KokrotImg

Opis rada i tehnička dokumentacija

David Davidović
Matematička gimnazija Beograd

Šta je Kokrot?

- Biblioteka za detekciju, lociranje i čitanje QR kôdova
- QR kôdovi su specijalni dvodimenzionalni bar-kôdovi[1], čiji je izumitelj japanska kompanija Denso Wave
  - Napravljeni su specijalno kako bi programi mogli lako da ih očitaju sa fotografija
  - Prvi put korišćeni u automobilskoj industriji za automatsko obeležavanje šasija vozila
  - Danas je „skeniranje QR kôda“ vrlo uobičajeno jer su sveprisutni i našli su široku primenu van originalne namene (enkodiraju adrese web sajtova, kontakt informacije, pomažu viralne marketing kampanje, itd.)
  - Problem brze i precizne detekcije i očitavanja je vrlo interesantan, spada u probleme kompjuterskog vida i procesuiranja slika

Kako Kokrot radi

- Kokrot je podeljen na dva modula od kojih svaki vrši izolovanu funkciju
  - **KokrotImg**, čija je uloga da na početnoj slici pronađe kôd i pročita ga u binarnu matricu
    - KokrotImg je realizovan u C-u
    - *Ova prezentacija se bavi isključivo njime*
  - **KokrotProc**, čija je uloga da ovu binarnu matricu dekodira i dođe do originalnih podataka koje QR kôd nosi
    - KokrotProc je realizovan u C#-u

KokrotImg

- Podeljen je na dve komponente
  - **Macrocosm**, komponenta KokrotImg-a koja analizira sliku, pronalazi kôd i izdvaja ga u posebnu bitmapu
  - **Microcosm**, komponenta KokrotImg-a koja analizira kôd u toj bitmapi i vrši čitanje
- Uz njega je razvijen i **KokrotViz**, jednostavna alatka za vizuelizaciju rezultata i performansi KokrotImg
  - KokrotViz je realizovan u C++-u uz pomoć **GTK+ 3** toolkita i (konkretno je korišćen **gtkmm**, a za iscrtavanje je korišćen **Cairo** API)
  - KokrotViz se može proširiti da radi kao alat za vizuelizaciju šire kategorije algoritama koji pretežno rade na slikama

KokrotImg - okvirni način rada

(1) Priprema slike za obradu (ugl. grayscaling)

- Ovo je ostavljeno klijentskim aplikacijama jer nema potrebe da se gubi vreme na skupu grayscaling operaciju ako npr. već ulazna slika nije u boji. KokrotImg traži 8-bitnu jednokanalnu sliku kao ulaz.

(2) Pozivanje Macrocosm podsistema da pronađe QR kôdove na slici

(3) Odabiranje jednog QR kôda (ugl. prvi detektovan)

(4) Pozivanje Microcosm podsistema da projektuje ovaj kôd, procesuira i pročita u finalnu matricu

Bez umanjenja opštosti ćemo nadalje govoriti o QR kôdovima koji su tamni, na beloj pozadini.

KokrotImg - okvirni način rada

- Chen et. al 2012[8] opisuju algoritam za detekciju i očitavanje QR kôda sa fotografija koji je u mnogo tačaka sličan načinu rada KokrotImg-a (čak je nekoliko ideja odatle iskorišćeno prilikom dizajna načina rada KokrotImg-a)
- Mora se obratiti pažnja na to da je metod iz [8], iako na prvi pogled sličan, daleko manje sofisticiran od metoda koji KokrotImg koristi. U njemu:
  - binarizacija nije lokalno-adaptivna već koristi globalni prag
  - pronalaženje finder-a je isključivo horizontalnim i vertikalnim skenom. Autorov eksperiment sa ovim je pokazao da se u tom slučaju fatalno gubi na preciznosti ukoliko je QR kôd mali ili izrazito perspektivno transformisan
  - radi samo sa QR kôdovima verzija 1-14
  - se ne obraća pažnja na timing pattern-e

KokrotImg - okvirni način rada

  - algoritam za pronalaženje alignment pattern-a je dubiozne tačnosti---na način koji je opisan u publikaciji, može se lako prevariti i odabrati netačnu lokaciju alignment pattern-a, posebno uz nedostatak skeniranja timing pattern-a koji bi mu dao veću preciznost u pretpostavljanju njihove pozicije
- Zbog nedostatka izvornog kôda i test korpusa, nemoguće je uporediti ova dva algoritma
- U [8] se navodi da je za sliku veličine 352×258 njihovom algoritmu potrebno oko 5ms
- Za sliku slične veličine (320×427) KokrotImg-u je potrebno oko 12ms
- Sem toga se ne mogu izvući nikakvi drugi uporedni zaključci (dokument je, takođe, zbog jezičke barijere ponegde slabo razumljiv)

Macrocosm

- Macrocosm ima zadatak da pronađe na ulaznoj slici QR kôdove, na osnovu njihovih karakterističnih odlika, što preciznije
- Pošto operira na celoj slici, izuzetno je bitno da bude što brži mogući
- Potrebna je zadovoljavajuća brzina i konzumpcija memorije na slikama od nekoliko megapiksela
- Problem je memory-bound (brzina pristupa memoriji diktira brzinu izvršavanja), procesorska snaga ne igra značajnu ulogu jer je potrebno vršiti relativno jednostavne operacije nad relativno velikom količinom podataka
- Prioritet je eliminisati što više piksela koji sigurno ne pripadaju nijednom QR kôdu i procesuiranje vršiti na suženom skupu

Kako Macrocosm radi

- Slika se binarizuje (svodi na striktno crno-belu reprezentaciju)
- Na binarizovanoj slici je potrebno pronaći „finder patterne“ koji čine QR kôd
- Pronađene finder pattern-e grupisati u QR kôdove i formirati konačan izlaz Macrocosma: listu četvorouglova koji čine kôdove na slici

Binarizacija

(1) Slika se binarizuje (svodi na striktno crno-belu reprezentaciju)

- Izuzetno bitno da ovaj korak bude brz, jer on mora proći kroz celu sliku
- Nije bitno da binarizovana slika bude jasna niti bez šuma van oblasti gde su QR kôdovi, dovoljno je da oblasti sa QR kôdovima budu binarizovane tako da je kôd i dalje prepoznatljiv

Binarizacija

- Ideje koje su implementirane u početku:
  - Jednostavan globalni prag - prag se određuje na osnovu srednje vrednosti boje na slici
    - Brzo ali nezadovoljavajuće u slučaju da postoje varijacije na slici (neki delovi će ispasti neupotrebljivo presvetli a neki pretamni)
  - Prag na osnovu gray-level histograma, Otsu 1979[2]
    - Zaodovoljavajuće, ali isuviše specifičan metod
  - Lokalno-adaptivna metoda, Sauvola & Pietikainen, 2000[3]
    - Sjajni rezultati ali *izuzetno* sporo
  - Lokalno-adaptivna metoda uz dve kumulativne tabele za ubrzanje, Shafait et al. 2008[4]
    - Ista metoda kao [3] ali oko 20x brža, ipak sporo zbog neizbežnih floating-point operacija, slična ideja sa kvadriranom kumulativnom tabelom viđena i u [5]

