Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Praćenje pokretnih objekata u video snimkama

Adrian Žgaljić

Zagreb, 2017.

Ovaj rad izrađen je u Zavodu za elektroničke sustave i obradu informacija pod vodstvom doc.dr.sc. Marka Subašića

Sadržaj

1. Uvod

Računalni vid područje je umjetne inteligencije koje uključuje metode za stjecanje, obradu, analizu i razumijevanje slike i općenito, više dimenzionalnih podataka iz realnog svijeta u cilju dobivanja numeričkih ili simboličkih informacija.
Od raznih područja u kojima se primjenjuje računalni vid, jedna od njih je i gospodarstvo, točnije marikultura. Marikultura je kontrolirani uzgoj riba, rakova, mekušaca i vodenoga bilja u morskoj ili bočatoj vodi. Odvija se u bazenima, plutajućim kavezima ili drugim uzgojnim instalacijama koje se smještaju u zaštićene dijelove priobalnoga mora. U marikulturu spada i kavezni uzgoj tuna koji je iznimno razvijen u Hrvatskoj te se godišnje proizvede gotovo 4000t tuna. Tune koje se uzgajaju smještaju se u plutajuće kaveze promjera od približno 50 m i dubine od gotovo 25 m, gdje se svakodnevno hrane. Proizvođačima tuna važno je znati s kojim brojem tuna raspolažu, a kako se radi o velikom broju tuna koje su stalno u pokretu dolazi do problema jer ih je teško izbrojati. Jedno od mogućih rješenja ovog problema upravo je primjena računalnog vida, točnije detekcija i praćenje pokretnih objekata u video snimci.

2. Cilj rada

Trenutačno se u uzgajalištima kod premještanja tuna iz jednog kaveza u drugi postupak snima podvodnom kamerom te se tune broje naknadno proučavanjem video snimaka. Taj postupak brojanja tuna je spor i zamoran te bi bilo jako dobro kada bi se cijeli postupak mogao automatizirati primjenom računalnog vida.
Cilj ovog rada je izrada programskog rješenja koje bi bilo u mogućnosti izbrojati tune u video snimci. Kako nisam bio u mogućnosti doći do snimaka tuna, za ovaj rad je korištena simulacija akvarija s tunama napravljena pomoću biblioteke OpenGL. Pomoću simulacije je moguće namještati razne postavke osvjetljenja, pozadine, kuta snimanja i pravca kretanja riba pa smatram da je video simulacija dobra zamjena za stvarnu snimku.
Cilj je ostvariti što veću točnost programskog rješenja u brojanju tuna kako bi se program mogao iskoristiti i u stvarnom svijetu.

3. Korištene tehnologije

Problem brojanja tuna mogao bi se podijeliti u dvije glavne cjeline: detekciju i praćenje tuna koje spadaju u domenu računalnog vida. Kako bi se izgradilo programsko rješenje, logičan izbor je korištenje biblioteke otvorenog koda za rad s računalnim vidom. Danas najpoznatije takve biblioteke su Matlabov Computer Vision System Toolbox te biblioteka OpenCV. Iako sam imao iskustva s obje biblioteke, mogućnost korištenja Java omotača za biblioteku OpenCV prevagnula je u odabiru tehnologije zbog dobrog poznavanje programskog jezika Java.

3.1. Programski jezik Java

Programski jezik Java nastao je iz programskog jezika Oak. Počev od 1995. godine kada je u internet preglednik Netscape Navigator ugrađena podrška za Javu pa sve do danas, jezik se razvija i širi te je danas praktički sveprisutan. [3]
Velika prednost u odnosu na većinu dotadašnjih programskih jezika je to što se programi pisani u Javi mogu izvoditi bez preinaka na svim operativnim sustavima za koje postoji JVM (Java Virtual Machine), dok je klasične programe pisane primjerice u C-u potrebno prilagođavati platformi(Operacijskom sustavu) na kojem se izvode. Programi napisani u Javi pomoću Java prevodioca javac prevode se u bajtkod (engl. bytecode) koji se zatim izvodi na Javinom virtualnom stroju. Virtualni stroj rješava i problem automatskog oslobađanja resursa uz pomoć skupljanja smeća (engl. garbage collector). Dio koda za oslobađanje memorije premješten je upravo u virtualni stroj, pa programer ne treba voditi brigu o upravljanju memorijom, niti pisati naredbe za oslobađanje iste. Upravo su uz upravljanje memorijom vezane najčešće pogreške u jezicima kao što su C i C++, te je po tom pitanju ovakvo rješenje u Javi uvelike olakšalo razvoj programske potpore. [1]
Java je objektno-orijentirani programski jezik. Središnji pojmovi s kojima ovaj jezik barata su klase (engl. class, koristi se još i termin razred) te primjerak klase ili objekt (engl. object). Osnovni tipovi u Javi, poput int, boolean i drugih, također su predstavljeni u klasama i imaju svoje klasne omotače (engl. wrapper). [4]
Java je jedan od najpopularnijih programskih jezika, te se procjenjuje se da ima više od 9 milijuna korisnika diljem svijeta. Koristi se uglavnom tamo gdje je brzina razvoja programskog sustava važnija od brzine rada programa, te u radu s mrežnim komunikacijama budući da pruža bolji stupanj sigurnosti i pouzdanosti od većine drugih jezika, te posjeduje bogati skup klasa za rad s njima. [2]

