# Preštevanje in osamitev objektov slike

Tadej Borovšak, 27142027

1. september 2015

**Povzetek**

V tem projektu smo skušali sestaviti postopek, s katerim bi lahko preštevali objekte na slikah s pomočjo programskega paketa Octave. V ta namen smo zasnovali dokaj enostaven algoritem, ki z različnimi modifikacijami vhodne slike dokaj uspešno prešteje objekte. Za osamitev objektov pa smo uporabili algoritem popljavljanja, ki sliko loči na posamezne komponente glede na strukturo.

## Kazalo



## 1 Algoritem poplavljanja

V tem razdelku bomo na kratko opisali algoritem ter predstavili našo delno optimizirano implementacijo za Octave.

### 1.1 Teoretične osnove

Osnova ideja algoritma poplavljanja [1] je, da 2-dimenzionalno sivinsko sliko interpretiramo kot 3-dimenzionalno površino, kjer ima je višina točke s koordinatama $(x, y)$ enaka vrednosti, ki jo vsebuje izbrana pika slike. Na dobljenem reliefu nato izvedemo simulacijo dvigovanja gladine vode iz lokalnih minimumov. Med izvajanjem simulacije se oblikujejo zadrževalni bazeni, ki se ob doseženi določeni višini gladine začnejo združevati. Točke, kjer se dva zadrževalna bazena prvič stakneta, nam ob končani simulaciji podata subdivizijo slike v več območij.

[1] ang. *watershed transformation*

Obstajata dve vrsti opisanega algoritma. Osnovna različica kot točke, kjer voda prihaja v bazene, uporabi vse lokalne minimume, naprednejša različica pa je vodeni algoritem, kjer ob začetku simulacije sami podamo izvore vode in s tem tudi določimo število območij, ki jih pričakujemo ob končanem postopku.

Več informacij in povezav je na voljo nas wikipediji: `https://en.wikipedia.org/wiki/Watershed_(image_processing)`, mi pa si bomo sedaj ogledali implementacijo algoritma.

## 1.2 Implementacija

Programsko okolje Octave algoritma poplavljanja na žalost še ne vsebuje, zato smo se odločili, da bomo pripravili delno optimizirano različico algoritma ter jo izpostavili kot navadno funkcijo.

Octave nam omogoča, da si računsko zahtevne operacije napišemo v programskem jeziku C++. Na takšen način lahko pripravimo binarni modul, ki ga Octave ob zagonu lahko naloži v pomnilnik in nato uporablja kot povsem navadno funkcijo, ki bi jo napisali v višjenivojskem jeziku.

Pri tem seveda precej pridobimo na hitrosti, ker imamo večji nadzor nad samimi podatkovnimi strukturami, ki jih uporabljamo za algoritem, obenem pa ni potrebe po sprotnem prevajanju kode, ker smo operacijo prevedli v binarno obliko.

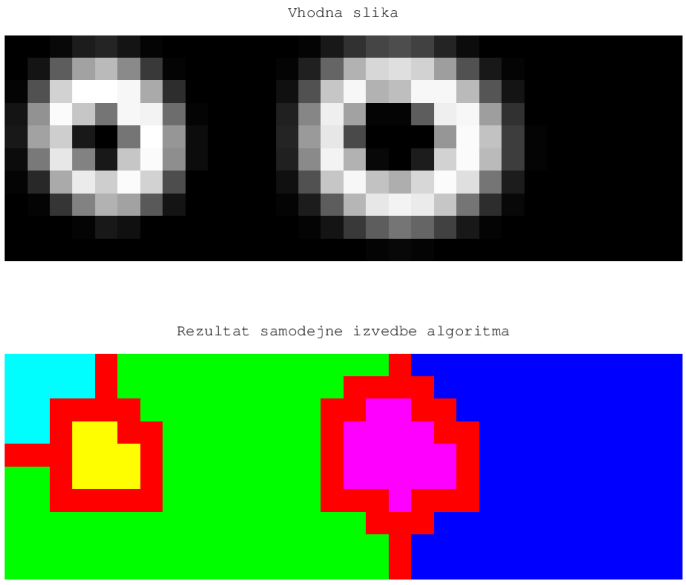
Podrobnosti implementacije se v tem poročilu ne bi spuščali, ker je vsa izvorna koda prosto dostopna na `github`.