Binarizacija

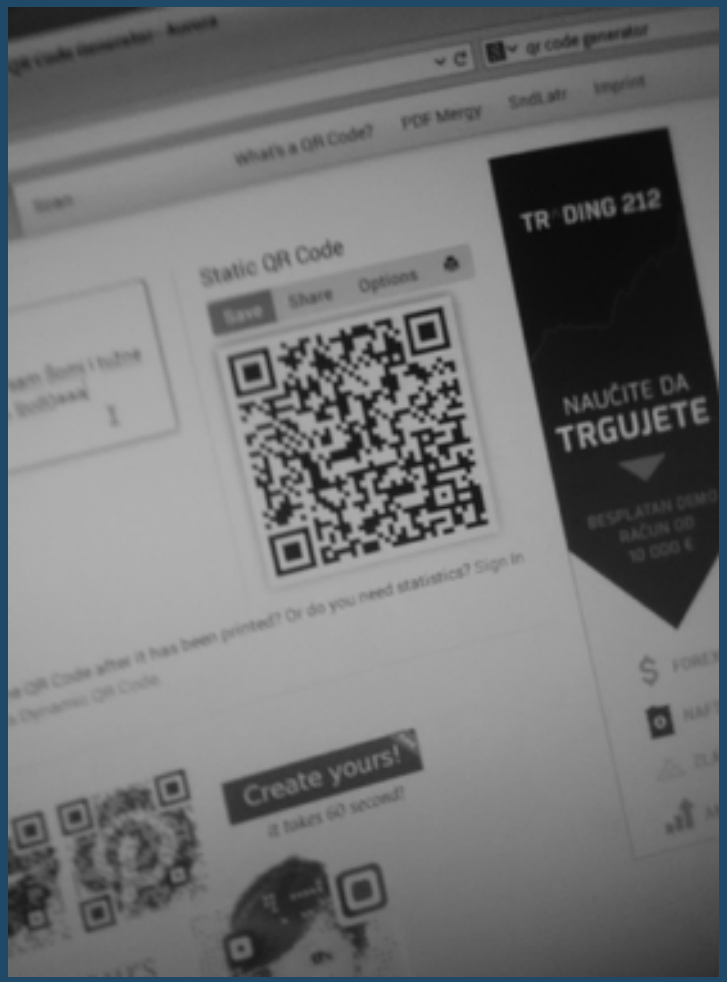
- Jednostavna lokalno adaptivna metoda sa jednom kumulativnom tabelom je najbrža i čini se da proizvodi dovoljno dobre rezultate
  - Za svaki piksel se kao prag odabira prosečna vrednost piksela u kvadratnoj okolini stranice a (gde je a funkcija veličine cele slike)
  - Koristi se kumulativna tabela (aka summed-area table / integral image) kako bi se jako brzo izračunao zbir i prosek te kvadratne okoline
  - Neupotrebljiva u opštem slučaju zbog mnogo šuma na slici
  - Međutim, u oblastima velikog kontrasta i visokih frekvencija (kao što su QR kôdovi) šuma gotovo i da nema
  - Nema floating-point operacija
  - Idealno za ovu upotrebu

Binarizacija

- Za binarizaciju slike od 8 MPX potrebno je utrošiti 110ms na referentnom sistemu, što čini oko 36% ukupnog vremena koje je provedeno u Macrocosm-u

Binarizacija

- Primer slike i njene binarizacije odabranom metodom



Pronalaženje finder pattern-a

(2) Na binarizovanoj slici je potrebno pronaći „finder patterne“ koji čine QR kôd

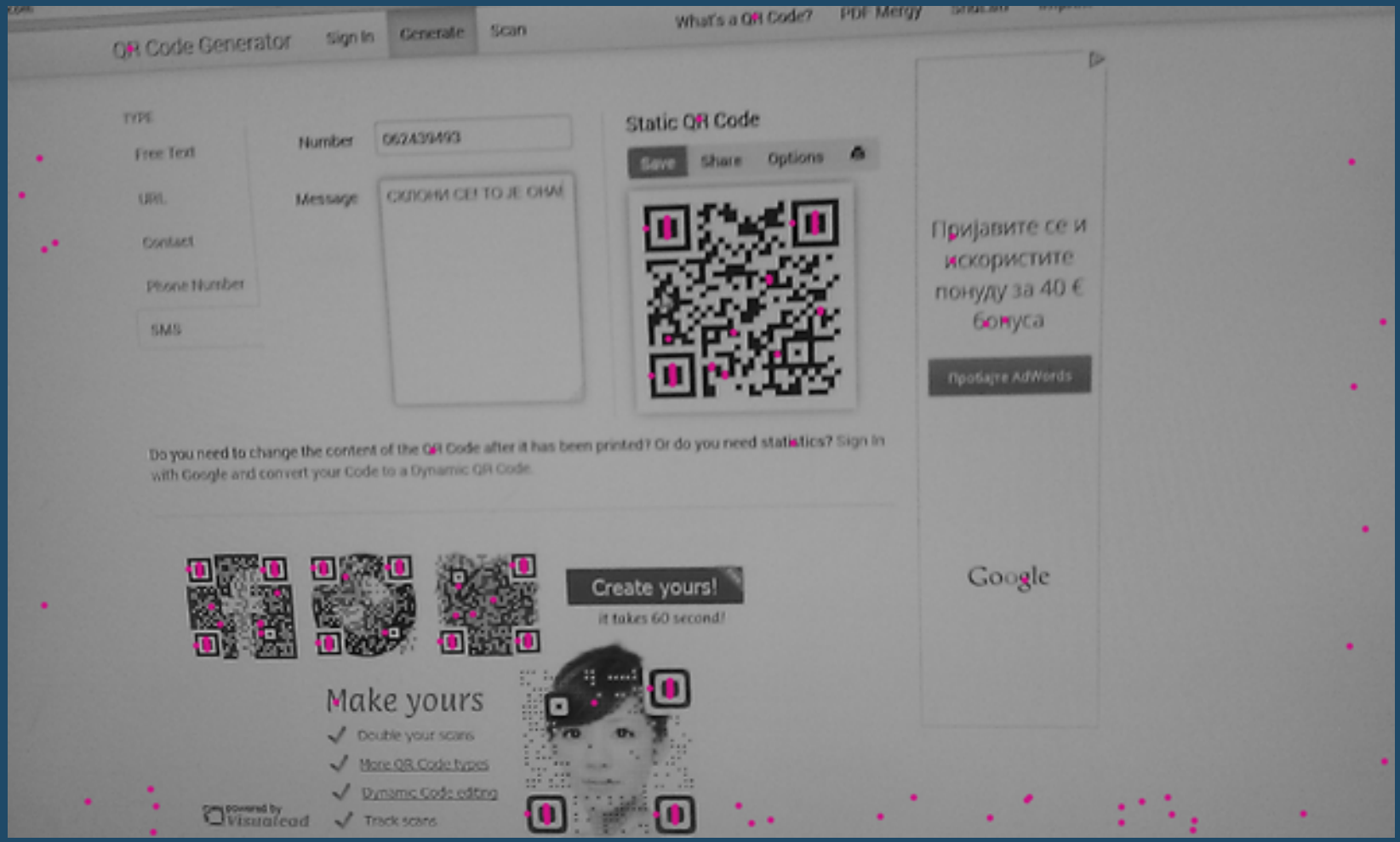
„Finder patterni“ su karakteristični uokvireni kvadrati koji se nalaze u gornjem levom, gornjem desnom i donjem levom uglu svakog QR kôda. Odnosi širina crno-belo-crno-belo-crno polja koja ih sačinjavaju su 1:1:3:1:1 zbog čega se relativno lako mogu naći i prepoznati na slici.