3.2. Biblioteka OpenCV

OpenCV (Open Source Computer Vision Library) je programska biblioteka otvorenog koda razvijena za rad na području računalnog vida. Razvijena je u istraživačkom centru Intel Rusija u Nižnjem Novgorodu. Kada je 1999. godine projekt službeno objavljen, glavna inicijativa bila mu je unaprijediti procesorski zahtjevne poslove kao dio serije projekata na područja računalne grafike. Prvo alfa izdanje biblioteke bilo je 2000. godine iza koje slijedi pet beta izdanja od 2001. do 2005. godine. Prva 1.0 verzija OpenCVja izdana je 2006. godine, a 2008. godine dobio je korporativnu podršku kompanije Willow Garage. Drugo značajno izdanje OpenCV 2 objavljeno je 2009. godine i uključuje nove funkcije te bolje
implementacije postojećih, u smislu performanci (ponajviše na višejezgrenim sustavima). Službena izdanja danas se objavljuju svakih 6 mjeseci, a u kolovozu 2012. godine podršku nad OpenCVjem je preuzela neprofitna organizacija OpenCV.org. [15]
Biblioteka je napisana u jezicima C i C++ a podržana je na operacijskim sustavima Windows, Android, Meamo, FreeBSD, OpenBSD, BlackBerry 10, iOS, Linux i OS X. Osim korištenja jezika C za razvoj openCV aplikacija, razvijena su sučelja prema mnogo drugih jezika više razine kao npr. Java, MATLAB, Python, Ruby itd. Biblioteka sadrži više od 2500 optimiziranih algoritama za rad s računalnim vidom i strojnim učenjem koji imaju razne primjene kao što su otkrivanje i prepoznavanje lica, identificiranje objekata, praćenje pokreta kamere, praćenje pokretnih objekata, izrada 3D oblaka točaka pomoću stereo kamera, pronalaženje sličnih slika u bazi, praćenje pokreta očiju itd. OpenCV koristi više od 47 tisuća ljudi, a broj preuzimanja biblioteke prelazi 7 milijuna. Koriste ga mnoge tvrtke kao što su Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota i mnoge istraživačke grupe. [5] [6]

4. Izvedba rješenja

4.1. Detekcija tuna

Detekcija objekata postupak je pronalaženja objekata iz stvarnog svijeta kao što su lica, vozila ili zgrade u slikama ili video snimkama. Algoritmi za detekciju objekata najčešće koriste ekstrakciju značajki i algoritme učenja kako bi prepoznali instance iz kategorije objekata. Detekcija objekata često se koristi u primjenama dohvata slika, sigurnosti, nadzora i sustava automatskog parkiranja. [7]

Kako je video snimka zapravo niz slika, detekcija tuna radit će se na svakoj slici, odnosno na statičkim slikama. Postoje mnoge metode za detekciju objekata, a ovisno o uvjetima potrebno je odabrati optimalnu. Odabir metoda za detekciju objekata tako će ovisiti o mnogim parametrima kao što su vremenska složenost samog algoritma, kvaliteta snimljenog videa ili slike te odlikama samog ojekta i pozadine.

4.2. Detekcija i ekstrakcija značajki

Detekcija i opisivanje značajki slika ima značajnu ulogu u područjima obrade slike, računalnog vida i strojnog učenja. Značajke se mogu podijeliti u dvije grupe: globalne i lokalne. Globalne značajke uključuju sve piksele i opisuju sliku kao cjelinu dok lokalne značajke opisuju ključne točke unutar slike i regije oko njih. Detekcija značajki je postupak pronalaženja ključnih točaka nakon kojeg slijedi postupak opisivanja tih značajki.
Opisnik svake značajne točke mora pružiti dobru reprezentaciju svoje okoline kako bi se mogla upariti s lokalnim točkama drugih slika. Kako bi to bilo moguće, algoritam za detekciju i ekstrakciju značajki mora imati sljedeća svojstva: robusnost - algoritam bi trebao moći detektirati značajke neovisno o skali, rotaciji, pomaku, fotometrijskim deformacijama i šumu, ponovljivost - algoritam bi trebao detektirati iste značajke iste scene pod raznim uvjetima gledanja, točnost - algoritam bi trebao točno lokalizirati značajne točke, generalizacija - algoritam bi trebao detektirati značajke koje se mogu koristiti za razne primjene, brzina - algoritam bi trebao dovoljno brzo detektirati značajke na novim slikama kako bi mogao podržavati primjene u stvarnom vremenu, kvantiteta- algoritam bi trebao detektirati sve ili većinu značajki na slici. [9]

4.3. Algoritmi za detekciju i ekstrakciju značajki.

4.3.1. SIFT

Scale-invariant feature transform (SIFT) je algoritam za detekciju i opisivanje značajki kojeg je 1999. godine razvio kanadski znanstvenik David Lowe. Značajke koje SIFT generira invarijantne su na skalu i rotaciju, a djelomično i na afine transformacije te varijacije u osvjetljenju i promjene kuta gledanja. [9]

Algoritam se može podijeliti na 4 koraka:

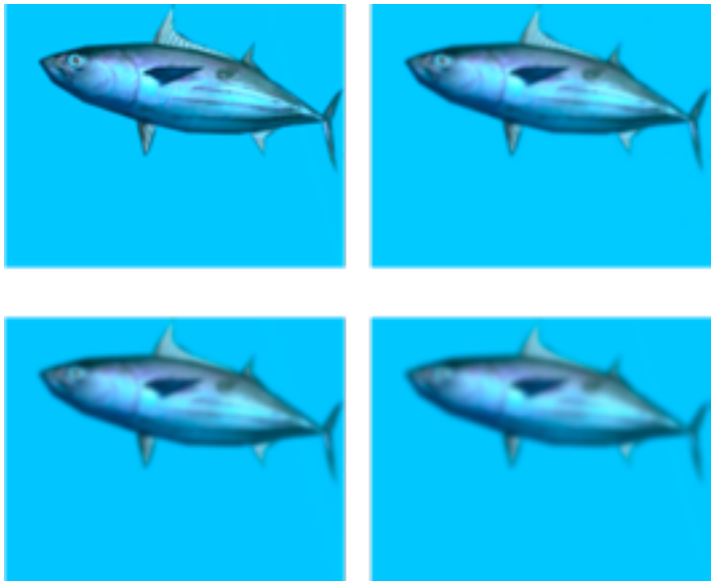
1. Detekcija ekstrema kroz niz skala (engl. Scale-space extrema detection)
Prvi korak algoritma, koristeći razlike Gaussovim filtrom zamućenih slika kao aproksimaciju Laplaceovog rubnog operatora, pronalazi ekstreme koji postaju točke od interesa (engl. interest keypoints).
Gaussov filter definiran je kao:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

