацију кода. Лоцирају се позициони обрасци, као и граничне линије кода. Поменутим кораком добија се тачна локација и смер простирања кода. Када је код лоциран, уколико је потребно, треба применити геометријско исправљање. У оквиру овог рада, описано је и имплементирано геометријско исправљање применом Афиних трансформација. Последњи корак представља формирање бинарне матрице на основу кода. У оквиру формиране бинарне матрице се налазе сви потребни подаци за декодирање кода.

4.1. Бинаризација

Претварање сиве слике (вишетонска слика) у црно-белу слику, такозвану бинарну слику, је процес бинаризације [9]. Бинаризација дигиталне слике се обично примењује када је потребно одвојити објекат на слици од његове позадине. Најједноставнији начин примене бинаризације је помоћу граничне вредности, односно тражења прага (*енгл.* thresholding). Ако је вредност интензитета пиксела слике већа од прага, тада се тај пиксел конвертује у белу боју, односно добија вредност 255. Слично томе, ако је вредност пиксела мања или једнака прагу, тада ти пиксели постају црни, односно вредност њиховог интезитета постаје 0 [10]. Постоје два типа бинаризације: бинаризација базирана на глобалном прагу и бинаризација базирана на региону. Када је у питању бинаризација базирана на глобалном прагу, тада имамо једну вредност прага, помоћу које бинаризујемо целу слику. Коришћењем једног глобалног прага за целу слику, углавном се губи или потискује локална варијанса слике која можда садржи неке важне информације и садржај [10]. Процес бинаризације базиран на региону функионише тако што се слика подели на више региона и за сваки регион се посебно рачуна праг и на основу тог прага се врши бинаризација тог региона. Када су у питању слике QR кодова, вршење бинаризације на основу једног глобалног прага даје сасвим добре резултате, није потребно имати више регионалних прагова. Проналазак оптималне вредности прага је у генералном случају кључна за бинаризацију. Различити алгоритми се користе у зависности каква је слика и ког типа су оштећења. У овкиру овог рада биће описан и имплементиран адаптивни алгоритам за тражење вредности прага.

4.1.1. Основни алгоритам за тражење прага

Може се приметити да у случају када су објекти на слици доста различити од позадине, односно кад се не стапају са позадином, могуће је користити један глобални праг за процес бинаризације. Основни алгоритам за тражење прага је итеративни алгоритам за адаптивно тражење прага на основу нивоа интензитета слике. Алгоритам се састоји од следећих корака:

1. Насумично одабрати почетни праг T
2. Извршити сегментацију слике на основу прага T. Сегментацијом ћемо добити две групе пиксела: G_1 која се састоји од свих пиксела чији је интензитет већи од прага T и групе G_2 која се састоји од осталих пиксела, односно пиксела чији је интензитет мањи или једнак прагу T.
3. Израчунати средњу вредност интезитета пиксела у свакој од група. Нека m_1 буде средња вредност интензитета пиксела у групи G_1, а m_2 средња вредност интезитета пиксела у групи G_2.

4. Израчунати нову вредност прага преко следеће једначине:

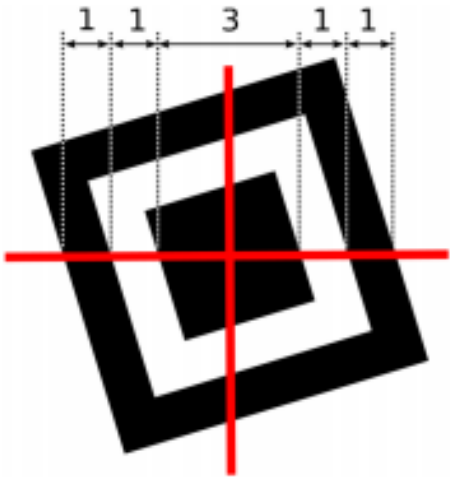
$$T = \frac{1}{2}(m_1 + m_2)$$

5. Понављати кораке од 2. до 4. све док разлика између старог и новог T не буде мања од неког унапред дефинисаног ΔT. [11]

Описан алгоритам ради сасвим добро када постоји јасна граница између објеката и позадине. Параметар ΔT се користи за контролу броја итерација и директно утиче на брзину извршавања алгоритма. У суштини, што је веће ΔT, то ће бити мање итерација и самим тим ће алгоритам имати краће време извршавања. Постоји још једно ограничење алгоритма, иницијална вредност прага мора бити изабрана тако да буде већа од минималног, а мања од максималног нивоа интензитета слике [11]. Средња вредност нивоа интензитета слике је сасвим добар избор за иницијалну вредност прага T.

4.2. Локализација

QR код пружа могућност брзог читања у свим правцима и то се остварује захваљујући обрасцима за откривање положаја (позициони обрасци) који се налазе у три угла кода. Ови обрасци гарантују стабилно читање кода великом брзином, заобилазећи негативне ефекте позадинских сметњи [12]. Према међународном стандарду кодирања QR кода, сваки од три позициона обрасца има особину да је при било којој ротацији кода могуће повући линију кроз центар позиционог обрасца, тако да се црне и беле секвенце које леже на њој смењују у односу 1:1:3:1:1 [7].



Слика 10 – Структура позиционог обрасца [3]

Локализација QR кода се односи на процес проналажења сва три позициона обрасца, како би се на основу њихових положаја одредила област простирања QR кода и идентификовали кодирани подаци. У циљу проналаска позиционих образаца, потребно је скенирати бинарну слику хоризонтално, ред по ред, од врха до дна.

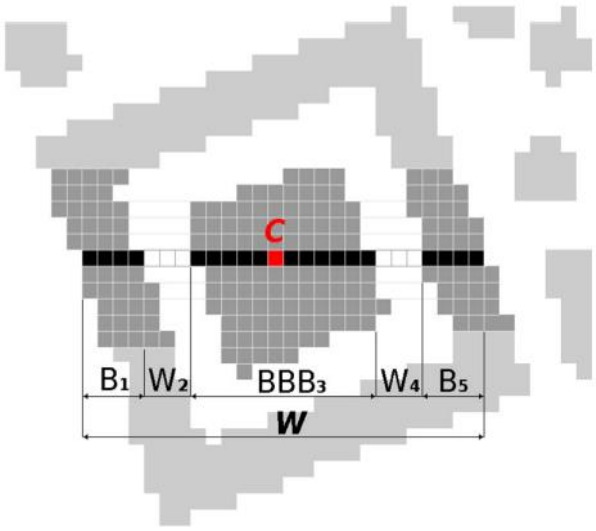
Приликом скенирања слике траже се узастопне секвенце црних и белих тачака у низу које су у односу 1:1:3:1:1 (B_1: W_2: BBB_3 : W_4 : B_5 где B_1, BBB_3 и B_5 представљају црне секвенце, док су W_2 и W_4 беле секвенце које се налазе између црних секвенци). Како дигиталне слике нису идеалне, тако ни однос који се тражи није апсолутно прецизан, већ има одређених одступања, па је потребно у обзир узети и одређени праг толеранције. Сматра се да је однос црних и белих секвенци у поменутом односу уколико су задовољени следећи услови [3]:

$$B_1, W_2, W_4, B_5 \in (w - 2\,, w + 2)$$

$$BBB_3 \in (3w - 2,\ 3w + 2), \text{ где је } w = \frac{B_1 + W_2 + BBB_3 + W_4, + B_5}{7}$$

$$BBB_3 > max\,(B_1 + W_2,\ W_4 + B_5)$$

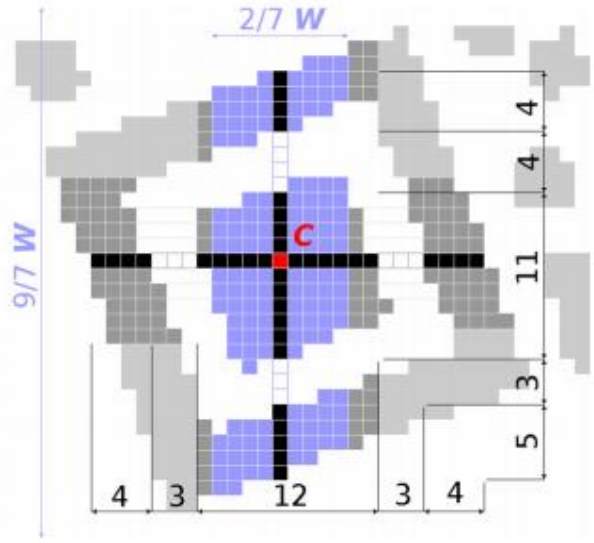
Свако пронађено подударање представља једног могућег кандидата за позициони образац. За сваког кандидата потребно је запамтити положај центра, односно координате централне тачке у средишњем црном блоку, као и укупну ширину свих блокова. Укупна ширина се добија сабирањем црних и белих секвенци.



Слика 11 - Кандидат за позициони образац по хоризонталној оси [3]

Након формирања листе могућих кандидата, потребно је груписати међусобно најближе кандидате. Два кандидата спадају у исту групу ако је њихово међусобно растојање највише три тачке хоризонтално и највише 3/7 W вертикално, где је W укупна ширина кандидата [3]. Нове координате центра C, као и нова ширина W добијају се као средња вредност груписаних кандидата. Након груписања и проналажења средње вредности од сваке групе се добија један кандидат.