Pronalaženje finder pattern-a

- Prvo se cela slika skenira, liniju po liniju, i u svakoj liniji se traže odnosi naizmeničnih boja koji odgovaraju 1:1:3:1:1
- Tolerancija je jako velika, dozvoljena je relativna greška od oko 100% što stvara jako veliki broj lažnih pozitiva
- Kad god se naiđe na ovakav odnos, upamti se samo centralni piksel (onaj koji bi u finder pattern-u bio centar unutrašnjeg kvadrata
- Od tipične slike koja ima 8 MPX izdvoji se oko 300 piksela „kandidata“

Pronalaženje finder pattern-a

- Primer slike sa obeleženim pikselima kandidatima



Pronalaženje finder pattern-a

- Nakon što su kandidati enumerisani, skup nad kome se radi je efektivno smanjen sa miliona piksela na nekoliko stotina
- To znači da se mogu priuštiti kompleksnija izračunavanja na ovom nivou
- Sledeći korak je da se od ovih kandidata (koji su samo tačke) pronađu pravi finder pattern-i (četvorouglovi)

Pronalaženje finder pattern-a

- Za svaku tačku kandidat:
  - Pokrene se flood fill iz nje tako da izolujemo povezanu komponentu tj. crnu oblast kojoj ona pripada (ovo je unutrašnji 3×3 kvadrat u finder pattern-u)
    - tokom sâmog algoritma pamtimo piksele koji se nalaze na ivici oblasti
    - flood fill je dovoljno „pametan“ da prekine rad čim oblast prestane da izgleda kao legitiman kvadrat pa se izbegava širenje kroz velike povezane komponente van bilo kakvog pravog finder pattern-a
  - Pronađe se konveksni omotač ovih piksela
    - Monotone chain algoritam (optimizacija klasičnog Grejamovog skena), de Berg et al. 2000[6]
  - Izračuna se zbir vrednosti piksela unutar dobijenog poligona
    - koristi se jednostavan algoritam koji koristi dve Brezenhamove linije[7] kako bi efikasno, sken-liniju po sken-liniju, prošao kroz sve piksele unutar poligona
    - ovaj zbir mora biti najvećim delom crn (npr. 95%) jer u suprotnom znači da je originalni region vidljivo nekonveksan, što ga diskvalifikuje kao moguć finder kvadrat

Pronalaženje finder pattern-a

- Za svaki kandidat konveksni poligon:
  - Poligon se dva puta poveća tako da aproksimira prvo 5×5 beli okvir, a zatim i 7×7 celu površinu finder pattern-a
  - Svaki put se izračuna zbir piksela unutar poligona, u slučaju da ne odgovara očekivanjima (5×5 okvir nije većinski beo i 7×7 okvir nije većinski crn) kandidat se odbacuje
  - Ukoliko su svi testovi prošli, možemo sa velikom sigurnošću biti sigurni da smo naišli na pravi finder pattern
  - Prva ideja je da se prošireni poligon iskoristi kao oblik ovog finder-a u daljim izračunavanjima
    - nažalost, ispostavlja se da u praksi ovo nije moguće
    - prošireni poligon je baziran na obliku unutrašnjeg kvadrata, a zbog raznih nesavršenosti on može biti vrlo „deformisan“
    - čak i male greške ovde dovode do velikih grešaka i nepreciznosti kasnije

Pronalaženje finder pattern-a

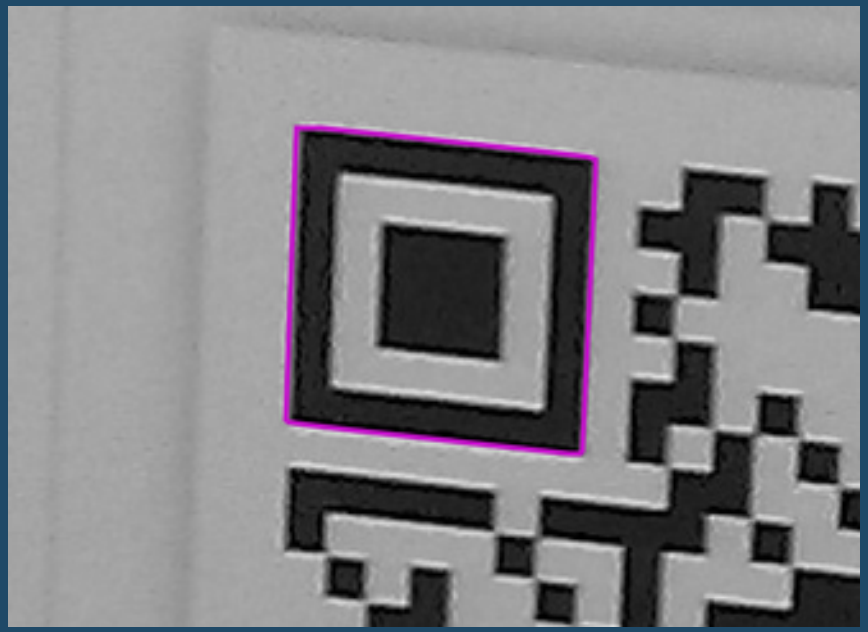
- Za svaki kandidat „proširen“ konveksni poligon:
  - Iskoristićemo proširen poligon kao smernicu
  - Želimo da pokrenemo još jedan flood fill koji će nam preciznije dati oblik celog finder pattern-a
  - Pokrenućemo „jeftin“ sken kroz prostor ovog poligona i pronaći ćemo prva 3 crna piksela iz kojih ćemo pokrenuti flood fill
  - Ovaj flood fill je vrlo konzervativan i obraća pažnju samo na jako mali pravougaonik oko originalnog poligona (i alocirano mu je malo resursa)
    - u slučaju da flood fill dostigne bilo koje ograničenje, biće korišćen originalan proširen poligon koji svejedno pruža dobru aproksimaciju pravog oblika
  - Kada se flood fill završi, uzimaju se pikseli sa ivice i od njih se pravi konveksni omotač koji je sada piksel-precizan opis oblika ovog finder pattern-a

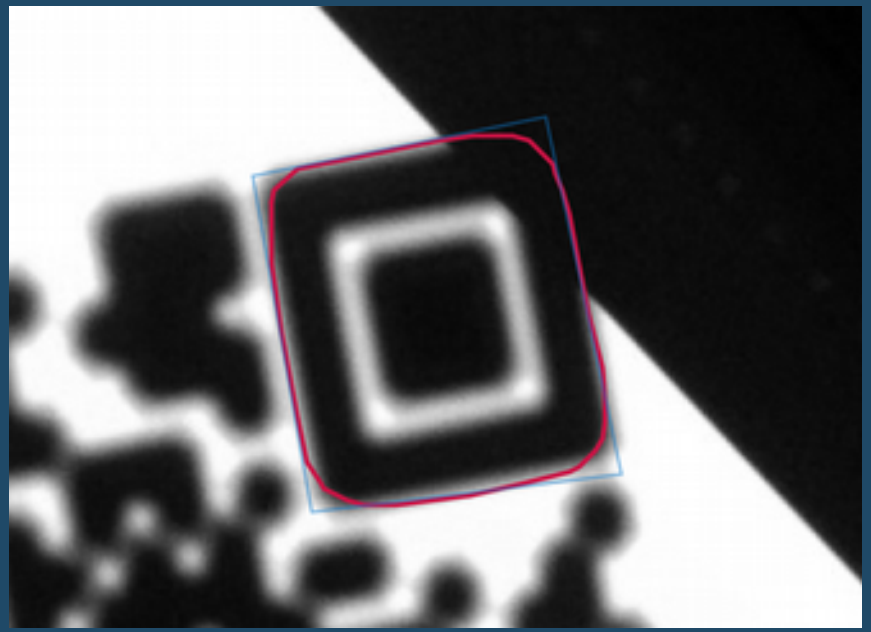
Pronalaženje finder pattern-a

- Za svaki kandidat konveksni poligon:
  - Na kraju, potrebno je da ovaj poligon redukujemo na striktno četvorougao, kako bismo mogli jeftino da radimo razne operacije (računanje površine, testiranje uglova, itd.)
  - Za ovo se koristi greedy approximation algoritam (reduce_polygon) koji iterativno oduzima tačku po tačku s poligona sve dok ne ostane onoliko koliko je potrebno
    - tačka se uvek bira tako da se površina poligona što manje promeni
    - algoritam je daleko od optimalnog i postoji mnogo patoloških slučajeva u kojima će dati vrlo neoptimalan poligon
    - na sreću, u našem slučaju, kada konveksni poligon već „liči“ na četvorougao, lokalna optimalna strategija je vrlo bliska globalnoj, pa je rezultantan četvorougao zadovoljavajuć
    - više informacija o algoritmu se nalazi u izvornom kôdu

Pronalaženje finder pattern-a

- Primer jedne situacije u kojoj je drugi, manji flood fill uspeo, i jedne u kojoj nije (jer je finder pattern povezan sa velikim crnim regionom). U drugoj situaciji pravac finder pattern-a (plavo) ne prati dovoljno dobro liniju QR kôda, dok u prvoj poligon sasvim precizno „obmotava“ konturu finder pattern-a.