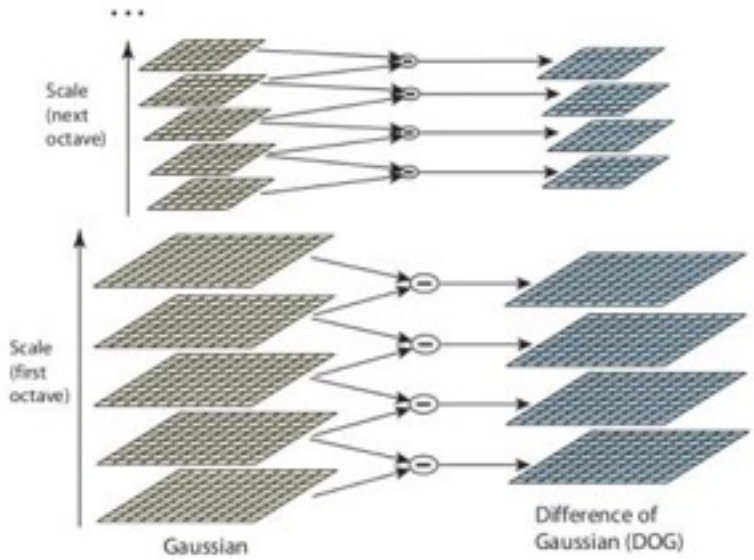
dok je Laplaceov rubni operator (engl. Laplacian of Gaussian; LoG) definiran kao:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

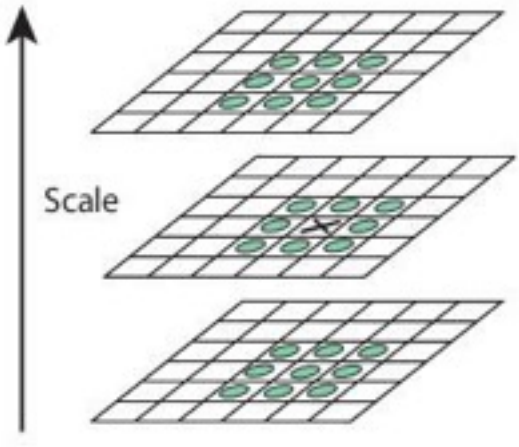
SIFT započinje tražeći točke u slikama koje bi zadovoljavale uvjete ponovljivosti, stabilnosti i
invarijantnosti. Invarijantnost točaka na različite skale može se postići koristeći detekciju na različitim skalama što se postiže stvaranjem Gaussove piramide. U Gaussovoj piramidi svaka razina naziva se oktava, a svaka oktava sastoji se od niza slika koje su zamućene Gaussovim filtrom s različitom standardnom devijacijom. Kod svake oktave, nova razina piramide zamućenih slika bit će jedna od zadnje dvije slike iz prethodne oktave, ali će joj rezolucija biti smanjena dva puta. Nakon što je izgrađena Gaussova piramida, gradi se piramida razlika slika zamućenih Gaussovim filtrom (engl. Difference of Gaussian). Detekcija ključnih točaka funkcionira traženjem lokalnih maksimuma tako da se svaki piksel uspoređuje s 8 susjednih piksela, ali i s 9 susjednih piksela na skalama koje su ispod i iznad njega u piramidi razlika Gaussovih zamućenja (Slika 3.).



Slika 1. Primjena Gaussovog filtra na sliku tune sa parametrima σ = 0, 1, 2 i 4



Slika 2. Izgradnja piramide razlika slika zamućenih Gaussovim filtrom [9]



Slika 3. Traženje lokalnih ekstrema[9]

2. Lokalizacija značajki

Nakon detekcije ekstrema, sljedeći korak algoritma preračunava njihove cjelobrojne lokacije i skale u realne. Takav pristup pridonosi preciznijoj lokalizaciji i većoj stabilnosti značajki.
Za svaki ekstrem interpolacijom se izračunava još točnija lokacija i skala na kojem je pronađen te se provjerava uvjet stabilnosti.
Kako bi se eliminirale nestabilne značajke SIFT koristi dvije metode: odbacivanje značajki na temelju slabog kontrasta i odbacivanje značajki na temelju slabih rubova.

3. Dodjeljivanje orijentacija

Nakon procesa lokalizacije i eliminacije SIFT značajkama pridjeljuje orijentaciju. Lowe je zaključio kako sljedeći pristup daje najstabilnije rezultate. Skala značajke određuje koju zamućenu sliku L, odabrati za daljnje izračune. Za svaki slikovni element odabrane slike L(x, y) izračuna se magnituda i orijentacija gradijenta prema sljedećim formulama:

$$m(x,y)=\sqrt{\left(L(x+1,y)-L(x-1,y)\right)^2+\left(L(x,y+1)-L(x,y-1)\right)^2}$$

$$\theta(x,y)=\tan^{-1}\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}$$

4. Izgradnja deskriptora

Nakon što su značajkama pridijeljene lokacija, skala i orijentacija zbog kojih je postignuta invarijantnost na skalu i rotaciju, sljedeći korak je postizanje invarijantnosti na promjene u osvjetljenju te na promjenu položaja kamere.