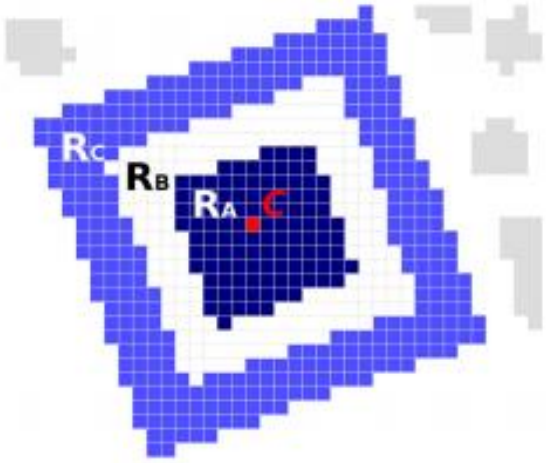
Кандидат за позициони образац је валидан, ако му је и у вертикалном правцу однос црних и белих секвенци, такође, 1:1:3:1:1. Провера се врши по истом принципу. Разлика се огледа само у начину скенирања. Приликом вертикалне провере скенира се колона, по колона, од крајње леве, до крајње десне колоне. Скенирање слике по колонама може се имплементирати на исти начин као и по редовима, само је потребно да се матрица улазне слике транспонује. Кандидати који задовоље обе провере, и вертикалну и хоризонталну, представљају валидне кандидате за позиционе обрасце, док остали бивају елиминисани.



Слика 12 – Валидан кандидат за позициони образац [3]

Сваки позициони образац чини квадрат који се састоји од 3 x 3 модула (Ra) који је смештен у оквир величине 7 x 7 модула (Rc). Због те правилности, сваки индивидуални кандидат мора да задовољи и следеће услове [3]:

- област (Ra) < област (Rb) < област (Rc) и област (Rb) / област (Ra) < 3 и област (Rc) / област (Ra) < 4
- 0.7 < ширина (Rb) / висина (Rb) < 1.3 и 0.7 < ширина (Rc) / висина (Rc) < 1.3
- растојање (центар (Ra), центар (Rb)) < 3 и растојање (центар (Ra), центар (Rc)) < 3



Слика 13 – Повезане области позиционог обрасца [3]

Како би се одредиле карактеристике области Ra, Rb и Rc потребно је применити Поплава алгоритам.

4.2.1. Поплава алгоритам

Поплава алгоритам (*енгл*. Flood Fill) је алгоритам који се користи за одређивање ограничене површине повезане са датом тачком у вишедимензионалном низу. Алгоритам је поприлично једноставан и чине га следећи кораци [13]:

1. Поставити положај почетне тачке
2. Одабрати опције за смер кретања: четири смера (север, исток, југ, запад) или осам смерова (север, исток, југ, запад, североисток, северозапад, југоисток, југозапад)
3. Изабрати тражену боју и боју за попуњавање (када алгоритам наиђе на тражену боју, мења је бојом за попуњавање)
4. Кретати се у изабраним смеровима
5. Ако је тачка на коју се наилази тражене боје, заменити је бојом за попуњавање
6. Понављати кораке 4 и 5, док се не обиђе свака тачка у ограниченој области која има тражену боју

Попуњавање се започиње од центра C (који се налази у оквиру црног централног квадрата Ra) и наставља се кроз бело окружење (Rb), па све до црне области која чини оквир позиционог обрасца (Rc). Током попуњавања региона, израчунавају се дескриптори горе поменутих региона:

- област (M_{00})
- центар (M_{10} /M_{00}, M_{01} / M_{00}), где су M_{00}, M_{01}, M_{10} моменти слике

4.2.2. Момент слике

Моменти се широко користе за опис математичких функција због њихове нумеричке једноставности и бројних физичких интерпретација. Концепт момента се већ дуги низ година користи у широком спектру области, од механике и статистике, па све до рачунарског вида и дигиталне обраде слике. Описивање слике на основу момента, уместо неких других карактеристика значи да је предност дата глобалним својствима слике, у односу на локална својства.

За дводимензионалну континуалну функцију f(x, y), момент се дефинише као [14]:

$$M_{ij} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^i y^j f(x,y) dx dy \text{ , за i, j} = 0, 1, 2$$

С обзиром да се слика састоји од пиксела, потребан је дискретан начин размишљања за описивање момента. Процесом дискретизације, интеграл постаје сума [14]:

$$M_{ij} = \sum_{i=0}^{x} \sum_{j=0}^{y} x^i y^j f(x,y), \text{ где је } f(x,y) \text{ интензитет пиксела}$$

Када се тражи момент одређене области бинарне слике, тражи се нулти момент, тј. M_{00} [15]. У том случају x и y немају утицаја на суму, тако да сума постаје:

$$M_{00} = \sum_{i=0}^{x} \sum_{j=0}^{y} f(x,y)$$

Као што се може закључити на основу обрасца, нулти моменат бинарне слике представља укупну суму пиксела у одређеној области.

Прва два момента:

$$M_{10} = \sum_{i=0}^{x} \sum_{j=0}^{y} x f(x,y)$$

и

$$M_{01} = \sum_{i=0}^{x} \sum_{j=0}^{y} y f(x,y)$$

представљају центар масе слике $f(x,y)$. Центар масе је тачка где би сва маса могла бити концентрисана без промене првог момента око било које осе [15]. У дводимензионалном случају, помоћу момената, координате центра се могу израчунати као:

$$x' = \frac{M_{10}}{M_{00}}, , y' = \frac{M_{01}}{M_{00}}$$

Поплава алгоритам се у оквиру овог рада користи да позициони образац раздвоји на области Ra, Rb и Rc, како би могли да се израчунају тражени моменти области.

Осим провере да ли су кандидати за позиционе обрасце валидни, помоћу израчунатих момената сваке области појединачно могу се прецизније одредити координате центра C, као и ширина једног модула MW и то помоћу следећих једначина [3]:

$$C = \left(\frac{M_{10}(Ra) + M_{10}(\mathrm{Rb}) + M_{10}(\mathrm{Rc})}{M_{00}(\mathrm{Ra}) + M_{00}(\mathrm{Rb}) + M_{00}(\mathrm{Rc})}, \frac{M_{10}(Ra) + M_{10}(\mathrm{Rb}) + M_{10}\ (\mathrm{Rc})}{M_{00}(\mathrm{Ra}) + M_{00}(\mathrm{Rb}) + M_{00}(\mathrm{Rc})}\right),$$

$$MW = \frac{\sqrt{M_{00}(\mathrm{Ra}) + M_{00}(\mathrm{Rb}) + M_{00}(\mathrm{Rc})}}{7}$$

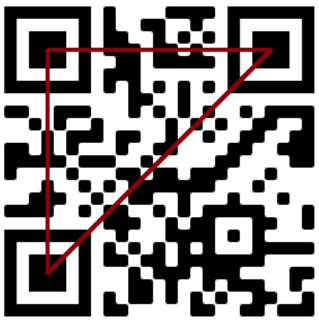
Област региона Ra је 9*MW* (3 x 3 *MW*), а област региона је Rc је 24*MW*.

Ако се крене од претпоставке да на слици постоји један QR код, велика је вероватноћа да ће постојати тачно три кандидата за позициони образац. Иако су изумитељи QR кодова за позициони образац узели однос који има најмању вероватноћу појављивања на сликама, свакако није немогуће да се појави и лажни кандидат, па да их има више од три. Како би се установило која су три кандидата сигурно центри позиционих образаца потребно је конструисати правоугли једнакокраки троугао који спаја поменуте центре. Конструисањем троугла, осим што се елиминишу лажни кандидати, тачно се лоцира које координате се односе на који позициони образац, па се самим тим једнозначно одређује положај кода.

Троугао се конструише тако што се прво креира матрица међусобних растојања свих кандидата. За сваку комбинацију се проверава да ли може да се конструише поменути троугао на основу следећих услова [3]:

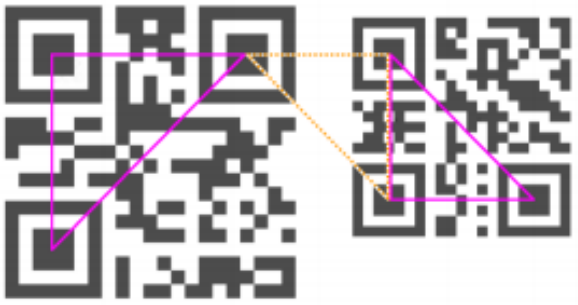
- Разлика између дужина катета да буде мања од 14
- Разлика између дужине реалне и теоријске хипотенузе да буде мања од 12

За мање оштећене и искривљене слике, довољна је и мања толеранција.



Слика 14 – Троугао који спаја позиционе обрасце

Претходно описано конструисање троугла је рађено уз претпоставку да на слици има један QR код. Уколико на слици има више кодова, потребно је извршити додатне провере. Поменути метод ће пронаћи правоугле једнакокраке троуглове, али у случају да су кодови међусобно близу, постоји могућност да темена троугла припадају различитим кодовима (*Слика 15*). У том случају потребно је искористити друге карактеристике QR кода.



Слика 15 – Случај кад темена троуглова припадају различитим кодовима [3]

Једна провера се односи на присуство тихе зоне (*енгл.* Quiet Zone). Као што је раније поменуто, тиха зона је бели оквир који се се пружа око целог кода и изолује код од његове околине. Присуство тихе зоне може се доказати тако што се провери да ли паралелно са катетама раније поменутог троугла, на растојању дужине три модула, постоји линија са само белим тачкама. Уколико не постоји, конструисани троугао се може одбацити.