Grupisanje u QR kôdove

(3) **Pronađene finder pattern-e grupisati u QR kôdove i formirati konačan izlaz Macrocosma: listu četvorouglova koji čine kôdove na slici**

- KokrotImg, najuže rečeno, nije ni namenjen da pročita više QR kôdova sa jedne slike, ali pokušavamo da se izborimo i sa tim slučajem
- Za ovaj korak ćemo koristiti algoritam koji prolazi kroz sve pronađene finder pattern-e i ocenjuje kombinacije po 3 na osnovu ugla (u idealnoj fotografiji ugao je 90°), udaljenosti (u idealnoj fotografiji centralne tačke čine jednakokrako-pravougli trougao) i odnosa površina (u idealnoj fotografiji površine su im iste)

Grupisanje u QR kôdove

- Algoritam takođe prepoznaje specijalan slučaj kada postoje samo 3 finder pattern-a na slici i uvek ih spoji zajedno, bez obzira na fitness skor dobijenog kôda
- Za svaki par findera se pronalazi treći finder koji im najbolje odgovara, a onda se za svaki finder pronalazi najbolji QR kôd koji ga sadrži i on se upisuje kao finalni (pazi se na duplikate)
- Sada treba za svaki od njih napraviti finalni četvorougao
  - sve tačke finder poligona se ubacuju u jedan „point cloud“
  - pronalazi se konveksni omotač
  - koristi se reduce_polygon da ovo svede na petougao koji predstavlja okrnjeni četvorougao QR kôda

Grupisanje u QR kôdove

Petougao koji se dobije na osnovu konveksnog omotača svih findera

- Nakon ovoga, jednostavnim ispitivanjem ovog petougla nalazimo koji je „okrnjen“ ugao i onda ga dopunjavamo kako bismo dobili ceo četvorougao
- Bitno je primetiti da je, zbog toga što se četvrta tačka u QR kôdu dobija produžavanjem linija koje diktiraju tri findera, iznenađujuće teško pronaći *tačnu* četvrtu tačku QR kôda ukoliko oblik na slici (zbog perspektive ili raznih svojstava sočiva) ne odgovara četvorouglu ili je rezolucija mala
- Ovom nepreciznošću se bavimo kasnije, u Microcosm modulu

Grupisanje u QR kôdove

- Primer uspešno detektovanog QR kôda sa prikazanim finder patternima i izračunatim četvorouglom

Kako Macrocosm radi

- U ovom trenutku posao Macrocosma je gotov
- Njegov izlaz su samo četvorouglovi koji sadrže QR kôdove
- Čak se ni binarizovana slika ne koristi već se QR kôd rebinarizuje
- Ovo je da bismo mogli da tačnije i pod kontrolisanijim uslovima izvršimo binarizaciju, jer je njen rezultat izuzetno bitan za naše čitanje finalnog kôda kasnije
- Sada nastupa Microcosm

Microcosm

- Microcosm ima zadatak da transformiše QR kôdove koje je Macrocosm prethodno našao u manju sliku, odredi njihove dimenzije i popuni finalnu matricu nulama ili jedinicama, u zavisnosti od toga da li je na finalnom kôdu određen modul crn ili beo
- Pošto operira na jako maloj slici, Microcosm je mnogo više CPU-bound, i zbog delikatne prirode ovog detektovanja je mnogo bitnije da bude tačan nego brz
- Nema potrebe da se žrtvuje preciznost zbog performansi jer se u Microcosmu provede tek nekoliko procenata od ukupnog vremena

Kako Microcosm radi

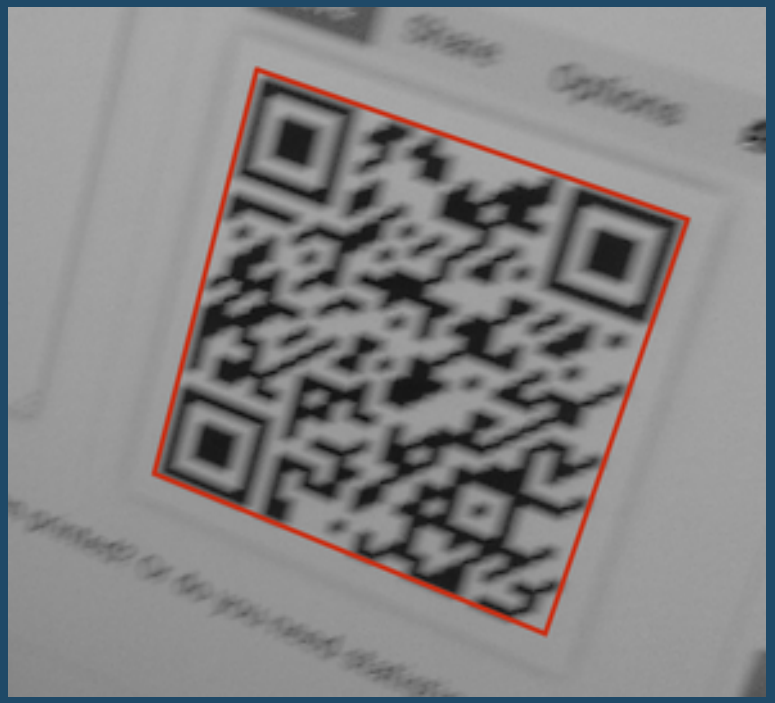
- **Četvorougao sa original slike u kome se nalazi QR kôd se transformiše u N×N bitmapu**
  - N je konfigurabilno, u default konfiguraciji je 384 što daje dovoljno dobru rezoluciju
- **Dobijena bitmapa se binarizuje**
- **QR kôd se po potrebi okreće za 90°, 180° ili 270°**
- **Skeniraju se timing patterni, uspostavljaju se okvirne x- i y-koordinate redova i kolona i određuje se verzija QR kôda**
- **Pronalaze se alignment patterni i određuju se „virtuelni“ alignment patterni koji ne postoje na slici ali se koriste kao tačke mreže**
- **Na osnovu prikupljenih podataka se gradi sempling mreža**
- **Na osnovu sempling mreže očitavaju se finalne vrednosti modula**

Transformacija kôda

(1) Četvorougao sa original slike u kome se nalazi QR kôd se transformiše u N×N bitmapu

- Ovo se radi prostom bilinearnom interpolacijom unutar ovog četvorougla i ravnomernim semplovanjem dobijenih tačaka
- Bolji rezultati bi mogli da se dobiju kada bismo u ovom postupku bili svesni perspektive i „zgusnuli“ semplove koji su dalje od kamere, videti *Moguća poboljšanja*

Transformacija kôda



Primer detektovanog kôda i njegove transformacije. Mogu se primetiti karakteristični artefakti bilinearne interpolacije, a takođe kôd izgleda nešto „zgnječeno“ jer se ne vrši korekcija perspektive (u kasnijim verzijama to će biti implementirano).