Do ovog posljednjeg koraka algoritma procesirane su sve značajke – pridijeljene su im lokacije, skale i orijentacije pomoću kojih je postignuta invarijantnost na skalu i rotaciju. U ovom koraku će se postići i invarijantnost na promjene u osvjetljenju te na promjene položaja kamere. Rješenje se temelji na radu Eldemana čiji pristup uključuje područje biološkog vida, odnosno simulaciju kompleksnih neurona iz primarne vizualne moždane opne. Neuroni odgovaraju gradijentu točno određene orijentacije, prostorne frekvencije i lokacije koja je podložna malim pomacima, a upravo su se oni pokazali kao odlično sredstvo za ostvarivanje invarijantnost na promjenu položaja kamere [13].

4.3.2. SURF

Algoritam SURF objavio je dr. Herbert Bay 2016. godine na Europskoj konferenciji računalnog vida. Kako mu i ime kaže (Speeded up robust features), prednost SURF algoritme je njegova brzina, te je nekoliko puta brži od algoritma SIFT [10]. Kod računanja ključnih točaka SURF koristi aproksimaciju determinanti Hessianove matrice za razliku od Gaussovih zamućenja kao kod SIFT algoritma. Prednost je što konvolucija kod ovog postupka može lako biti izračunata uz pomoć integralnih slika te se može računati paralelno na više skala. Integralna slika je struktura podataka za efikasno računanje sume vrijednosti u pravokutnoj matrici. Prvi put ju je opisao Frank Crow 1984. U području računalnog vida popularizirao ju je Lewis 1995. [11], a veliku primjenu stekla je u Viola-Jones algoritmu za detekciju objekata [12] . Integralna slika na poziciji T=(x,y) predstavlja sumu intenziteta svih piksela koji se nalaze unutar kvadrata omeđenog točkom T i ishodišta slike:

$$I_{\Sigma}(x) = \sum_{x=0}^{x \le x} \sum_{j=0}^{j \le y} I(i,j)$$

Slika 4. Suma intenziteta unutar kvadrata ABCD računa se formulom I = C-B-D+A

Kada je integralna slika jednom izračunata, postaje lako izračunati sumu intenziteta bilo koje kvadratne površine.
Za aproksimaciju Hessianove matrice koristi se Box filtar koji aproksimira Gaussovu derivaciju drugog reda te se računa jako brzo pomoću integralne slike.

Kako se ključne točke trebaju moći detektirani na različitim skalama potrebno ih je moći i uspoređivati na slikama u različitim skalama. Zbog primjene integralnih slika, kod SURF algoritma nije potrebno svaku skalu posebno zamutiti Gaussovim filtrom kao kod SIFT algoritma već se može direktno primjeniti box filter.
Početni sloj skale je filtar veličine 9x9 piksela, što odgovara aproksimaciji Gaussove derivacije s parametrom sigma = 1.2. Prostor skala podijeljen je na oktave a jedna oktava predstavlja seriju odaziva filtra koji se dobivaju konvolucijom polazne slike s filtrima kod kojih se dimenzije povećavaju.
SURF opisnik opisuje distribuciju intenziteta kroz susjedstvo ključnih točaka. Opisnik se gradi na temelju odziva Haarovih valića prvog reda u horizontalnom i vertikalnom smjeru iskorištavajući integralnu sliku što doprinosi brzini algoritma. Prvi korak je fiksiranje reproducibilne rotacije koja se temelji na informaciji dobivenoj iz kružne regije oko točke interesa. Nakon toga se konstruira kvadratna regija koja je poravnata s odabranom orijentacijom i iz nje se izvlači SURF opisnik [14].

4.3.3. Usporedba: SURF i SIFT

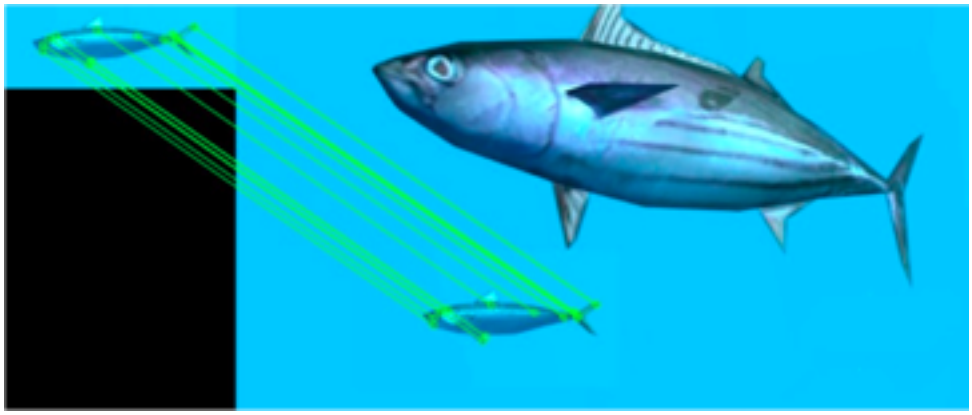
4.3.3.1. Usporedba brzine računanja ključni točaka i njihovih opisnika.

