****

UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

**NASLOV DIPLOMSKOG RADA**

Diplomski rad

Studijski program: Elektrotehnika i računarstvo

Modul: Računarstvo i informatika

Student: Mentor:

Ivana Milivojević, 16704 Prof. dr Aleksandar Milosavljević

Niš, septembar 2022. godina

Univerzitet u Nišu

Elektronski fakultet

**NASLOV DIPLOMSKOG RADA**

**NASLOV NA ENGLESKOM**

Diplomski rad

Studijski program: Elektrotehnika i računarstvo

Modul: Računarstvo i informatika

Student: Ivana Milivojević, 16704

Mentor: Prof. dr Aleksandar Milosavljević

Zadatak: *Ovde ide zadatak napisan italic stilom*

Datum prijave rada: xx.xx.xxxx

Datum predaje rada: xx.xx.xxxx

Datum odbrane rada: xx.xx.xxxx

Komisija za ocenu i odbranu:

1. Prof. dr Aleksandar Milosavljević, Predsednik Komisije
2. Prof. dr Ime i Prezime, Član
3. Prof. dr Ime i Prezime, Član

NASLOV RADA

**SAŽETAK**

Moderni tehnološki procesi zahtevaju složene načine pokretanja i upravljanja višemotornim pogonima. Decentralizovano upravljanje u ovim složenim sistemima ima veoma važnu ulogu zbog potrebe povećanja produktivnosti, kvaliteta i brzine proizvodnje, ekonomičnosti, kao i uštede električne energije. U radu se prvo izlažu načini funkcionalne i mehaničke povezanosti višemotornih pogona u cilju sistematične analize i određivanja optimalne upravljačke konfiguracije. Zatim je definisan pojam decentralizovanog upravljanja i ukazano je na prednosti korišćenja ovog načina upravljanja u odnosu na centralizovano upravljanje. Kao primer jednog višemotornog pogona sa elastičnom mehaničkom vezom pogonskih vratila, koji se najčešće sreću u industriji, opisan je višemotorni pogon sistema za premotavanje. Data je principijelna šema, opisan je način funkcionisanja, a zatim je izvršeno njegovo detaljno matematičko modelovanje. Sistemi za premotavanje imaju veoma značajnu primenu u industriji za proizvodnju i obradu papira, gume, metala, plastičnih folija, tekstila i drugih tankih materijala sa elastičnim svojstvima. Sistemi za premotavanje se generalno sastoje iz istih elemenata, bez obzira na to o kakvim se proizvodima radi. Nezavisno od faze tehnološkog procesa, potrebna je visoka tačnost regulisanja brzine, a u toku proizvodnje potrebna je i visoka tačnost kod regulacije sile zatezanja trake.

Glavni cilj ovog rada je da se projektuje uspešan upravljački sistem višemotornog pogona sistema za premotavanje. Za ostvarivanje ovog zahteva korišćen je upravljački sistem koji je zasnovan na regulacionim petljama sa PI regulatorima. Podešavanje PI regulatora je veoma intuitivno, jednostavno i prihvaćeno u mnogim aplikacijama. Sa PI regulatorima se može postići zadovoljavajući kompromis u performansama u pogledu brzine odziva sistema, robusnosti i stabilnosti. Na osnovu matematičkog modela višemotornog pogona sistema za premotavanje napravljen je simulacioni model u programskom paketu MATLAB/Simulink i izvršeno je snimanje važnih procesnih veličina pomoću kojih su utvrđena teoretska razmatranja. Sagledavanjem rezultata dobijenih računarskom simulacijom, može se zaključiti da je primenjeni postupak decentralizovanog upravljanja sa PI regulatorima linijskih brzina i sila zatezanja višemotornog pogona sistema za premotavanje veoma efikasan jer se dobijaju zadovoljavajuće performanse i stabilnost sistema, uprkos promeni dinamike tokom procesa premotavanja. Linijske brzine kretanja trake i sile zatezanja trake se regulišu na adekvatan način u unapred dozvoljenom opsegu za vreme rada sistema za premotavanje, što se zahteva tehnološkim procesom.

**Ključne reči**: višemotorni električni pogoni, decentralizovano upravljanje, sistem za premotavanje, PI regulator.

MASTER (BACHELOR) THESIS TITLE

**ABSTRACT**

Modern technology processes require complex algorithms for speed and torque control of multi-motor drives. Decentralized control of these complex systems has a very important role because of the need to increase productivity, quality and speed production, costeffectiveness, and energy savings. In this work are first presented types of functional and mechanical connections between drives within the multi-motor drives in order to systematic analysis and determine the optimal control scheme. Then it is defined the concept of decentralized control and also are pointed out the advantages of using this method of control in relation to centralized control. As an example of multi-motor drives with elastic connection between drives, it is described the multi-motor drive of web winding system. It is given principle scheme, described way of functioning and then is done mathematical modeling of multi-motor drive of web winding system. Multi-motor drive of web winding systems have a very important role in the industry for manufacturing and processing of paper, metal, plastic film, textiles and other thin materials with elastic properties. Web winding systems are generally consisted of the same machine elements in spite of the diversity of the transported products. Regardless of the stage of the technology process, high accuracy of velocity and tension control for web is required.