Binarizacija

(2) Dobijena bitmapa se binarizuje

- Koristimo istu metodu kao i u Macrocosm-u, s tim što ovde koristimo drugačiju veličinu lokalnog kvadrata u kome se gleda prosek piksela
- Primer originalnog kôda i njegove binarizacije:

Reorijentacija

(3) QR kôd se po potrebi okreće za 90°, 180° ili 270°

- Ovo se radi tako što se prođe kroz sva četiri ugla i pretpostavi se, na osnovu nekoliko skenova, koji od njih sadrži finder pattern a koji ne, a zatim se bitmapa okreće tako da donji desni ugao ne sadrži finder
- U ovom koraku takođe, na osnovu veličine ovih finder pattern-a, možemo da izračunamo otprilike kolika je veličina QR kôda (njegova dimenzija) odnosno „verzija“
  - Verzija QR kôda diktira koliko modula postoji po širini i dužini, broj modula se računa kao **17 + 4 * *verzija***

Timing pattern-i

(4) Skeniraju se timing patterni, uspostavljaju se okvirne x- i y- koordinate redova i kolona i određuje se verzija QR kôda

- Timing pattern-i su jedan horizontalni i jedan vertikalni niz modula koji alterniraju između crnog i belog
- Koriste se kako bi se dekoder „sinhronizovao“ sa kolonama i redovima u kôdu

Timing pattern-i

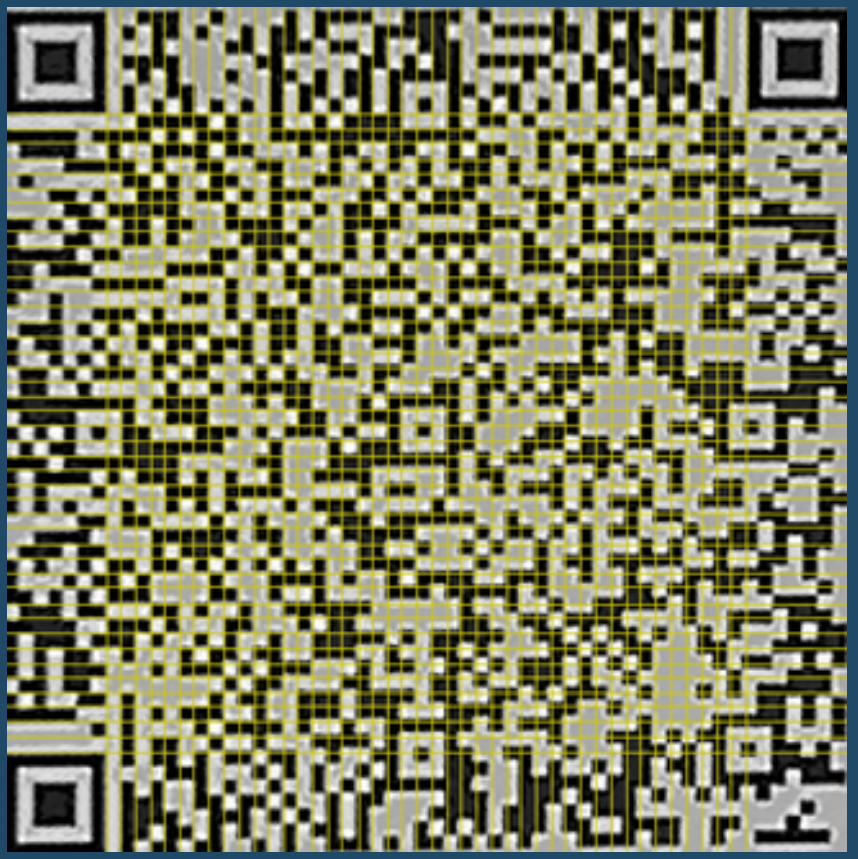
- Pošto iz reorijentacije kôda znamo otprilike gde počinju finder pattern-i, iskoristićemo ove podatke
- Takođe znamo i aproksimiranu dimenziju QR kôda, iako ovo nije sigurna merka, u praksi se pokazuje da je u svim ispitanim slučajevima tačna na ±8 modula (u više od 75% slučajeva je 100% tačna)
- Skeniraćemo više horizontalnih i vertikalnih 1-pikselnih linija iz ugla prvog finder pattern-a (u gornjem levom uglu)
- Pritom pamtimo koliko smo crnih i belih naizmeničnih oblasti piksela videli
- One čiji se broj ne podudara sa našom pretpostavkom o dimenziji odmah odbacimo

Timing pattern-i

- Od preostalih ćemo odabrati onu liniju kod koje širine modula imaju najmanju varijancu (i samim tim standardnu devijaciju)
  - Rezon iza ove odluke je da je linija kod koje su sve širine približno blizu jedna drugoj uglavnom najpreciznija
- Na osnovu ovih podataka sada znamo na kojim koordinatama počinju redovi i kolone koje timing pattern-i obuhvataju što će nam mnogo pomoći kasnije

Timing pattern-i

Dve glavne linije koje smo dobili parsiranjem timing pattern-a



Mreža koja se dobije samo na osnovu skeniranih timing pattern-a nije dovoljno precizna da bi se samo pomoću nje očitao ceo kôd: u početku je zadovoljavajuće, ali pri dnu se greška akumulira i mreža više nije sinhronizovana sa kôdom

Alignment pattern-i

(5) Pronalaze se alignment pattern-i i određuju se „virtuelni“ alignment pattern-i koji ne postoje na slici ali se koriste kasnije za dobijanje mreže

- Alignment pattern-i su slični finder-ima s tim što se nalaze unutar oblasti kôda, ne na uglovima
- Što je QR kôd veće dimenzije, to ima više alignment pattern-a, jer se u tim slučajevima male greške na početku više akumuliraju