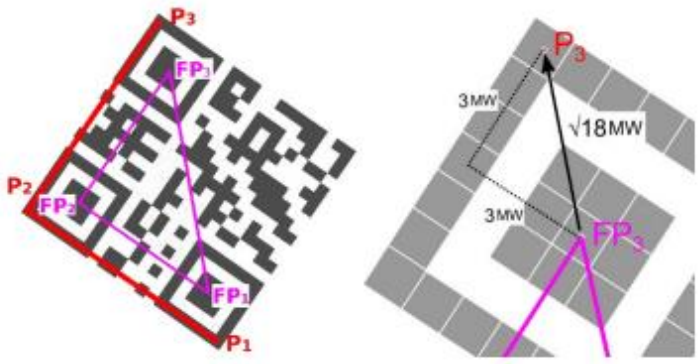
Слика 16 –Тиха зона око QR кода [3]

Друга провера се односи на временске обрасце QR кода. Временски обрасци спајају унутрашње углове позиционих образаца и то наизменичним појављивањем белог и црног модула. Провером временских образаца се, такође, могу елиминисати погрешно конструисани троуглови.

Слика 17 – Временски образац у унутрашњости QR кода [3]

Темена троугла представљају, као што је раније поменуто, централне тачке позиционих образаца. Како би се пронашао гранични оквир QR кода, потребно је проширити троугао тако да катете проширеног троугла пролазе кроз ивичне модуле кода. На пример, померај између FP_3 и P_3, може бити изражен као [3]:

$$P_3 = FP_3 + \frac{FP_3 - FP_1}{|FP_3, FP_1|} MW\sqrt{18},$$ где је MW ширина модула



Слика 18 – гранични оквир [3]

Аналогно томе, могу се пронаћи и остала два темена проширеног троугла. Када се одреди гранични оквир, врло лако може се одредити и пројекција четврте тачке кода, уколико је потребна за неку трансформацију. Осим тога, растојање између проширених тачака представља ширину QR кода и може се користити приликом геометријског исправљања о коме ће бити реч у наредном поглављу. Ширина кода се може одредити и без споменутог корака, тако што се на растојање између централних тачака позиционих образаца дода ширина седам модула MW, али овим кораком се добија нешто прецизније решење.

4.3. Геометријско исправљање

Слика QR кода може се појавити са одређеним степеном геометријског изобличења и деформација због угла из ког је скениран QR код, као и због неуједначеног интензитета светлосног окружења. Геометријско исправљање се односи на процес враћања QR кода у квадратни облик и то се може учинити помоћу Афиних трансформација [7].

4.3.1. Афине трансформације

Афине трансформације (*енгл.* Affine Transformations) представљају скуп геометријских трансформација. Оне помажу у модификовању геометријске структуре слике, чувајући паралелизам линија, али не и дужине и углове. Иако не чувају дужине и углове, задржавају колинеарност и односе између дужина. Ове трансформације се ослањају на матрице управљања ротацијом, транслацијом, скалирањем и смицањем. Општи облик Афине трансформације за појединачну тачку гласи [16]:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{M} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

где су x' и y' координате пиксела након трансформације, x и y координате пиксела пре трансформације, а M матрица неке од Афиних трансформација димензије 3 x 3. Као што је поменуто раније, у Афине трансформације спадају транслација, ротација, скалирање и смицање и свака од њих има своју карактеристичну матрицу.

Транслација је функција која помера сваку тачку за константно растојање у одређеном смеру. Приликом транслације свака тачка на слици прати паралелни пут и не долази до ротације [17]. У оквиру матрице транслације дефинише се померај по хоризонталној оси (dx) и вертикалној оси (dy). Матрица транслације гласи [16]:

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

Ротација је кружна трансформација која се може имплементирати око тачке или око осе, потребно је само задати угао ротације. У оквиру матрице ротације се дефинише угао ротације θ. Матрица ротације гласи [16]:

$$\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Скалирање је линеарна трансформација која увећава или смањује предмете на слици за фактор повећања који је исти у свим правцима. У основи је зумирање или смањење слике. Параметар Sx представља фактор повећања по хоризонталној оси, а Sy представља фактор повећања по вертикалној оси. Матрица скалирања гласи [16]:

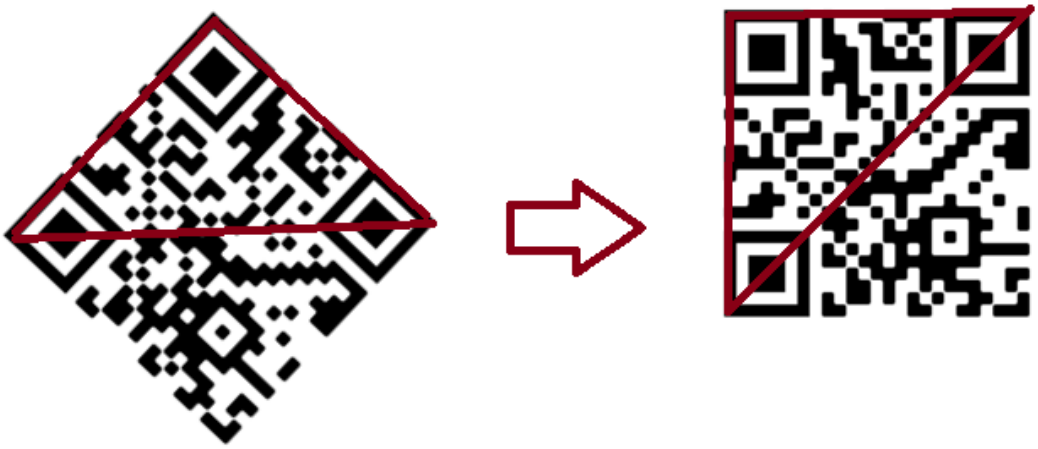
$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Смицање је функција која помера сваку тачку са константним растојањем у хоризонталном и/или вертикалном правцу. Применом функције смицања дужине остају исте, али не и углови [17]. Параметри матрице hx и hy представљају факторе смицања по хоризонталној и вертикалној оси, респективно. Матрица смицања гласи [16]:

$$\begin{bmatrix} 1 & hx & 0 \\ hy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Иза Афиних трансформација стоји доста апстрактне математике, али када је реч о обради дигиталне слике то се своди на линеарну трансформацију која је резултат једне или више манипулација попут транслације, ротирања, скалирања или смицања. Комбиновање једне или више различитих трансформација се реализује међусобним множењем горе поменутих матрица, након чега се добија једна матрица трансформације. Може се приметити да је трећи ред код свих матрица трансформације 0 0 1, тако да ће и након комбиновања остати исти. С обзиром да је матрица трансформације иста за сваку тачку, а постоји шест непознатих, за одређивање параметара матрице потребне су три неколинеарне тачке. Захваљујући претходном кораку, у оквиру кога су лоцирани позициони обрасци, познате су координате три тачке, колико је и потребно да се може користити формула Афине трансформације са шест параметара за геометријско исправљање [7].

У циљу одређивања параметара матрице потребне за Афину трансформацију, потребно је решити систем шест једначина са шест непознатих. Систем се формира помоћу шест тачака. За прве три тачке могу се узети углови три позициона обрасца, док друге три представљају жељене пројекције тих тачака. Пројекције тачака се могу одредити тако што се једна тачка оригиналне слике мапира у исту ту тачку, док се друге две одређују на основу ње. За тачку која остаје иста и након трансформације бира се угао позиционог обрасца који се налази у левом горњем углу QR кода, када је код окренут како треба, односно образац који се налази наспрам хипотенузе правоуглог једнакокраког троугла који формирају позициони обрасци. Друге две тачке се добијају померањем по x, односно у оси, за ширину QR кода.



Слика 19 – QR код пре и после Афиних трансформација

Проблем код Афиних трансформација је тај што се приликом трансформације, неки пиксели оригиналне слике пројектују у исту тачку нове слике, док се неки пиксели уопште не пројектују, односно изгубе се у току трансформације. Због те појаве, слика након трансформације има одређене пукотине, које се манифестују појавом црних тачака или линија. Како би се попуниле пукотине, потребно је урадити инверзну трансформацију над излазном сликом, да би се нашли одговарајући пиксели оригиналне слике [18]. Координате оригиналне слике могу се добити тако што се матрица трансформације транспонује и помножи са координатама слике након трансформације:

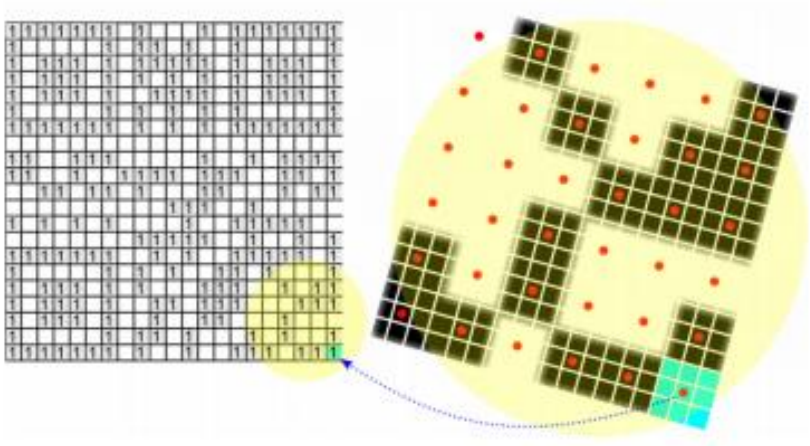
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{M^T} \times \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Када се пронађу координате оригиналне слике, потребно је одредити којом методом изабрати вредност интензитета пиксела за попуњавање одређене пукотине. Када су у питању неке комплексније слике, за бирање вредности интензитета пиксела се користе интерполацијске методе. Такве методе су рачунарски захтевне и доста успоравају цео процес. Због природе QR кода, а и начина како се манифестују пукотине могуће је користити методу најближег суседа која представља најједноставнију и најмање рачунски захтевну методу, а даје сасвим добре резултате. Метода најближег суседа се заснива на томе да се вредности пиксела у излазној слици придружи интензитет пиксела оригиналне слике након инверзне трансформације [18].