Za usporedbu brzine izvođenja algoritama SIFT i SURF korištene su funkcije za ekstrakciju i opisivanje značajki iz biblioteke OpenCV, a programsko rješenje izvedeno je u Javi. Kako bi mjerenje bilo što točnije, svaki postupak je mjeren 10 puta te je izračunata prosječna vrijednost. Oba algoritma detektirala su značajne

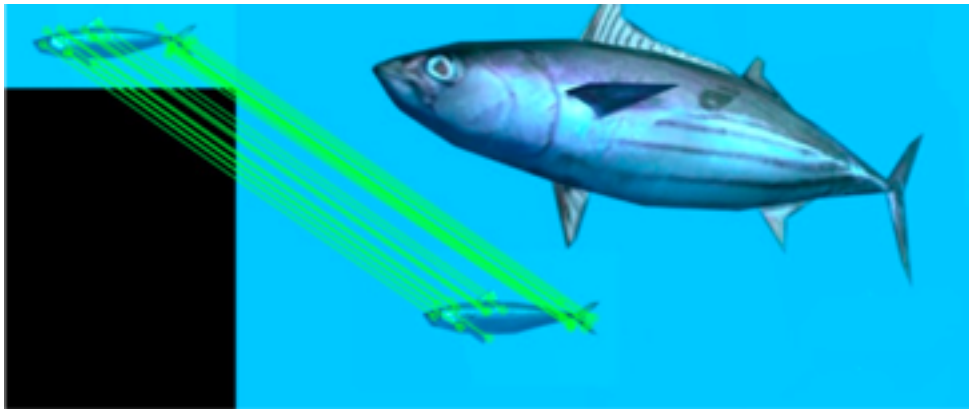
točke na istoj slici iz video simulacije akvarija s tunama. kao što je bilo i očekivano SURF algoritam bio je brži i to za 2.21 puta od SIFT algoritma.

Tablica 1. Usporedba trajanja SIFT i SURF algoritma

	SIFT	SURF
broj ponavljanja	**trajanje algoritma[ms]**	**trajanje algoritma[ms]**
1	836	265
2	939	394
3	802	311
4	700	267
5	690	306
6	697	359
7	667	367
8	640	357
9	675	352
10	690	340
prosijećno	733	331



Slika 5. detekcija SIFT algoritmom



Slika 6. detekcija SURF algoritmom

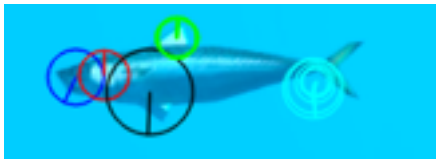
Kako je SURF algoritam višestruko brži od SIFT algoritma te je detektirao više ključnih točaka, odlučio sam koristiti SURF za detekciju tuna.

4.4. Detekcija tuna pomoću algoritma SURF

Kako bi se objekt na slici detektirao prvo je potrebno detektirati ključne točke te njihove opisnike na predlošku traženog objekta, zatim detektirati ključne točke i opisnike na slici iz video snimke te svaku ključnu točku predloška spojiti s ključnom točkom kojoj ona odgovara po opisniku na slici iz video snimke. Nakon što su ključne točke slike predloška pospajane s ključnim točkama slike, traži se geometrijska transformacija koja dobro povezuje točke, tj. odgovara li prostorni raspored točaka na slici onom na predlošku kako bi se moglo zaključiti da se radi o pronađenom objektu. Kako se vidi na Slici 5 i Slici 6, SIFT i SURF algoritam su uspješno detektirali sliku tune.

Problem kod ovog načina je taj što nije predviđen za detekciju više objekata na slici što je slučaj u videu tuna. Kako bi se detektiralo više istih objekata na slici potrebno je svaku ključnu točku detektiranu u predlošku tražiti na više mjesta na slici. Prvi zadatak kod detekcije je upravo odabir predloška objekta, tj. slike koja će predstavljati tunu. Tuna koja je odabrana da bude predložak mora biti što sličnija svim tunama koje se pojavljuju u video snimci kako bi njene ključne točke odgovarale i drugim tunama. Kako sve tune u video snimkama nisu identične i razlikuju se zbog kuta snimanja, odbljeska i općenito po obliku tijela, odabir slike predloška ključan je kako bi detekcija tuna uspješno funkcionirala. Nakon odabira

predloška slijedi detekcija ključnih točaka te izračun njihovih opisnika. Svaka ključna točka mora biti pouzdana, tako da bude detektirana na što više tuna u kadrovima video snimke, a dovoljno specifična da ne bude lažnih detekcija. Svaki opisnik zapravo je 64-dimenzionalni vektor pa se sličnost dvaju opisnika računa pomoću euklidske udaljenosti. Kako bi detekcija ispravno radila potrebno je odrediti koja može biti najveća granična udaljenost, a da opisnici opisuju istu ključnu točku. Postavljanje granične udaljenosti veće od optimalne rezultiralo bi pogrešnim detekcijama, a kod premale vrijednosti bi ispravne detekcije prošle nezabilježeno. Granična vrijednost odabrana je tako da se za 20 kadrova gledao broj nedetektiranih i broj lažno detektiranih ključnih točaka te za koju će graničnu vrijednost taj zbroj biti manji. Također je bitno da broj ključnih točaka ne bude prevelik kako bi algoritam mogao raditi u stvarnom vremenu. Proučavanjem pojavljivanja ključnih točaka, odabrano je njih 5 koje se koriste u detekciji tuna.



Slika 7. Odabrane klučne točke na slici predloška

Postupak detekcije može se podijeliti u 3 koraka:

1. Pronalaženje svih ključnih točaka na slici kadra te računanje njihovih opisnika
2. Filtriranje ključnih točaka prema njihovim opisnicima tako da se iz skupa detektiranih izbace sve koje ne odgovaraju 5 inicijalno odabranih točaka
3. Pripajanje ključnih točaka detektiranim tunama. Za svaku ključnu točku provjerava se postoji li u kadru već detektirana tuna kojoj bi ta ključna točka prema svojoj poziciji i veličini mogla pripasti, a ako se takva tuna ne pronađe, stvara se novi objekt tune kojoj se točka pripaja

Nakon što su iz detektiranih ključnih točaka stvoreni objekti tuna, računaju se središta tuna kako bi ih se moglo lakše pratiti.