## 1.3 Primeri

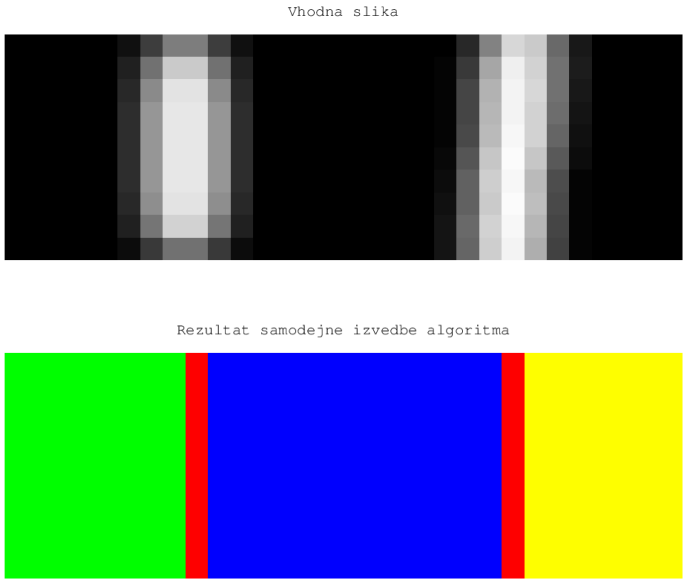
Za konec si oglemo še nekaj primerov izvedbe algoritma.

Na sliki 1 je prikazana vhodna slika za algoritem, pod njo pa dobljeni rezultat. Rdeča barva prikazuje črte, ki med seboj ločijo posamezne komponente subdivizije slike, vsaka izmed komponent pa ima svojo barvo.

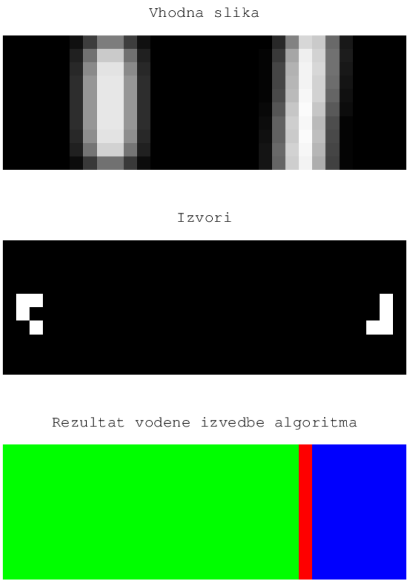
Na naslednjih treh primerihh pa lahko primerjamo rezultate, ki jih dobimo s samodejno in vodeno različico algoritma. Slika 2 je rezultat samodejnega algoritma, slika 3 prikazuje rezultat vodene segmentacije v dve komponenti, slika 4 pa ponovno prikazuje vodemo segmentaijo, a z drugačnima semenoma.



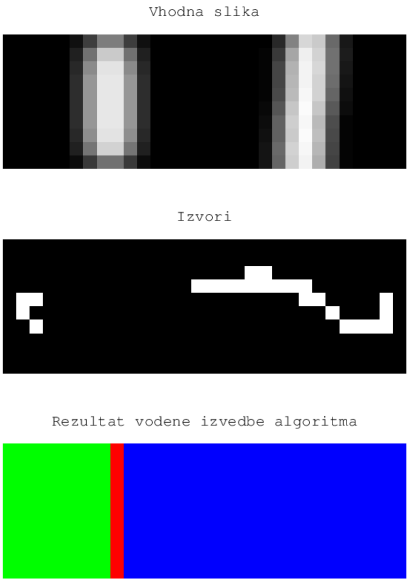

Slika 1: Prikaz algoritma na dveh obročih.