4.4. Формирање бинарне матрице

QR код је дводимензионална матрица, што значи да садржи податке у оба правца, вертикалном и хоризонталном. Читач QR кода анализира код тако што разложи целу мрежу на појединачне модуле и додељује им вредност на основу тога да ли је модул црне или беле боје. Црни модул репрезентује бинарно 1, док бели модул репрезентује бинарно 0. Затим групише модуле како би креирао веће обрасце и прочитао податке [19]. У циљу успешног декодирања кода, потребно је слику кода конвертовати у облик који је разумљив читачима (скенерима) QR кодова.

Слику кода потребно је конвертовати у квадратну бинарну матрицу, такву да сваки елемент матрице представља један модул кода [3]. Након претходних корака обраде, сваки модул кода се састоји од секвенце црних или белих тачака. Следећи корак се огледа у томе да се секвенца тачака који чине један модул мапира у само један бинарни број, 1 или 0, у зависности од тога да ли је секвенца формирана од црних или белих тачака. На слици 20 приказано је мапирање једног сегмента кода.



Слика 20 – Мапирање слике QR кода у бинарну матрицу [3]

За формирање бинарне матрице, неопходно је да код буде квадратног облика [3]. Димензија формиране матрице је AxA, где A представља број модула у једном реду. Такође, због неопходног квадратног облика, A је и број модула који чине једну колону.

5. Имплементација

Процес предобраде слике QR кода имплементиран у оквиру овог рада одвија се у више фаза:

- Конверзија слике у нијансе сиве
- Процес бинаризације
- Хоризонталана претрага кандидата за позиционе обрасце
- Вертикална претрага кандидата за позиционе обрасце
- Груписање најближих кандидата
- Израчунавање средње вредности по свакој групи кандидата
- Конструисање једнакокраког правоуглог троугла који формирају позициони обрасци
- Израчунавање момента и ажурирање положаја центра позиционог обрасца
- Конструисање проширеног троугла (лоцирање граница QR кода)
- Геометријско исправљање – примена Афиних трансформација
- Примена инверзне Афине трансформације
- Формирање бинарне матрице од QR кода

Функције предобраде су имплементиране помоћу програмског језика ***Python***, коришћењем библиотека ***NumPy, OpenCV*** и ***Python Math Library***.

У оквиру овог одељка, биће објашњен сваки корак имплементације уз визуелни приказ. На примерима ће бити објашњено и приказано како сваки корак појединачно утиче на оригиналну слику. Осим тога, биће приказани одређени недостаци имплементираних метода, објашњене ситуација када се ти недостаци појављују, као и предлог како их уклонити.

На самом почетку потребно је учитати оригиналну слику на којој се налази QR код. То се може урадити помоћу *imread()* функције из ***OpenCV*** библиотеке. Оригинална слика, односно слика пре било какве обраде, приказана је на слици 21.

Слика 21 – Слика QR кода пре обраде

Први корак представља конверзију слике у 256 нијанси сиве. Када је реч о сликама QR кода, пребацивање слике у нијансе сиве не утиче на слику, а обрада постаје драстично мање комплексна. Како изгледа слика након првог корака приказано је на слици 22.

Слика 22 – Слика QR кода након пребацивања у нијансе сиве

Наредни корак је процес бинаризације. У оквиру овог процеса итерира се кроз целу слику, и сваки пиксел чији је интензитет мањи од вредности прага (*енгл.* threshold), постаје 0, док сваки који је већи или једнак вредности прага, постаје 255. На тај начин је вишетонска слика пребачена у црно-белу слику, односно бинарну слику.

Генерално за процес бинаризације је битно изабрати добру вредност прага, али када је реч о QR кодовима, који су налик на шаховску таблу и у основи су већ донекле бинарни, довољно је изабрати средњу вредност интензитета пиксела. У оквиру овог рада је свакако имплементиран адаптиван алгоритам за тражење прага, иако није неопходан у овом конкретном случају, али свакако може само да даје боље резултате у случају неких евентуалних оштећења. Пребацивање слике у бинарну представља основу за даљу обраду. Слика у бинарном облику нам је потребна за рачунање односа црних и белих секвенци, како би лоцирали позиционе обрасце, а на основу њих и цео код. Како изгледа оригинална слика након процеса бинаризације приказано је на слици 23.

Слика 23 – Слика QR кода након процеса бинаризације

Као што се може приметити на слици, након процеса бинаризације, код, као и остатак слике, је изгубио правилне ивице. Разлог томе је баш то што је слика сад бинарна и има само две вредности интензитета. Када је имала 256 нијанси сиве, тада је постојао постепен прелаз који људском оку даје привид правилне линије, док кад има две вредности, сваки прелаз је израженији. Поменута појава не ремети у великој мери процес обраде, односно наредни кораци су имплементирани на начин да ова појава не утиче на даљи процес.

Наредни кораци, све до геометријског исправљања, односе се на лоцирање кода. Прво је потребно наћи локације сва три позициона обрасца. Како би нашли кандидате за позиционе обрасце прво вршимо хоризонталну претрагу. Хоризонтална претрага се односи на итерирање кроз слику, ред по ред, и тражење низа белих и црних секвенци у односу 1:1:3:1:1, што представља основну карактеристику позиционог обрасца. Овај корак се реализује тако што се рачуна број узастопних пиксела истог интензитета. Док се рачуна сума узастопних пиксела, памте се и координате почетка сваке секвенце.

Поменути подаци су потребни за проналажење координата централне тачке позиционог обрасца. На сваку ивицу, сума се додаје у низ, а бројач се рестартује. Када низ има пет чланова, проверава се да ли први члан представља црну секвенцу, уколико да, проверава се да ли су испуњени услови раније описани у оквиру поглавља *Локализација*. Уколико услови јесу испуњени, израчунавају се координате центра и укупна ширина. Након тога, брише се први члан низа и наставља се исти процес кроз целу слику. На слици 24 означени су центри пронађених кандидата након хоризонталне претраге.

Слика 24 – Слика QR кода са означеним кандидатима за позиционе обрасце након хоризонталне претраге

Након хоризонталне претраге, потребно је урадити и вертикалну, како би се елиминисали лажни кандидати. Вертикална претрага се реализује на исти начин као и хоризонтална, само се у случају вертикалне итерира по колонама, а не по редовима. Како би се итерирало по колонама, потребно је само на улаз довести транспоновану матрицу бинарне слике.

Слика 25 – Слика QR кода са означеним кандидатима за позиционе обрасце након хоризонталне и вертикалне претраге

Након пронађених хоризонталних и вертикалних кандидата, потребно је груписати најближе кандидате. Кандидати спадају у исту групу ако нису више од три тачке хоризонтално и 3/7 од укупне ширине кандидата вертикално, удаљени. Након груписања, потребно је наћи средњу вредност координата центра, као и средњу вредност укупне ширине, сваке групе појединачно. Следећи корак је провера који су све кандидати пронађени и у хоризонталној и у вертикалној претрази. Сви кандидати који се појављују у само једној претрази, бивају елиминисани, јер нису у складу са особином позиционог обрасца да је из свих праваца однос црних и белих секвенци 1:1:3:1:1.

Комплетна имплементација је рађена под претпоставком да на слици постоји само један QR код. У том случају, потребно је да остану само три кандидата за позиционе обрасце. Како би се проверило која три, у случају да их има више, тражи се комбинација кандидата који формирају правоугли једнакокраки троугао. Прво се направи матрица међусобних растојања. На основу растојања, нађу се три кандидата чија су међусобна растојања таква да формирају поменути троугао, уз одређена одступања због несавршености дигиталне слике. Уколико је постојало више од три кандидата, сви остали се елиминишу.

Горе описан корак се не прескаче ни кад има тачно три кандидата, јер служи и за одређивање за сваког кандидата појединачно, на који тачно позициони образац се односи. На основу тога може се једнозначно одредити простирање QR кода.

Слика 26 – Слика QR кода са означеним троуглом који спаја позиционе обрасце

Комплетна обрада се врши на бинарној слици, само су конструисани троуглови приказани на оригиналној слици.

Следећи корак представља прецизније одређивање положаја центра позиционих образаца уз помоћ момента слике и алгоритма поплаве. Како би се прецизније одредиле координате центра, потребно је израчунати момент за сваку од области позиционог обрасца (црни квадрат у средини, бели оквир, спољни црни оквир). Код бинарних слика нулти момент се рачуна као број пиксела у области. Први момент се рачуна тако што се сума пиксела у једном реду помножи са координатом тог реда и дода укупној суми. Други момент се рачуна на исти начин, само што уместо реда иде се по колонама. Ажурирана вредност координате x је однос збира првих момента свих области и збира нултих момента свих области, док је ажурирана вредност координате у однос збира других момента и збира нултих момента.

За раздвајање регија, како би се израчунали моменти, користи се алгоритам поплаве. Он је имплементиран тако да креће од центра позиционог обрасца и све своје црне суседе боји у једну нијансу сиве. Након тога прелази на први бели пиксел који се налази у области око црног средишњег дела. Наставља да боји све своје беле суседе у другу нијансу сиве. Када заврши, прелази на први пиксел у црном оквиру и понавља исти процес, само са другом нијансом сиве. Како то изгледа приказано је на слици 27. Алгоритам поплава је рекурзиван алгоритам и потребан је велики стек, тако да је потребно уз помоћ *Sys* библиотеке повећати лимит стека.