4.5. Praćenje tuna

Praćenje riba je postupak koji omogućuje njihovo brojanje i osigurava da neka riba ne prođe nezabilježeno ili da se pribroji više puta. Važno je da se riba prati od trenutka kada uđe u kadar dok ne izađe iz njega, a bitno je i da algoritam radi kod međusobnog zaklanjanja tuna što je čest slučaj.

Praćenje je nalaženje putanje pokretnog objekta na temelju niza slika i u općenitom slučaju vrlo je zahtjevan problem. Međutim, korištenjem specifičnosti konkretne

primjene, problem se može pojednostavniti, pa je moguće ostvariti sustav koji uspješno i učinkovito prati određenu vrstu objekata u nekom rasponu uvjeta osvjetljenja. Vizualno praćenje se danas koristi u raznim primjenama kao što su: automatsko nadgledanje (sigurnost, praćenje prometa, automatizacija industrijskih postrojenja), prepoznavanje temeljeno na pokretu, interakcija između čovjeka i računala, sažimanje video materijala, indeksiranje video materijala, 3D rekonstrukcija, navigacija vozila. Uz tako široku primjenu došlo je i do razvoja algoritama za praćenje objekata. Iako biblioteka OpenCV dolazi s algoritmima koji omogućuju praćenje objekata, za potrebe ovog projekta napisan je jednostavan algoritam za praćenje koji u obzir uzima specifičnosti snimljenog videa.

4.5.1. Algoritam praćenja objekata

Kako bi se omogućilo što preciznije praćenje tuna, potrebno je razmotriti način na koji se one kreću te na osnovu tih parametara razviti algoritam za praćenje. Cilj algoritma je prepoznati novu tunu kada se pojavi u videu te ju povezati sa svim sljedećim pojavljivanjima te iste tune u sljedećim slikama videa. Osnovna ideja je da se omogući spremanje praćenih tuna u listu te da se prolazeći kroz video sliku po sliku sve detektirane tune pokušaju povezati s nekom od već spremljenih detekcija iz liste. Navedena lista sadržava popis tuna koje se trenutno prate te sve detektirane pozicije praćenih tuna. Kada dobije pozicije svih detektiranih tuna u nekoj slici, algoritam za praćenje bi za svaku tunu morao razabrati radi li se o novoj tuni ili o tuni koja je već detektirana u prijašnjim kadrovima.
Kod povezivanje tune iz trenutne i prethodne slike algoritam uzima u obzir specifičnosti njihova pokreta. Ako se radi o videu u kojem se tune uvijek kreću u istom smjeru te imaju približno konstantu brzinu moguće je predvidjeti na kojoj poziciji će se nalaziti tuna na sljedećoj slici. Upravo predviđanje pozicija omogućuje kvalitetnije praćenje tuna jer se tada pozicije detektiranih tuna ne povezuju s pozicijama prijašnjih tuna već s predikcijama njihovih budućih pozicija.

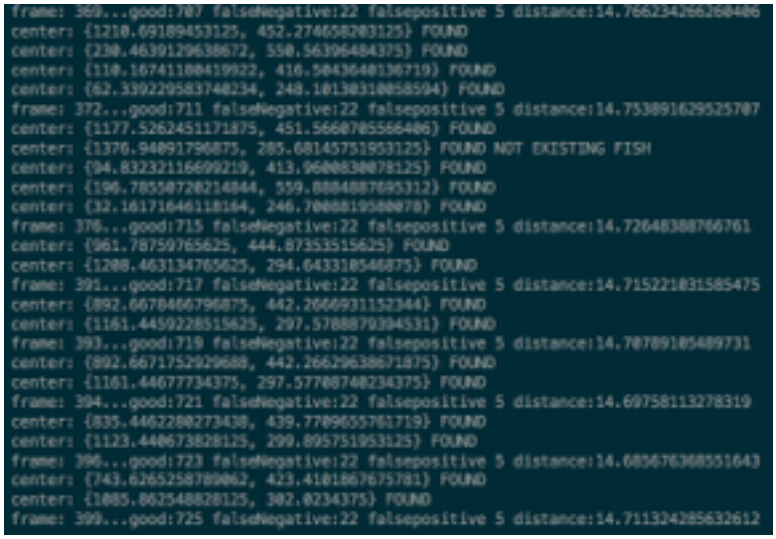
Algoritam za predviđanje buduće pozicije tune zasnovan je na pretpostavci da se sve tune gibaju jednoliko pravocrtno, što približno odgovara stvarnoj situaciji. Predviđena pozicija izračunata je tako da se izračuna pomak između trenutne i prethodne pozicije i taj pomak doda trenutnoj poziciji

5. Rezultati

Glavni cilj ovog projekta je sa što većom preciznošću izbrojati tune u video snimci. Kako je proces brojanja tuna, počevši od same detekcije pa do praćenja tuna podložan mnogim krivim procjenama, logično je za očekivati da se dobiveni rezultati neće u potpunosti poklapati sa stvarnim mjerenjima.