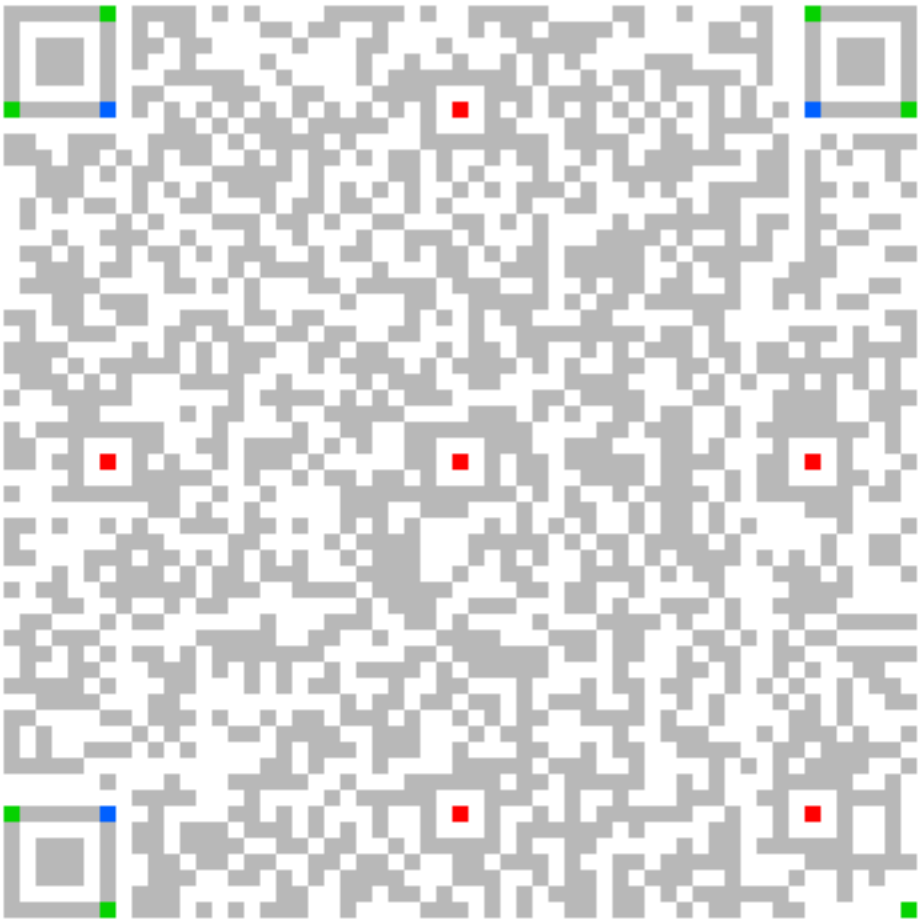
Alignment pattern-i

- Prvo ćemo na osnovu podataka o tome gde počinju i završavaju se redovi i kolone i verzije kôda pokušati da pogodimo gde se okvirno nalaze alignment pattern-i
- Baratamo isključivo sa centralnim pikselima ovih pattern-a
- Zatim ćemo skenirati oblasti oko ovih pretpostavljenih pozicija i pronađi gde raspored boja najbolje odgovara 1:1:1:1 odnosu karakterističnom za njih
- Kumulativna tabela QR kôda nam ovde pomaže da efikasno nađemo zbir piksela u nekom kvadratu

Alignment pattern-i

- Kada smo dobili vrlo precizne informacije o tome gde se svaki alignment pattern nalazi, možemo na osnovu toga da nađemo još neke karakteristične tačke koje su krucijalne za očitavanje kôda
- Njih zovemo „virtuelnim“ alignment pattern-ima jer služe za izgrađivanje mreže ali ne postoje zaista na slici
- Samo ih generišemo kako bismo postigli da algoritam za finalno očitavanje bude elegantan i bez mora specijalnih slučajeva

Alignment pattern-i



- Crveno - pravi alignment pattern-i
- Plavo - virtuelni alignment pattern-i čije koordinate su prirodna posledica načina na koji originalna specifikacija predviđa njihove pozicije
- Zeleno - virtuelni alignment pattern-i veštački ubačeni da bi mogla da se napravi sempling mreža u oblastima koje ostali alignment pattern-i ne pokrivaju

Građenje sempling mreže

(6) Na osnovu prikupljenih podataka se gradi sempling mreža

- Dovoljno je samo da iskoristimo podatke o alignment pattern-ima koje smo prethodno skupili i između njih napravimo četvorouglove koje ćemo onda ravnomerno semplovati, modul po modul

Građenje sempling mreže

Četvorouglovi između alignment pattern-a na osnovu kojih se vrši semplovanje

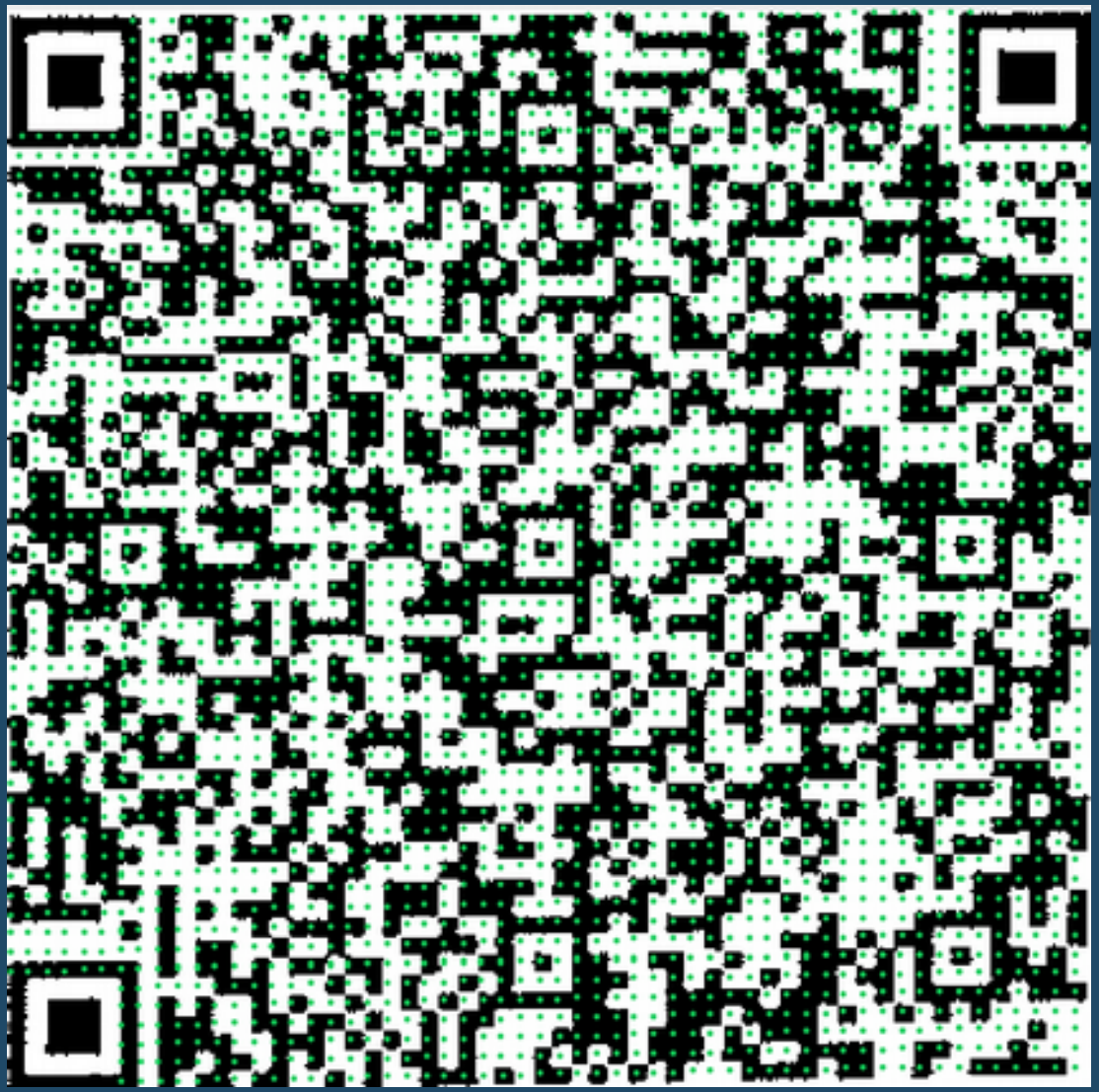
ovo je u međuvremenu unapređeno, v. commit `ae7f028`

Očitavanje finalnog kôda

(7) Na osnovu sempling mreže se očitavaju finalne vrednosti modula

- Istom metodom bilinearne interpolacije kao i prilikom projektovanja kôda prolazimo kroz četvorouglove i čitamo modul po modul iz binarizovane slike

Očitavanje finalnog kôda



Finalna mreža modula (prikazani centri semplova)

Očitavanje finalnog kôda

Očitana matrica (renderovan u istim dimenzijama kao original slika zbog poređenja

Kraj KokrotImg procesa

- U ovom trenutku sve je spremno da se finalna matrica sa nulama i jedinicama vrati pozivajućoj aplikaciji za dalju obradu :)
- Sada na red nastupa KokrotProc čija je uloga da ovu matricu dekodira i dođe do originalnih podataka koje QR kôd nosi