Слика 27 – Слика QR кода након примене алгоритма поплаве

Сада, кад су све регије другачије означене, потребно је само итерирати кроз целу слику и израчунати моменте. Након израчунавања момената сваке области појединачно, може се прецизније одредити положај центра сваког од позиционог обрасца. Такође, у оквиру овог корака се рачуна и средња вредност ширине модула.

Уколико је слика оштећена у пределу позиционих образаца, на пример, има беле пикселе у оквиру црне регије, могуће је да алгоритам поплаве не успе да раздвоји регије, па самим тим израчунавање момената неће бити могуће. Алгоритам се може донекле оптимизовати уколико се врше додатне провере да ли одређени пиксели треба ту да се налазе, али то није имплементирано у оквиру овог рада. У већини тест примера, алгоритам се понаша у складу са захтевима, тако да није било потребна додатна оптимизација.

Следећи корак је тражење граница QR кода. Границе се налазе тако што се прошири раније конструисан правоугли једнакокраки троугао. Како би се задржао угао простирања, потребно је конструисати вектор, који спаја центар позиционог обрасца и центар модула у самим угловима кода. Правац вектора који је потребно конструисати када су у питању горњи десни и доњи леви позициони образац лежи на хипотенузи. Када је у питању горњи леви позициони образац, потребно је конструисати вектор чији се правац поклапа са висином која спаја његов центар са хипотенузом. Смер вектора је у сва три случаја је ка ободу кода, док је интензитет једнак дужини хипотенузе правоуглог троугла, чије су катете једнаке тростукој средњој ширини модула. Применом Питагорине теореме, добија се да је интензитет вектора једнак $\sqrt{18}$ пута средња ширина модула.

Слика 28 – Слика QR кода са означеним проширеном троуглом (љубичастом бојом)

Следећи корак представља геометријско исправљање. Као што је раније описано, поменути корак служи да коду врати квадратни облик, уколико га је изгубио приликом настанка дигиталне слике, као и да ротира код уколико је то потребно. Геометријско исправљање у оквиру овог рада имплементирано је помоћу Афиних трансформација. За примену Афиних трансформација, прво је потребно изабрати шест тачака. Прве три тачке су тачке оригиналне слике и налазе се у три угла кода, а друге три су жељене пројекције прве три тачке на излазној слици. Тачке пројекције су изабране тако да тачка која се налази у углу позиционог обрасца у горњем левом углу се мапира у исту тачку на излазној слици, док се остале две пројектују у односу на њу.

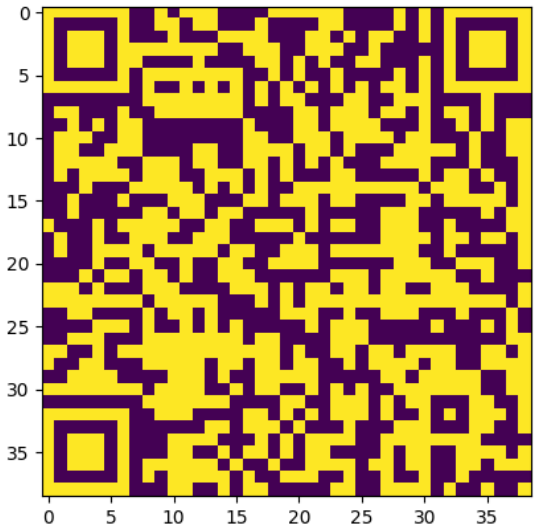
Пројекција тачке позиционог обрасца који се налази горе десно добија се тако што се по хоризонтали дода заокружена дужина катете раније формираног проширеног троугла уз додатак од ширине једног модула, пошто се тачке налазе у центру модула на угловима, а не у самим угловима, док вертикална координата остаје иста. На сличан начин се добија и пројекција тачке позиционог обрасца који се налази у доњем левом углу, само се у том случају горе поменуто растојање додаје по вертикали, а хоризонтална координата остаје иста. Након што су одређене све тачке, потребно је решити систем једначина и на основу тога одредити матрицу трансформације, која се примењује на све тачке бинарне слике. Како изгледа код након Афиних трансформација приказан је на слици 28.

Слика 29 – Слика QR кода након Афиних трансформација

Као што се може приметити, слика кода након Афиних трансформација има по целој својој површини линије које пре овог корака није имала. Разлог ове појаве, као што је раније споменуто, је тај што се приликом мапирања тачака, неке тачке оригиналне слике мапирају у исту тачку у излазној слици, чак и више пута, док се неке не мапирају уопште. Зато настају пукотине које се манифестују појавом линија. Попуњавање пукотина је имплементирано помоћу инверзне Афине трансформације и методе најближег суседа. За инверзну Афину трансформацију потребно је прво транспоновати матрицу трансформације. Након тога је потребно том матрицом помножити сваку тачку излазне слике. Множењем се добијају координате оригиналне слике које је прво потребно заокруглити на цео број. Након тога се узима вредност пиксела који се налази на добијеним заокругљеним координатама и њоме попуњава пукотина излазне слике. Описан процес представља методу најближег суседа.

Слика 30 – Слика QR кода након инверзне Афине трансформације

Последњи корак представља формирање бинарне матрице која се користи за декодирање кода. Матрица се формира тако што се итерира кроз цео код почев од горњег десног угла, односно средишта модула у самом углу. Корак итерације је заокругљена средња ширина модула. Матрица је на почетку празна и при сваком кораку се додаје вредност 0 или 1. При свакој итерацији проверава се да ли је вредност интензитета на тим координатама једнака 0 или 255. Уколико је 0, тада се у матрицу додаје 1, а уколико је 255, додаје се 0. Како изгледа када се прикаже формирана бинарна матрица помоћу библиотеке *Matplotlib* приказано је на слици 31.

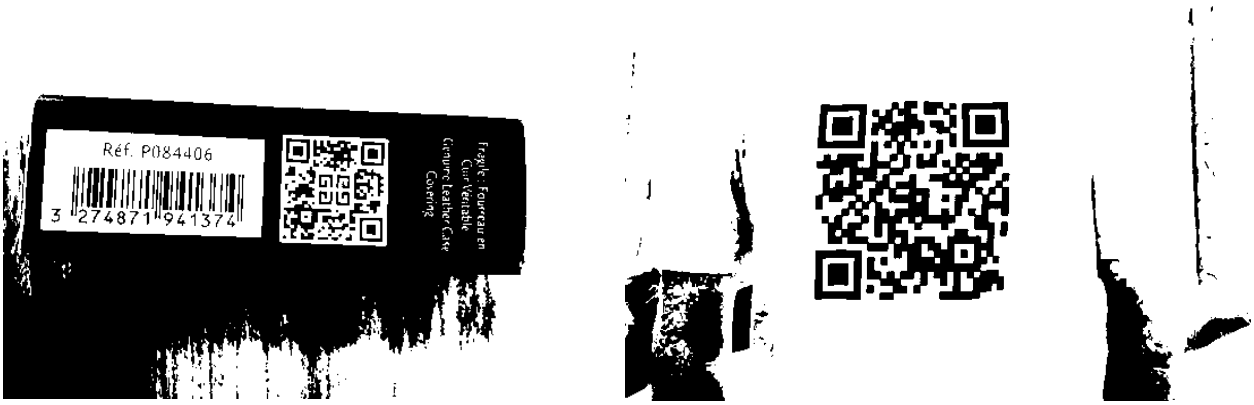


Слика 31 – Приказ бинарне матрице формиране од QR кода

Може се приметити да је приказана бинарна матрица својом структуром доста слична коду од кога је формирана. Када су у питању неки кодови, то није случај, већ има одређених одступања. На одступање утичу оштећења кода која су настала приликом формирања дигиталне слике, а нису уклоњени приликом описане предобраде. Нека мања одступања не утичу на декодирање кода, јер QR код је направљен тако да може опорави код који има до 30% оштећења.

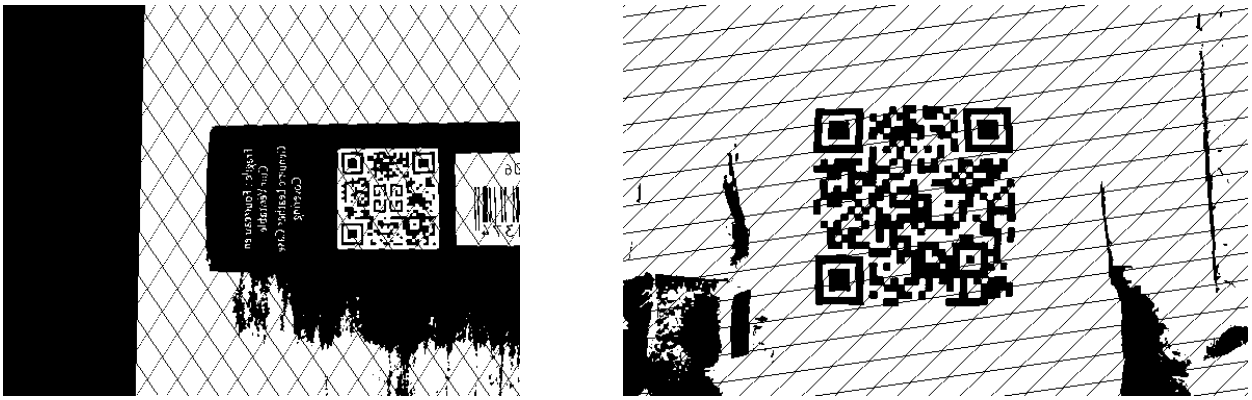
На још неколико примера биће приказани само основни кораци предобраде слика QR кодова имплементираних у оквиру овог рада: процес бинаризације, лоцирање позиционих образаца, геометријско исправљање (Афина и инверзна Афина трансформација) и формирање бинарне матрице.