Jedini način na koji se može provjeriti točnost detekcije i brojanja jest ručno označavanje i brojanje. Za označavanje tuna implementirana je programska podrška koja omogućuje da se za svaki kadar ručno označe pozicije tuna te spreme označene pozicije. Nakon što su tune označene, program je svaku detekciju mogao usporediti sa spremljenim oznakama te zaključiti radili se o ispravno detektiranoj tuni, lažnoj detekciji tune ili tuna uopće nije detektirana.
Postupak označavanja i provjere točnosti napravljen je na prvih 400 kadrova te je od ukupno 744 pojavljivanja tuna, tuna detektirana 725 puta, dok je 5 puta detektirana nepostojeća tuna. Ukupno je 97.05% svih tuna detektirano, a ako se pogrešci pribroje i lažno detektirane tune, ona iznosti 3.7%.

Postupak označavanja i provjere točnosti napravljen je na prvih 400 kadrova te je od ukupno 744 pojavljivanja tuna, tuna detektirana 725 puta, dok je 5 puta detektirana nepostojeća tuna. Ukupno je 97.05% svih tuna detektirano, a ako se pogrešci pribroje i lažno detektirane tune, ona iznost 3.7%.



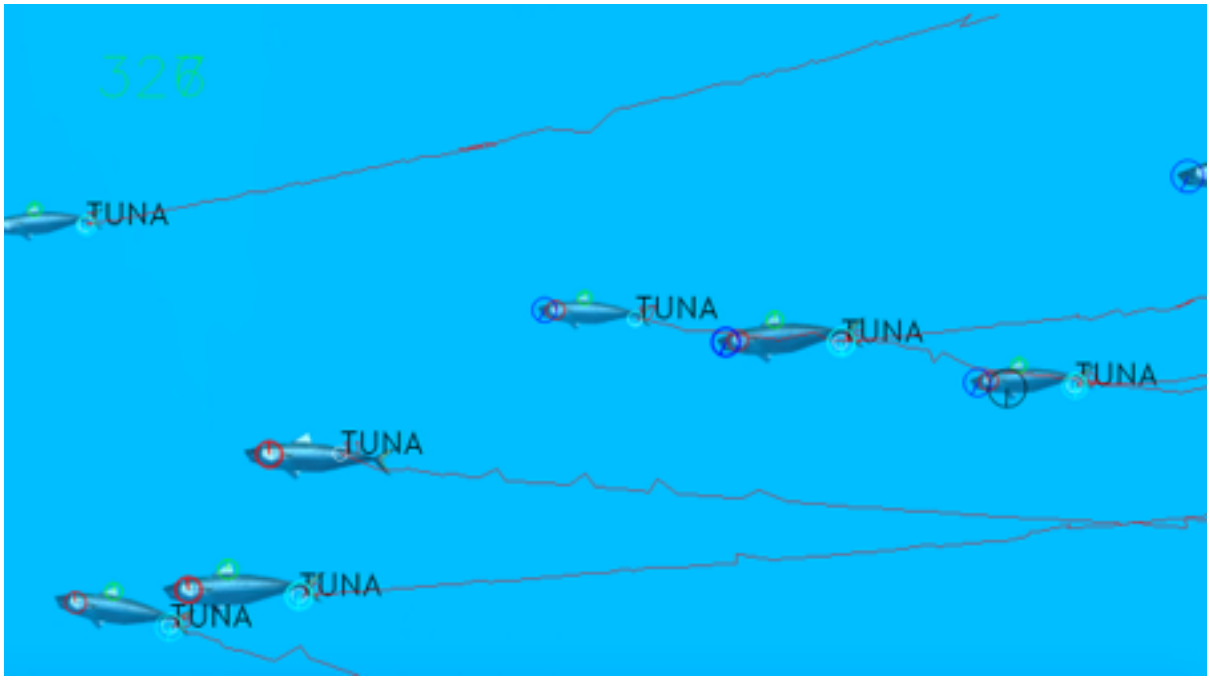
Slika 7. Ispis rezultata detekcije

Kako bi se provjerila točnost prebrojavanja tuna, također ih je potrebno ručno prebrojati u praćenom video isječku. Program je implementiran tako da iscrtava putanju svake praćene tune te uz nju ispisuje redni broj, tj. koja je ona po redu detektirana tuna. Nakon pregledavanja video snimke, izbrojano je 134 tune dok je program izbrojao 121 što je pogreška od 9.7%. Velika pogreška ponajviše je rezultat višestrukog brojanja iste tune. Najčešće se takva pogreška događa kada se praćena

tuna zakloni iza druge veće tune, te se nakon nekog vremena opet otkloni pa ju program broji kao novu tunu (slika 8.). Moguće rješenje ovog problema bilo bi praćenje tuna koje ulaze u kadar na početku slike, ali tada bi se izostavilo brojanje

onih tuna koje su zaklonjene baš pri ulasku u sliku.

Slika 8. Zaklanjanje tuna. Praćena tuna zaklonjena je od velike tune na središnjoj slici te se ponovo prati kao nova tuna na desnoj slici.



Slika 9. Praćenje tuna

5.1. Vremenska složenost algoritma

Osim točnosti algoritma, još jedan parametar je i njegova vremenska složenost te mogućnost rada algoritma u stvarnom vremenu. Ukupno vrijeme može se podijeliti

na dva djela: vrijeme detekcije i vrijeme praćenja. Vrijeme praćenja je vrijeme potrebno za spajanje tuna s trenutne slike s tunama detektiranim u prethodnoj slici u videu i ono je zanemarivo u odnosu na vrijeme detekcije tako da je ukupna brzina algoritma proporcionalna vremenu detekcije. Vrijeme potrebno za detekciju ovisi i o samoj slici pa je potrebno uzeti prosječno vrijeme. Mjerenjem detekcije na 100 slika izračunato je prosječno vrijem detekcije od 325ms. Kako se u jednoj sekundi video snimke prikaže 25 sličica slijedi da je potrebna brzina detekcije 40ms te da algoritam u trenutnoj implementaciji ne može raditi u stvarnom vremenu.