Dalja poboljšanja

- KokrotImg je u ranoj alpha fazi - iako je u mogućnosti da prepozna i uspešno dekodira dosta slika (čak i nekih sa kojima se najpoznatija open-source alternativa---ZXing---ne može izboriti) i dalje postoji mnogo mesta za usavršavanje
- Dosta se vremena provede šireći se flood fill-om kroz regione koji predstavljaju samo šum binarizacije, biblioteka bi bila mnogo brža kada bi mogla još brže da detektuje ovaj slučaj
- Već spomenuto ranije, kôd za projektovanje QR kôda nije svestan perspektive pa se projektovanje događa vrlo neoptimalno (ovo je jedan od prioriteta)

Dalja poboljšanja

- Kada bi projektovanje bilo preciznije u tom smislu, možda bi mogli da se potpuno eliminišu neki koraci u Microcosm-u koji su tu da se izbore sa nepreciznošću ulazne fotografije
- Bilinearna interpolacija bi se mogla zameniti bikubnom, mada je diskutabilno koliko bi ovo donelo na tačnosti
- **Zbog nedostatka dovoljno virtuelnih alignment pattern-a, očitavanje jako velikih kôdova je u većini slučajeva vrlo suboptimalno.** Ovo je prioritet za buduće verzije (i radi se na tome), ali u trenutku pisanja to nije rešeno

  (u međuvremenu rešeno u `ae7f028`)

Dalja poboljšanja

- Veličina bitmape u Microcosmu je fiksirana bez obzira na verziju kôda --- bilo bi „pametnije“ kada bi se ova veličina adaptirala u odnosu na verziju pročitanog kôda
- Razne metode binarizacije nisu testirane u kontekstu ostalog kôda, već samo izolovano---možda bi sporiji metod mogao da, eliminisanjem šuma, zapravo postigne da ceo Macrocosm radi brže
- Debug output kroz debug sink interfejs u biblioteci možda nije od pomoći koliko bi trebalo da bude
- Instrumentacija vremena u Microcosmu trenutno nije sasvim dovršena

  (u međuvremenu rešeno u `04f6903`, takođe v. `97f4898`)

KokrotViz

- Mali GTK+ 3 (gtkmm) C++ program koji učitava sliku, dinamički učitava KokrotImg biblioteku i koristi njen API da detektuje kôd na slici
- Pritom koristi dijagnostičke mogućnosti koje pruža biblioteka kako bi dobio uvid u vremensku i memorijsku konzumpciju i međustanja biblioteke
- Bio veoma koristan tokom razvoja biblioteke jer je zbog dinamičkog učitavanja mogao skoro istog trenutka da prikaže kako su promene u kôdu uticale na izlaz i omogućavao vizuelizaciju celog procesa od originalne slike do finalne matrice

KokrotViz

- Nije bio prioritet pa je pun memory leakova i kôd je na određenim mestima izrazito ružan
- Zato što koristi Cairo API a ne OpenGL (i pritom na neoptimizovan način), vizuelizacija (posebno kada su uključeni layer-i sa mnogo elemenata) laguje prilikom panning-a
- Treba implementirati promenu boje layer-a i alpha blending za bitmape
- Uz malo dorade bi mogao biti koristan za debugging image processing softvera opšte namene
- Framework za image processing je jedan od potencijalnih smerova u kome će se kretati ovaj projekat, tu će KokrotViz izuzetno pomoći

KokrotViz


kokrotviz
49
Save
printed? Or do you need statistics? Sign In
KokrotViz
Image file:
3.jpg
Scan image
Macrocosm
Microcosm
Messages
Metrics
Layers
ID Name Visible Opacity Line width
1202 Original image 255.000000 5.000000
1201 Binarized image 255.000000 5.000000
1101 Finder candidates 255.000000 5.000000
1109 QR code quads 255.000000 5.000000
1107 Finder pattern polygons 255.000000 5.000000
1106 Finder pattern quads 255.000000 5.000000