Слика 32 – Слике QR кодова пре обраде



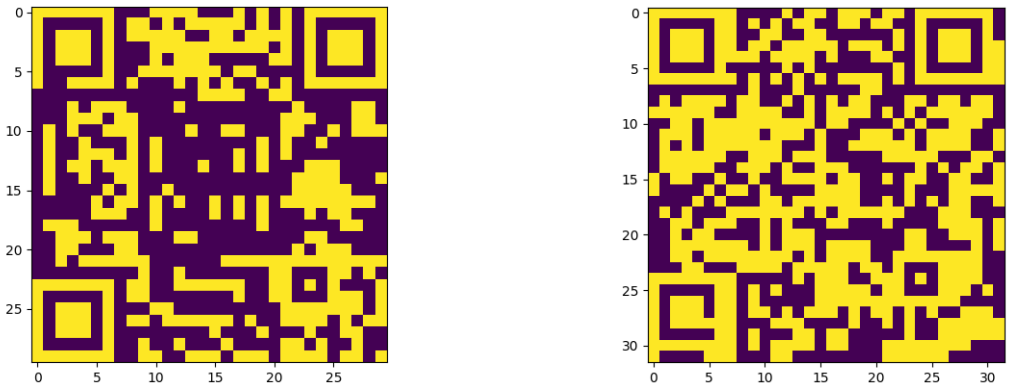
Слика 33 – Слике QR кодова након процеса бинаризације

Слика 34 – Слике QR кодова са лоцираним позиционим обрасцима



Слика 35 – Слике QR кодова након Афине трансформације

Слика 36 – Слике QR кодова након инверзне Афине трансформације



Слика 37 – Приказ бинарних матрица формираних од QR кода

6. Закључак

У оквиру рада је предложен и имплементиран један од могућих приступа предобраде слика QR кода у циљу ефикаснијег препознавања. Предложен приступ користи карактеристичне позиционе обрасце, који се налазе у три угла кода, за лоцирање и идентификацију кода. За исправљање дисторзије коришћене су Афине трансформације.

Описана предобрада је погодна за локализацију и геометријско исправљање појединачних кодова на сликама ниске резолуције. За успешно декодирање кода, локализација кода представља неопходан, али не и довољан услов. У условима неравномерног осветљења, доћи ће до појаве одређених дисторзија на слици, зато је потребно геометријском трансформацијом исправити код и вратити га у првобитни квадртани облик.

Предложена имплементација реализује геометријско исправљање помоћу Афиних траснформација, које захтевају шест тачака, уместо стандардне трансформације перспективе која захтева осам тачака. Трансформацијом перспективе добили би се бољи резултати, али би у сам процес морао да се дода и међукорак за пројектовање координата тачке четвртог угла кода, што би цео процес учинио више математички захтевним, а самим тим и споријим. Применом Афиних трансформација, добијају се сасвим добри резултати уз мање време извршавања.

Постоји мноштво алгоритама чијом применом би се побољшао квалитет кода, али са сваким додатним кораком у процесу предобраде слике, повећава се и време извршавања. Време извршавања не сме да пређе одређену границу, јер онда би QR код изгубио своју најважнију особину, а то је брзо читање. Зато је потребно наћи баланс, да читање кода буде у што већој мери успешно, са што мањим временом извршавања. Такав приступ представља основни принцип овог рада.

7. Литература

[1] I. Pitas, Digital image processing algorithms and applications, John Wiley & Sons, 2000.

[2] Q. Chen, et al. „Fast QR code image process and detection“, *Internet of Things*, Springer, 2012, pp. 305--312.

[3] L. Karrach, E. Pivarčiova, P Božek „Identification of QR Code Perspective Distortion Based on Edge Directions and Edge Projections Analysis“, *Journal of Imaging*, т. 67, p. 67, 2020.

[4] „QR Code Generator, QR Codes 101: A Beginner's Guide,“ [На мрежи]. доступно на: https://www.qr-code-generator.com/qr-code-marketing/qr-codes-basics/. [Последњи приступ: 13. 2. 2021.].

[5] T. Vidas, et al. „QRishing: The susceptibility of smartphone users to QR code phishing attacks“, *International Conference on Financial Cryptography and Data Security*, Springer, 2013, pp. 52--69.

[6] A. Mehta, „QR code recognition from Image“, *International Journal of Advanced Research in Computer Science and Software Engineering*, т. 5, pp. 781--785, 2015.

[7] C. Weibing, G. Yang, and G. Zhang, „A simple and efficient image pre-processing for QR decoder“, *2nd International Conference on Electronic \& Mechanical Engineering and Information Technology*, Atlantis Press, 2012, pp. 234--238.

[8] M. Nazim, A. Ansari, „Quick Response Code and ITS Use in Libraries: A Recent Trend“, *Current Trends of Libraries in the ICT Era*, Research India Publication, 2014, pp. 225-236.

[9] T. Martinez, C. Tensmeyer, "Historical document image binarization: A review", *SN Computer Science*, vol. 1, pp. 1--26, 2020.

[10] R. Gupta, „Binarization Process,“ [На мрежи]. доступно на: https://madhavuniversity.edu.in/binarization-process.html#:~:text=Binarization%20is%20the%20method%20of,image%20(two%20tone%20image)%20.&text=If%20the%20gray%20value%20of,are%20converted%20into%20the%20white%20. [Последњи приступ 13. 2. 2021.].

[11] R. E. Woods. R. C. Gonzalez, Digital Image Processing, Pearson, 4th edition, Pearson, 2018.

[12] „QRcode.com, Denso Wave, the Inventor of QR Code,“ [На мрежи]. доступно на: https://www.qrcode.com/en/. [Последњи приступ 13. 2. 2021.].

[13] „FreeCodeCamp, Flood Fill Algorithm Explained,“ [На мрежи]. доступно на: https://www.freecodecamp.org/news/flood-fill-algorithm-explained/#:~:text=Flood%20fill%20is%20an%20algorithm,re%20gonna%20talk%20about%20next. [Последњи приступ 12. 2. 2021.].

[14] „HandWiki, Image moment,“ [На мрежи]. доступно на: https://handwiki.org/wiki/Image_moment. [Последњи приступ 10. 2. 2021.].

[15] M. Pawlak, S. X. Liao, „On image analysis by moments“, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* т. 18, pp. 254--266, 1996.

[16] J. C. Keyser, D. House, Foundations of Physically Based Modeling and Animation, CRC Press, 2016.

[17] „Description of an Affine Transformation,“ [На мрежи]. доступно на: http://webhelp.infovista.com/Planet/62/Subsystems/MapInfo/Content/pro_help/coordinates/descriptionofanaffinetransformation.html. [Последњи приступ 14. 2. 2021.].

[18] A. Coste, *Affine Transformation, Landmarks Registration, Non linear Warping,* October, 2012.

[19] „Kaspersky, What is QR Code? Is it safe to scan QR codes,“ [На мрежи]. доступно на: https://www.kaspersky.com/resource-center/definitions/what-is-a-qr-code-how-to-scan. [Последњи приступ 14. 2. 2021.].