Slika 2: Samodejni algoritem na treh prekatih.




Slika 3: Vodeni algoritem na treh prekatih.




Slika 4: Vodeni algoritem na treh prekatih.

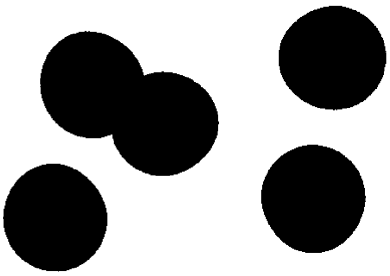
## 2 Praktični primer

Kot obljubljeno si bomo sedaj na kratko ogledali enega izmed mogočih načinov uporabe algoritma. V našem primeru bomo skušali poiskati ter osamiti posamezne kovance s slike 5.

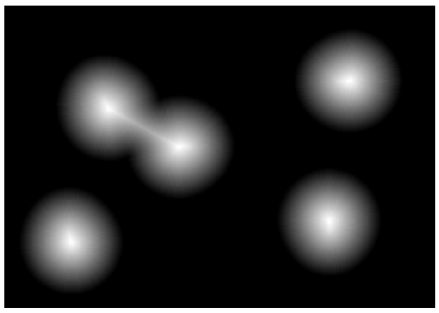
Skozi praktični primer se bomo sprehodili le ob slikah vmesnih rezultatov, izvorna koda programa, ki je pripravil slike, pa je na voljo na `github`.

Slika 5: Vzorčna slika, s katere želimo izolirati kovance.

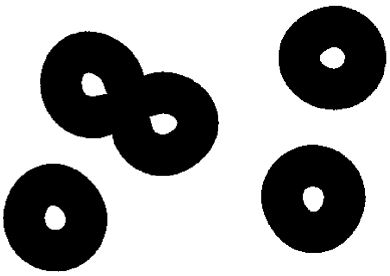
Na prvem koraku sliki najprej odstranimo ozadje, rezultat odstranitve je prikazan na sliki 6. Nato izračunamo oddaljenost vsakega črnega piksla do najbližjega belega in dobimo sliko 7 (svetlejši deli slike so bolj oddaljeni od črnih ikslov v izvoru). Po filtriranju glede na razdaljo na sliki 8 dobimo z belo barvo označene izhodiščne vire za algoritem. Končni rezultat je prikazan na sliki 9.



Slika 6: Odstranjeno ozadje.



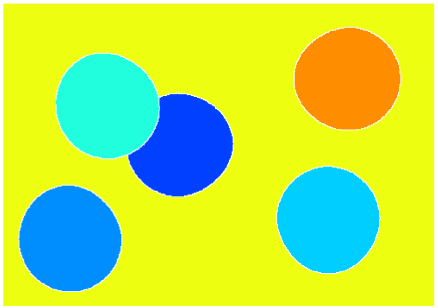
Slika 7: Izmerjene razdalje do ozadja.



Slika 8: Začetni izvori.

Za ločitev kovancev na sliki v več slik pa sedaj izvedemo maskiranje in obrezovanje vhodne slike s pomočjo maske, ki jo ustvarimo za vsak segment posebej.

Zakaj smo v tem primeru sploh potrebovali algoritem poplavljanja? Težava sta kovanca, ki se prekrivata in pri uporabi večine filtrov tvorita celoto, ki se je ne da enostavno ločiti. Segmentacija s pomočjo vodenega poplavljanja pa nam je omogočila ločiti tudi ta dva kovanca.



Slika 9: Segmentirana slika.

## 3 Zaključek

V tem delu smo se precej nežno dotaknili procesiranja slik ter si privoščili kratek izlet v malo manj poznane predele programskega paketa Octave. Pripravili smo ne povsem naivno implementacijo algoritma ter jo uspešno uporabili na praktičnem primeru.