KokrotViz

KokrotViz

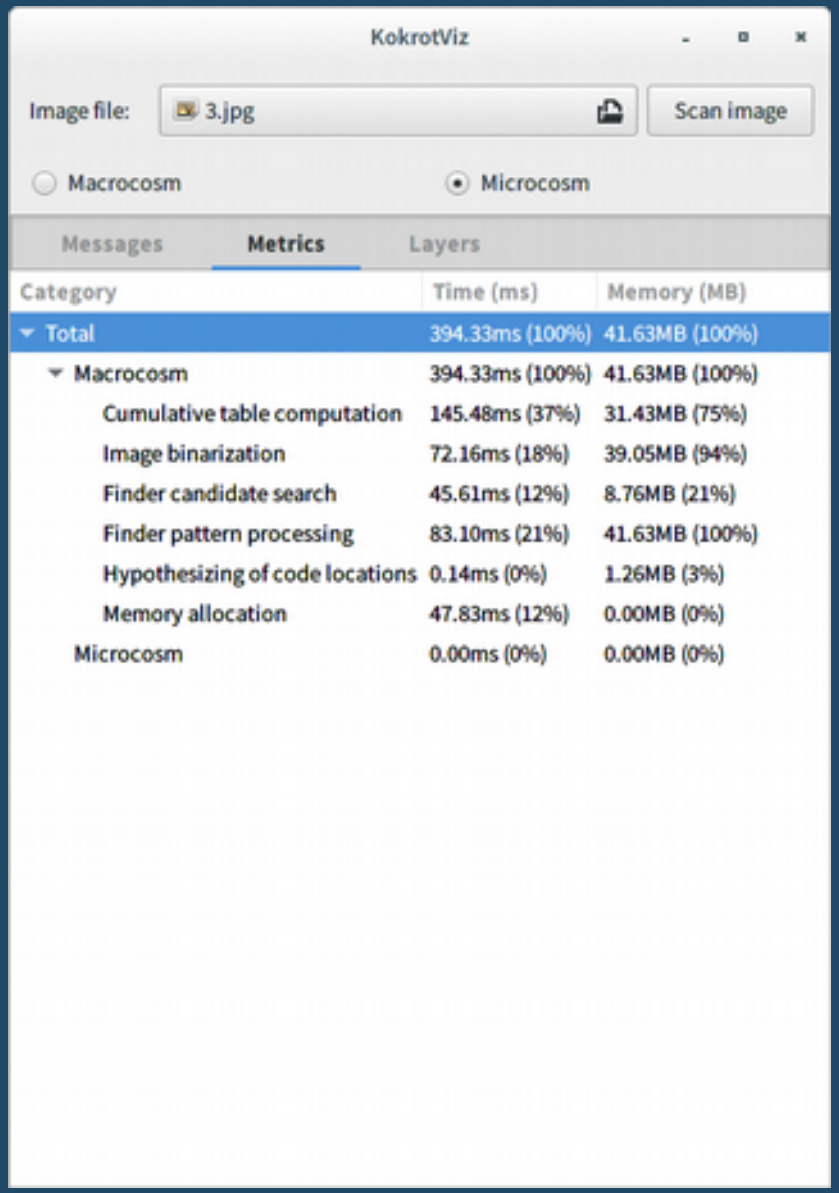
Image file: 3.jpg Scan image

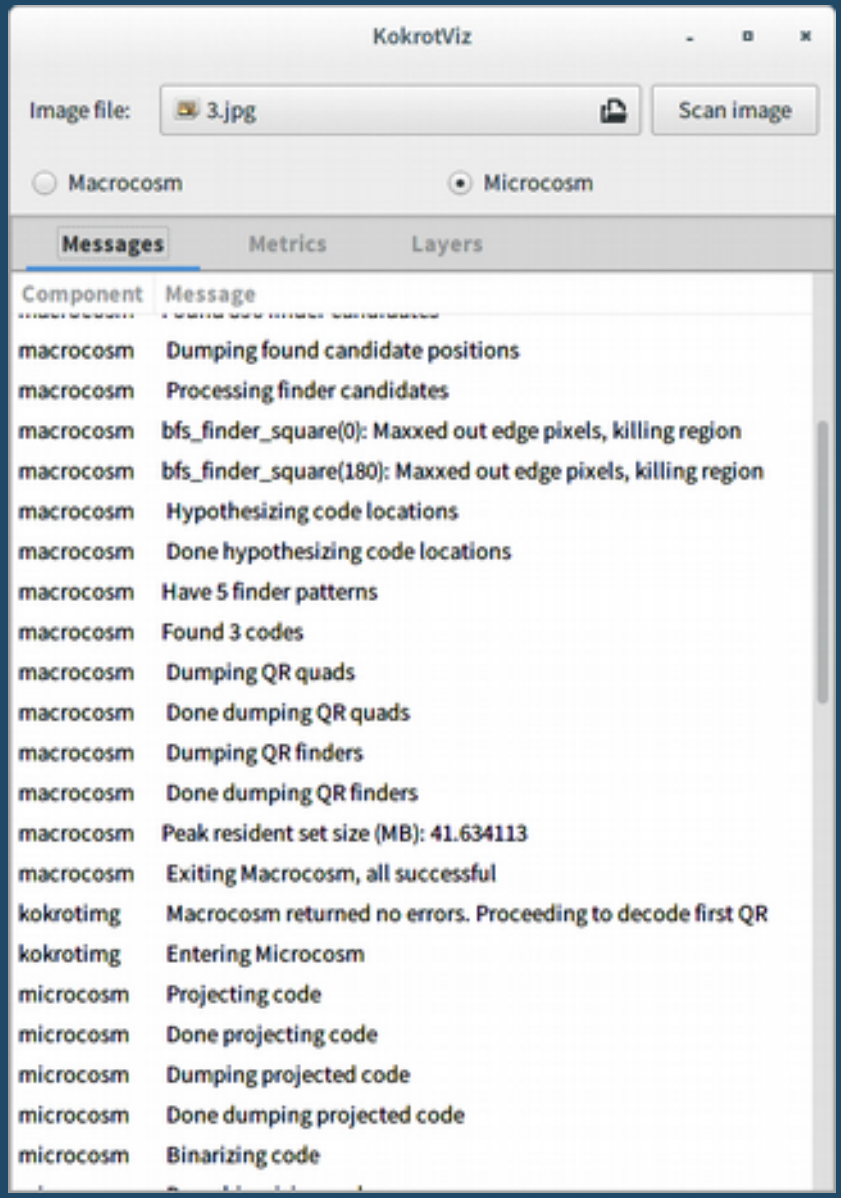
Macrocosm Microcosm

Messages Metrics Layers

Component	Message
microcosm	bad count y 42
microcosm	bad count x 56
microcosm	bad count y 42
microcosm	Done parsing timing patterns
microcosm	Conjecturing alignment pattern locations
microcosm	Done conjecturing alignment pattern locations
microcosm	Refining guesses for alignment pattern locations
microcosm	Done refining guesses for alignment pattern locations
microcosm	Placing virtual alignment patterns
microcosm	Done placing virtual alignment patterns
microcosm	Reading final QR
microcosm	Reading QR quad (6, 6) (30 6) (6 30) (30 30)
microcosm	Reading QR quad (6, 0) (30 0) (6 6) (30 6)
microcosm	Reading QR quad (0, 6) (6 6) (0 30) (6 30)
microcosm	Reading QR quad (6, 30) (30 30) (6 36) (30 36)
microcosm	Reading QR quad (30, 6) (36 6) (30 36) (36 36)
microcosm	Done reading final QR
microcosm	Writing to output matrix
microcosm	Done writing to output matrix
microcosm	Dumping final QR image
microcosm	Done dumping final QR image
microcosm	Exiting Microcosm, all successful

KokrotViz


KokrotViz
Image file:
3.jpg
Scan image
Macrocosm
Microcosm
Messages
Metrics
Layers
Category
Time (ms)
Memory (MB)
Total 394.33ms (100%) 41.63MB (100%)
Macrocosm 394.33ms (100%) 41.63MB (100%)
Cumulative table computation 145.48ms (37%) 31.43MB (75%)
Image binarization 72.16ms (18%) 39.05MB (94%)
Finder candidate search 45.61ms (12%) 8.76MB (21%)
Finder pattern processing 83.10ms (21%) 41.63MB (100%)
Hypothesizing of code locations 0.14ms (0%) 1.26MB (3%)
Memory allocation 47.83ms (12%) 0.00MB (0%)
Microcosm 0.00ms (0%) 0.00MB (0%)


KokrotViz
Image file:
3.jpg
Scan image
Macrocosm
Microcosm
Messages
Metrics
Layers
Component
Message
macrocosm Dumping found candidate positions
macrocosm Processing finder candidates
macrocosm bfs_finder_square(0): Maxxed out edge pixels, killing region
macrocosm bfs_finder_square(180): Maxxed out edge pixels, killing region
macrocosm Hypothesizing code locations
macrocosm Done hypothesizing code locations
macrocosm Have 5 finder patterns
macrocosm Found 3 codes
macrocosm Dumping QR quads
macrocosm Done dumping QR quads
macrocosm Dumping QR finders
macrocosm Done dumping QR finders
macrocosm Peak resident set size (MB): 41.634113
macrocosm Exiting Macrocosm, all successful
kokrotimg Macrocosm returned no errors. Proceeding to decode first QR
kokrotimg Entering Microcosm
microcosm Projecting code
microcosm Done projecting code
microcosm Dumping projected code
microcosm Done dumping projected code
microcosm Binarizing code

Reference

- [1]: "Information technology — Automatic identification and data capture techniques — Bar code symbology — QR Code", ISO/IEC 18004, 2000
- [2]: "A Threshold Selection Method from Gray-Level Histograms", N. Otsu, IEEE Transactions on Systems, Man and Cybernetics vol. SMC-9 #1, pgs. 62-66, 1979
- [3]: "Adaptive document image binarization", J. Sauvola / M. Pietikäinen, Pattern Recognition vol. 33, pgs. 225-236, 2000

Reference

- [4]: "Efficient implementation of local adaptive thresholding techniques using integral images", F. Shafait / D. Keysers / T. M. Breuel, 2008
- [5]: "Summed-Area Variance Shadow Maps", GPU Gems 3 chapter 8, Andrew Lauritzen, 2008
- [6]: "Computational Geometry: Algorithms and Applications", Third Edition, M. de Berg / O. Cheong / M. van Kreveld / M. Overmars, 2008
- [7]: "Algorithm for computer control of a digital plotter", IBM Systems Journal vol. 4 #1, pgs. 25-30, J. E. Bresenham, 1965

Reference

- [8]: "A Simple and Efficient Image Pre-processing for QR Decoder", W. Chen / G. Yang / G. Zhang, Advances in Intelligent Systems Research, 2012

Izvorni kôd

- Celokupan izvorni kôd KokrotImg-a i KokrotViz-a dostupan je pod MIT licencom na **https://github.com/geomaster/kokrotimg**
- Na njima se aktivno radi
- Trenutno je primarno podržana samo Linux platforma ali je moguće kompajlovati i koristiti biblioteku na Windows-u pomoću Cygwin-a, uskoro će nekompatibilnosti biti sređene pa će moći da se napravi nativni build