8. Прилог: Кодови

```
import numpy as np
import cv2
import math
import sys
from matplotlib import pyplot as plt
import statistics

sys.setrecursionlimit(10000) # повећање стека потребног за рекурзију код поплава
алгоритма

DEVIATION = 2 # константа за одступање

# константе нивоа сиве потребне за поплава алгоритам
GRAY_LEVEL_1 = 80
GRAY_LEVEL_2 = 120
GRAY_LEVEL_3 = 160

# функција за учитавање слике
# функција приказује и враћа учитану слику у нијансама сиве

def readImage(imagePath):
    img = cv2.imread(imagePath)
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGRA2GRAY)
    cv2.imshow('Binary image', grayImg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


    return grayImg

# num -> број црних или белих пиксела у једном реду
# width -> ширина блока у једном реду
# epsilon -> одступање (+ / - epsilon)

def helpWidthFunction(num, epsilon, width):
    if(num < epsilon + width and num > width - epsilon):
        return True
    else:
        return False
```

```
# функција за тражење вредности прага
# delta -> дозвољено одступање

def findThreshold(imagePath, delta):
    grayImg = readImage(imagePath)
    newList = []
    for i in range(grayImg.shape[0]):
        for j in range(grayImg.shape[1]):
            newList.append(grayImg[i][j].astype('float'))
    initialThreshold = statistics.mean(newList)
    threshold = initialThreshold
    G1 = []
    G2 = []
    for t in newList:
        if (t > threshold):
            G1.append(t)
        else:
            G2.append(t)
    m1 = statistics.mean(G1)
    m2 = statistics.mean(G2)
    newThreshold = round((m1 + m2) / 2)
    while (abs(threshold - newThreshold) > delta):
        threshold = newThreshold
        G1 = []
        G2 = []
        for t in newList:
            if (t > threshold):
                G1.append(t)
            else:
                G2.append(t)
        m1 = statistics.mean(G1)
        m2 = statistics.mean(G2)
        newThreshold = (m1 + m2) / 2

    return newThreshold
```

```
# бинаризација слике

def binarizationOfImage(imagePath):
    grayImg = readImage(imagePath)
    threshold = findThreshold(imagePath, 1)
    for i in range(grayImg.shape[0]):
        for j in range(grayImg.shape[1]):
            if grayImg[i][j] < threshold:
                grayImg[i][j] = 0
            else:
                grayImg[i][j] = 255
    return grayImg


# услови за проналажење 1:1:3:1:1 односа

def conditionHelpFunction(elementsArr):
    w = sum(elementsArr) / 7 # ширина једног блока црног или белог у једном реду
    copyElementsW = elementsArr.copy()
    del copyElementsW[2] # копија елемената без средишњег еленета
    condition1 = all(helpWidthFunction(p, DEVIATION, w) for p in copyElementsW)
    condition2 = elementsArr[2] > 3*w - 2 and elementsArr[2] < 3*w + 2
    condition3 = elementsArr[2] > max(elementsArr[0] + elementsArr[1], elementsAr
r[3] + elementsArr[4])
    return condition1, condition2, condition3

# функција за проналажење свих хоризонталних кандидата за позициони образац

def finderPatternHorizontalHelpFunction(binaryImage):
    d = 0
    bwPixels = 0
    elementsW = []
    centroidCoord = []
    widthCentroidDict = {}
    for i in range(binaryImage.shape[0]):
        for j in range(binaryImage.shape[1] - 1):
            bwPixels += 1
            if(binaryImage[i][j] != binaryImage[i][j+1]):
                elementsW.append(bwPixels)
                bwPixels = 0
                centroidCoord.append([i, j])
                if(len(elementsW) == 5):
                    isBlack = binaryImage[centroidCoord[0][0]][centroidCoord[0][1
]] == 0 # тачно, ако је пиксел црне боје
```

```
                    if(isBlack):
                        condition1, condition2, condition3 = conditionHelpFunctio
n(elementsW)
                        if(condition1 and condition2 and condition3):
                            W = sum(elementsW) # укупна ширина
                            C = [centroidCoord[0][0], round((centroidCoord[1][1]
+ 1 + centroidCoord[2][1]) / 2)]
                            widthCentroidDict[d] = [W, C]
                            d += 1
                    elementsW.pop(0)
                    centroidCoord.pop(0)
        elementsW = []
        centroidCoord = []
    return widthCentroidDict

# функција за проналажење свих вертикалних кандидата за позиционе обрасце

def finderPatternVerticalHelpFunction(binaryImage):
    transposeBinaryImage = binaryImage.transpose()
    d = 0
    bwPixels = 0
    elementsW = []
    centroidCoord = []
    widthCentroidDict = {}
    for i in range(transposeBinaryImage.shape[0]):
        for j in range(transposeBinaryImage.shape[1] - 1):
            bwPixels += 1
            if(transposeBinaryImage[i][j] != transposeBinaryImage[i][j+1]):
                elementsW.append(bwPixels)
                bwPixels = 0
                centroidCoord.append([j, i])
                if(len(elementsW) == 5):
                    isBlack = binaryImage[centroidCoord[0][0]][centroidCoord[0][1
]] == 0
                    if(isBlack):
                        condition1, condition2, condition3 = conditionHelpFunctio
n(elementsW)
                        if(condition1 and condition2 and condition3):
                            W = sum(elementsW) # укупна ширина
                            C = [round((centroidCoord[1][0] + 1 + centroidCoord[2
][0]) / 2), centroidCoord[0][1]]
                            widthCentroidDict[d] = [W, C]
                            d += 1
                    elementsW.pop(0)
                    centroidCoord.pop(0)
```

```
        elementsW = []
        centroidCoord = []
    return widthCentroidDict

# функција за тражење средње вредности групе
def meanHelpFunction(group):
    sumX = 0
    sumY = 0
    sumW = 0
    for i in range(len(group)):
        sumX += group[i][1][0]
        sumY += group[i][1][1]
        sumW += group[i][0]
    xAvg = int(sumX / len(group))
    yAvg = int(sumY / len(group))
    wAvg = int(sumW / len(group))
    return wAvg, [xAvg, yAvg]

# функција за груписање кандидата

def groupCandidates(widthCentroidDict):
    isLocated = False
    candidatesDict = {}
    for i in range(len(widthCentroidDict)):
        isLocated = False
        if(not candidatesDict):
            candidatesDict[0] = [widthCentroidDict.get(i)]
        else:
            for j in range(len(candidatesDict)):
                for m in range(len(candidatesDict.get(j))):
                    condition1 = abs(widthCentroidDict.get(i)[1][1] - candidatesD
ict.get(j)[m][1][1]) <= 3
                    condition2 = abs(widthCentroidDict.get(i)[1][0] - candidatesD
ict.get(j)[m][1][0]) <= 3 * widthCentroidDict.get(i)[0] / 7
                    if(condition1 and condition2):
                        candidatesDict.get(j).append(widthCentroidDict.get(i))
                        isLocated = True
                        break
            if(not isLocated):
                candidatesDict[len(candidatesDict)] = [widthCentroidDict.get(i)]
    avgCandidatesDict = {}
    for i in range(len(candidatesDict)):
        avgCandidatesDict[i] = meanHelpFunction(candidatesDict.get(i))

    return avgCandidatesDict
```

```
# растојање између две тачке

def distance(p1, p2):
    return math.sqrt(math.pow((p2[0]-p1[0]),2) + math.pow((p2[1]-p1[1]),2))

# провера да ли и вертикални и хоризонтални кандидати имају однос 1:1:3:1:1

def checkHorizontalAndVertical(avgCandidatesH, avgCandidatesV):
    avgCandidatesHCopy = avgCandidatesH.copy()
    avgCandidatesVCopy = avgCandidatesV.copy()
    isNearby = False
    for i in range(len(avgCandidatesH)):
        for j in range(len(avgCandidatesV)):
            d = distance(avgCandidatesH.get(i)[1], avgCandidatesV.get(j)[1])
            if(d < 3):
                isNearby = True
                break
        if(not isNearby):
            del avgCandidatesHCopy[i]
        else:
            isNearby = False

    return avgCandidatesHCopy

# функција за бојење области

def flood_fill(posX, posY, targetColor, color):
    if(int(binaryImageCopy[posX][posY]) == color or int(binaryImageCopy[posX][pos
Y]) == -1):
        return
    if(int(binaryImageCopy[posX][posY]) != targetColor):
        return

    binaryImageCopy[posX][posY] = color

    flood_fill(posX + 1, posY, targetColor, color)  # иди на југ
    flood_fill(posX - 1, posY,targetColor,  color)  # иди на север
    flood_fill(posX, posY + 1, targetColor, color)  # иди на исток
    flood_fill(posX, posY- 1, targetColor, color)  # иди на запад

    return
```

```
# функција за тражење области позиционог обрасца и примена поплава алгоритма

def findArea(candidate):

    # црни квадрат у средини

    flood_fill(candidate[1][0], candidate[1][1], 0, GRAY_LEVEL_3)

    # бели оквир

    newPosX = -1
    newPosY = -1
    for i in range(candidate[1][0], candidate[1][0] + round(candidate[0]/3)):
        for j in range(candidate[1][1], candidate[1][1] + round(candidate[0]/3)):
            if(binaryImageCopy[i][j] == 255):
                newPosX = i
                newPosY = j
                break
        break
    flood_fill(newPosX, newPosY, 255, GRAY_LEVEL_2)

    # црни оквир

    newNewPosX = -1
    newNewPosY = -1
    for i in range(newPosX, newPosX + round(candidate[0]/3)):
        for j in range(newPosY, newPosY + round(candidate[0]/3)):
            if(binaryImageCopy[i][j] == 0):
                newNewPosX = i
                newNewPosY = j
                break
        break
    flood_fill(newNewPosX, newNewPosY, 0, GRAY_LEVEL_1)
    return binaryImageCopy

def pointsFinderPatterns(candidate):
    W = candidate[0]
    A = [candidate[1][0] - math.floor(W/2), candidate[1][1] - math.floor(W/2)]
    B = [candidate[1][0] - math.floor(W/2), candidate[1][1] + math.floor(W/2)]