6. Zaključak

Brojanje tuna pomoću tehnika računalnog vida jedno je od rješenja koje bi zasigurno olakšalo poslovanje uzgajivačima tuna. Ovo programsko rješenje dalo je vrlo dobre rezultate ali glavni nedostatak je taj što je razvijeno na simulaciji, a ne stvarnim snimkama. Iako je simulacijom pokušano što više simulirati stvarne uvijete kao što su odbljesci i zaklanjanja tuna, teško je utvrditi kolika bi bila točnost algoritma na stvarnoj snimci.
Rad u stvarnom vremenu nije uvjet kako bi program radio ispravno, ali ako bi se htjelo ubrzati rad programa, postoje metode s kojima bi se to moglo postići. Brzina detekcije najviše ovisi o dimenzijama slike pa bi se smanjivanjem rezolucije videa sigurno ubrzao rad programa, ali bi se izgubili detalji sa slike pa bi trebalo paziti da detekcija još uvijek radi ispravno. Drugi način je da se pretraživanje ključnih točaka ograniči na područja koja nisu pozadina, što ne bi predstavljao problem pošto je pozadina uvijek plave boje pa bi ju bilo lako označiti.
Ako bi došlo do suradnje s uzgajivačima tuna pa tako i adaptacije algoritma za rad u stvarnim snimkama bilo bi potrebno odrediti nove ključne točke te njihove opisnike.

Osim traženja ključnih točaka, jedno od rješenja koje bi omogućilo detektiranje tuna na snimkama snimljenim u različitim uvjetima i rad u stvarnom vremenu je korištenje strojnog učenja. Strojno učenje je programiranje računala tako da se optimizira neki kriterij uspješnosti temeljem podatkovnih primjera ili prethodnog iskustva. Uz dovoljan skup pozitivnih primjera, mogao bi se stvoriti klasifikator koji bi raspoznao tune neovisno o uvjetima pod kojim su snimljene.

7. Popis litetrature

[1] Topolnik, M. i Kušek, M. (2008). Uvod u programski jezik Java, Skripta uz kolegij Informacija, logika i jezici. Zagreb: Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.

[2]Number of Java Developers. (2010). Preuzeto 20. travnja 2017. sa stranice http://www.numberof.net/number-of-java-developers/

[3]Čupić, M. (2014). Programiranje u Javi. Zagreb: Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.

[4] https://hr.wikipedia.org/wiki/Java_(programski_jezik) (2016.) Preuzeto 20. travnja 2017.

[5] Bradski, G. Kaehler, A. Learning OpenCV. Computer Vision with the OpenCV Library. Sebastopol: O'reilly Media, Inc. 2008.

[6] About OpenCV | OpenCV website. Preuzeto 21. travnja 2017. s http://opencv.org/about.html

[7] Object detection in computer vision. Preuzeto 21. travnja s https://www.mathworks.com/discovery/object-detection.html

[8] Ali Ismaid Awad, Mahmoud Hassaballah (2016.) Image Feature Detectors and Descriptors: Foundations and Applications. Egypt: Faculty of engineering AL Azhar University, Egypt

[9] Introduction to SIFT (Scale-Invariant Feature Transform) Preuzeto 22. travnja 2017. s http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

[10] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L., (2008.) Speeded – Up Robust Features,
Computer Vision and Image Understanding (CVIU)

[11] Lewis, J.P. (1995). Fast template matching. Proc. Vision Interface. pp. 120–123

[12] Viola, P., Jones, M.J. (2001.) Robust Real-Time Object Detection. USA: One Cambridge Center Cambridge, Massachusets

[13] Edelman, S., Intrator, N., i Poggio, T. (1997.) Complex cells and object recognition. USA: Center for Biol & Comp Learning Cambridge, Massachusets

[14] Bay, H., Tuytelaars, T. i Van Gool, L. (2006.) SURF: Speeded up robust features. Proceedings of the European Conference on Computer Vision

[15] Bradski G. I Kaehler A. (2008.) Learning OpenCV, prvo izdanje, Sebastopol US, O'Reilly Media,

8. Sažetak

Adrian Žgaljić

Detekcija i brojanje pokretnih objekata u videosnimkama

Ovaj rad obrađuje temu izrade programskog rješenja za brojanje pokretnih objekata u videosnimkama. Baziran je na razvoju rješenja koje bi omogućilo brojanje tuna u kaveznim uzgajalištima tuna. U radu su obrađene metode za detekekciju i praćenje objekata. Detekcija objekata postignuta je uz pomoć algoritama za detekciju i ekstrakciju značajnih točaka SURF i SIFT. Programsko rješenje razvijeno je koristeći biblioteku otvorenog koda OpenCV te je implementirano u programskom jeziku Java. Razvijeno programsko rješenje u mogućnosti je izbrojati tune s točnošću od 90.3%, ali je testirano na video simulaciji te bi ga za rad na pravim video snimkama trebalo prilagoditi.

Ključne riječi: detekcij, praćenje, video, tune, OpenCV

Detection and counting of moving objects in video sequences

This thesis examines the topic of software development process for counting of moving objects in video sequences. It is based on developing solution that will enable counting of tuna fish in fish farming facilities. Various methods for object detection and tacking are described. Object detection is accomplished with algorithms for feature detection and extaction: SIFT and SURF. Software is developed using open source libray OpenCV and it is implemented in Java language. Developed software is able to count fishes with accuracy rate of 90.3 percent , but it is developed and tested using video simulation and it should be adapted in order to work in real video sequences.

Keywords: detection, tracking, video, tuna fish, OpenCV