    C = [candidate[1][0] + math.floor(W/2), candidate[1][1] - math.floor(W/2)]
    D = [candidate[1][0] + math.floor(W/2), candidate[1][1] + math.floor(W/2)]

    return A, B, C, D
```

```
# функција за рачунање момента M00, M01, M10 за једну област
def findMoments(targetColor, candidate):
    dxy = 2
    m00 = 0
    m10 = 0
    m01 = 0
    A, B, C, D = pointsFinderPatterns(candidate)
    k = 0
    for i in range(A[0]-dxy, C[0]+dxy):
        for j in range(A[1]-dxy, B[1]+dxy):
            if(binaryImageCopy[i][j] == targetColor):
                k += 1
                m00 += 1
        m10 += i * k
        k = 0
    # вертикални момент -> m01
    binaryImageCopyTranspose = binaryImageCopy.transpose()
    for i in range(A[1]-dxy, B[1]+dxy):
        for j in range(A[0]-dxy, C[0]+dxy):
            if(binaryImageCopyTranspose[i][j] == targetColor):
                k += 1
        m01 += i * k
        k = 0
    return m00, m10, m01

# излаз -> ажуриране координате центра и ажурирана средња ширина модула
def findNewCentroidAndModuleWidth(candidate):
    rA_M00, rA_M10, rA_M01 = findMoments(GRAY_LEVEL_3, candidate)
    rB_M00, rB_M10, rB_M01 = findMoments(GRAY_LEVEL_2, candidate)
    rC_M00, rC_M10, rC_M01 = findMoments(GRAY_LEVEL_1, candidate)
    Cx = round((rA_M10 + rB_M10 + rC_M10 ) / (rA_M00 + rB_M00 + rC_M00))
    Cy = round((rA_M01 + rB_M01 + rC_M01 ) / (rA_M00 + rB_M00 + rC_M00))
    MW = math.sqrt(rA_M00 + rB_M00 + rC_M00) / 7
    return MW, [Cx, Cy]

def findAllCentroidsAndWidth(candidates):
    updatedRealCandidates = {}
    d = 0
    for i in candidates.items():
        findArea(i[1])
        updatedRealCandidates[d] = findNewCentroidAndModuleWidth(i[1])
        d += 1
    return updatedRealCandidates
```

```
# функција проверава да ли тачке формирају правоугли једнакокраки троугао

def isTriangle(p1index, p2index, p3index, repackagedRealCandidates):
    leg1 = distance(repackagedRealCandidates[p1index][1], repackagedRealCandidate
s[p2index][1])
    leg2 = distance(repackagedRealCandidates[p1index][1], repackagedRealCandidate
s[p3index][1])
    realHypo = distance(repackagedRealCandidates[p2index][1], repackagedRealCandi
dates[p3index][1])
    hypo = math.sqrt(math.pow(leg1,2) + math.pow(leg2, 2))
    if(abs(leg1 - leg2) < 14 and abs(hypo - realHypo) < 12):
        return True
    else:
        return False

def getTop3candidates(realCandidates):
    repackagedRealCandidates = {}
    k = 0
    for i in realCandidates.items():
        repackagedRealCandidates[k] = i[1]
        k += 1
    d = 0
    top3Candidates = {}
    for i in range(len(repackagedRealCandidates) - 2):
        for j in range(i+1, len(repackagedRealCandidates)):
            for k in range(j+1, len(repackagedRealCandidates)):
                fp = isTriangle(i, j, k, repackagedRealCandidates)
                sp = isTriangle(j, i, k, repackagedRealCandidates)
                tp = isTriangle(k, j, i,repackagedRealCandidates)
                if (fp):
                    top3Candidates[d] = repackagedRealCandidates[i]
                    top3Candidates[d+1] = repackagedRealCandidates[j]
                    top3Candidates[d+2] = repackagedRealCandidates[k]
                elif(sp):
                    top3Candidates[d] = repackagedRealCandidates[j]
                    top3Candidates[d+1] = repackagedRealCandidates[i]
                    top3Candidates[d+2] = repackagedRealCandidates[k]
                elif(tp):
                    top3Candidates[d] = repackagedRealCandidates[k]
                    top3Candidates[d+1] = repackagedRealCandidates[j]
                    top3Candidates[d+2] = repackagedRealCandidates[i]

    return top3Candidates
```

```
# функција црта троугао

def drawTriangle(candidates, clr, image):
    pts = np.array([[candidates.get(0)[1][1], candidates.get(0)[1][0]],
                    [candidates.get(1)[1][1], candidates.get(1)[1][0]],
                    [candidates.get(2)[1][1], candidates.get(2)[1][0]]],
                    np.int32)
    cv2.polylines(image, [pts],  isClosed=True, color=clr, thickness=2)


# функција тражи тачке у угловима кода

def findExtendedPoints(candidates):
    hypoCenter = [round(abs((candidates.get(1)[1][1] + candidates.get(2)[1][1]) /
 2)), round(abs((candidates.get(1)[1][0] + candidates.get(2)[1][0]) / 2))]
    odnosX = (candidates.get(0)[1][1] - hypoCenter[0]) /  distance(candidates.get
(0)[1], hypoCenter)
    odnosY  = (candidates.get(0)[1][0] - hypoCenter[1]) /  distance(candidates.ge
t(0)[1], hypoCenter)
    mw = math.sqrt(18) * candidates.get(0)[0]
    h = math.sqrt(math.pow(distance(candidates.get(0)[1], candidates.get(1)[1]),
2) - math.pow(distance(candidates.get(1)[1], candidates.get(2)[1]), 2) / 4)
    p1X = round(candidates.get(0)[1][1] + (candidates.get(0)[1][1] - hypoCenter[0
]) / h * math.sqrt(18) * candidates.get(0)[0])
    p1Y = round(candidates.get(0)[1][0] + (candidates.get(0)[1][0] - hypoCenter[1
]) / h * math.sqrt(18) * candidates.get(0)[0])
    p3Y = round(candidates.get(1)[1][0] + (candidates.get(1)[1][0] - candidates.g
et(2)[1][0]) / distance(candidates.get(1)[1], candidates.get(2)[1]) * math.sqrt(1
8) * candidates.get(1)[0])
    p3X = round(candidates.get(1)[1][1] + (candidates.get(1)[1][1] - candidates.g
et(2)[1][1]) / distance(candidates.get(1)[1], candidates.get(2)[1]) * math.sqrt(1
8) * candidates.get(1)[0])
    p2Y = round(candidates.get(2)[1][0] + (candidates.get(2)[1][0] - candidates.g
et(1)[1][0]) / distance(candidates.get(1)[1], candidates.get(2)[1]) * math.sqrt(1
8) * candidates.get(2)[0])
    p2X = round(candidates.get(2)[1][1] + (candidates.get(2)[1][1] - candidates.g
et(1)[1][1]) / distance(candidates.get(1)[1], candidates.get(2)[1]) * math.sqrt(1
8) * candidates.get(2)[0])

    return (p3X, p3Y), (p1X, p1Y), (p2X, p2Y)
```

```
# функција за цртање граничних линија кода

def drawExtendedLines(candidates, color, image):
    p1, p2, p3 = findExtendedPoints(candidates)
    cv2.line(image, p1, p2, color, 2)
    cv2.line(image, p2, p3, color, 2)

# функција која прави матрицу трансформације на основу 6 тачака

def getTransMatrix(A, pA, B, pB, C, pC):
    XY = np.array([[A[1], A[0], 1],[B[1],B[0],1],[C[1],C[0],1]])
    pX = np.array([pA[1], pB[1], pC[1]])
    pY = np.array([pA[0],pB[0], pC[0]])
    firstRowParams = np.linalg.solve(XY, pX)
    secondRowParams = np.linalg.solve(XY, pY)
    transMatrix = np.array([firstRowParams, secondRowParams, [0, 0, 1]])
    return transMatrix

# Афине трансформације и инверзна Афина трансформација

def affineTransformation(binaryImg, P1, P2, P3, MW):
    height = binaryImg.shape[0]
    width = binaryImg.shape[1]
    A = distance(P1, P2) + MW
    P1p = [P2[0] + round(A), P2[1]]
    P3p = [P2[0], P2[1] + round(A)]
    transMatrix = getTransMatrix(P1, P1p, P2, P2, P3, P3p)
    newImage = np.empty([height, width])
    for x in range(height):
        for y in range(width):
            tempPixel = binaryImg[x][y]
            arr = np.array([[x], [y], [1]])
            newMatrix = transMatrix.dot(arr)
            xPrim = round(newMatrix[0][0])
            yPrim = round(newMatrix[1][0])
            if(xPrim < height and yPrim < width):
                newImage[xPrim][yPrim] = tempPixel

    inverseMatrix =  np.linalg.inv(transMatrix)

    for x in range(height):
        for y in range(width):
            arr = np.array([[x], [y], [1]])
            newMatrix = inverseMatrix.dot(arr)
            xOriginal = round(newMatrix[0][0])
```

```
            yOriginal = round(newMatrix[1][0])
            if(xOriginal > 0 and yOriginal > 0 and xOriginal < height and yOrigin
al < width):
                newImage[x][y] = binaryImg[xOriginal][yOriginal]
    P2p = [P2[0],P2[1]]

    return P1p, P2p ,P3p, newImage

# формирање бинарне матрице

def getBinaryMatrix(p1, p2, p3, MW, img):
    allArr = []
    tempArr = []
    for i in range(p2[1], p3[1] + MW, MW):
        for j in range(p2[0], p1[0] + MW, MW):
            if(img[i][j] == 255):
                tempArr.append(0)
            else:
                tempArr.append(1)
        allArr.append(tempArr)
        tempArr = []
    plt.imshow(allArr)
    plt.show()

# комплетна предобрада слике QR кода

def QRCodePreprocessing(imagePath):
    QRcodeImage = cv2.imread(imagePath)
    cv2.imshow('Original image', QRcodeImage)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    binaryImage = binarizationOfImage(imagePath)
    cv2.imshow('Binary image', binaryImage)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    widthCentroidDictH = finderPatternHorizontalHelpFunction(binaryImage)
    widthCentroidDictV = finderPatternVerticalHelpFunction(binaryImage)

    avgCandidatesDictH = groupCandidates(widthCentroidDictH)
    avgCandidatesDictV = groupCandidates(widthCentroidDictV)

    realCandidates = checkHorizontalAndVertical(avgCandidatesDictH, avgCandidates
DictV)
```

```
    top3Candidates = getTop3candidates(realCandidates)
    updatedRealCandidates = findAllCentroidsAndWidth(top3Candidates)

    drawTriangle(updatedRealCandidates, (12, 0, 138), QRcodeImage,)
    cv2.imshow('Original image', QRcodeImage)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


    drawExtendedLines(updatedRealCandidates, (255, 0, 120), QRcodeImage)
    cv2.imshow('Original image', QRcodeImage)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    MW = round(updatedRealCandidates.get(0)[0])


    P1, P2, P3 = findExtendedPoints(updatedRealCandidates)
    p1new, p2new, p3new, newImg = affineTransformation(binaryImage, P1, P2, P3, M
W)
    cv2.imshow('Image after Affine transformation', newImg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


    getBinaryMatrix(p1new, p2new, p3new, MW, newImg)
```