The main aim of this work is to design a successful decentralized control system for multi-motor drive of web winding system. For the achievement of this requirement, it was used control system which is based on a control loop with PI controllers. Tuning of PI controllers is very intuitive, simple and accepted in many applications. With PI controllers can be achieved a satisfactory compromise in performance in terms of speed of system response, robustness and stability. Simulation model of multi-motor drive of web winding system it was made in software MATLAB/Simulink based on the mathematical model, and then were done set of simulations. Reviewing the results, obtained by computer simulation, it can be concluded that the applied method of decentralized control with PI controllers of linear speed and tension for multi-motor web winding system is very efficient because the system is stabile, despite the fact of changing dynamics during the winding/unwinding process. Velocities and tensions of web are regulated on adequately way in allowable range during the winding/unwinding process, that is required by technology process.

**Keywords**: multi-motor electrical drives, decentralized control, web winding system, PI controller.

**SADRŽAJ**

**1. UVOD ................................................................................................................................... 7**

1.1. Teorijske osnove diplomskog (bečelor) rada (Akreditacija 2008) .................................... 7

1.2. Teorijske osnove master rada (Akreditacija 2013)............................................................. 8

1.3. Organizacija diplomskog (master) rada.............................................................................. 9

**2. PREGLED LITERATURE............................................................................................... 10**

2.1. Pravilno korišćenje izvora informacija – citiranje literature ............................................ 10

2.2. Preporuke za formatiranje teksta ...................................................................................... 10

**3. TEORIJSKE OSNOVE..................................................................................................... 12**

3.1. Naslov podpoglavlja ........................................................................................................ 12

3.2. O plagijarizmu .................................................................................................................. 12

**4. EKSPERIMENTALNI REZULTATI.............................................................................. 13**

**5. ZAKLJUČAK..................................................................................................................... 15**

**LITERATURA ...................................................................................................................... 16**

**DODATAK 1. OPŠTE NAPOMENE................................................................................... 17**

**DODATAK 2. KOD MATLAB PROGRAMA ................................................................... 18**

DETEKCIJA OSOBA

Detekcija objekata je tehnologija koja spada u oblast računarskog vida i bavi se pronalaženjem objekata koji pripadaju određenim klasama (npr. ljudi, zgrade, automobili) na digitalnim slikama i video snimcima. Koristi se i za praćenje objekata, npr. praćenje osoba na video snimcima ili praćenje lopte tokom fudbalskog meča.

Metode za detekciju objekata se mogu podeliti na one koje su bazirane na neuronima i one koje nisu. Kod metoda koje nisu zasnovane na neuronima neophodno je definisati feature i nakon toga koristiti neku od tehnika za klasifikaciju, poput SVM (support vector machine). Neuronske tehnike mogu detektovati objekte bez definisanja feature-a i najčešće su bazirane na konvolucionim neuronskim mrežama (CNN). Primeri metoda koje nisu zasnovane na neuronima: Viola–Jones object detection framework based on Haar features, Scale-invariant feature transform (SIFT), Histogram of oriented gradients (HOG) features. Primeri metoda koje su zasnovane na neuronima: Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN, cascade R-CNN), Single Shot MultiBox Detector (SSD), You Only Look Once (YOLO), Single-Shot Refinement Neural Network for Object Detection (RefineDet), Retina-Net, Deformable convolutional networks. [17]

Detekcija objekata može se izvoditi korišćenjem tradicionalnih tehnika obrade slika ili savremenih metoda dubokog učenja.

-Tehnike procesiranja slike ne zahtevaju podatke za treniranje, nenadgledane su prirode i ne zahtevaju anotirane slike koje bi ljudi ručno označavali. Sa druge strane, ograničene su usled postojanja komplekne pozadine, delimično prekrivenih objekata, osvetljenja i šuma. Era detektora objekata krenula je sa Viola-Jones Detector (2001), HOG Detector (2006) i DPM (2008) gde je predstavljena bounding box regression.

-Tehnike dubokog učenja zavise od nadgledanog i nenadgledanog učenja, pri čemu su nadgledane metode standard u računarskom vidu. Performanse su ograničene snagom GPU-a, koja je veća iz godine u godinu. Značajno su otpornije na prekrivenost objekata, kompleksne scene i promene u osvetljenju. Nedostatak je što zahtevaju veliku količinu podataka za treniranje i proces označavanja slika je veoma zahtevan. Svakako, veliki referentni dataset-ovi ([MS COCO](https://viso.ai/computer-vision/coco-dataset/), Caltech, KITTI, PASCAL VOC, V5) nude dostupne, unapred pripremljene podatke. Danas je duboko učenje za detekciju objekata je široko prihvaćena metoda i koristi se u izradi komercijalnih proizvoda. Period metoda dubokog učenja krenuo je 2014. i alrogitmi se mogu podeliti na two-stage i one-stage algoritme. Detektori objekata bazirani na dubokom učenju izdvajaju feature sa izlazne slike. Detektor obavlja 2 zadatka: 1. pronalaženje određenog broja objekata i nakon toga 2. klasifikacija svakog od njih i određivanje njegove veličine obuhvatajućim pravougaonikom. Ovaj proces se može razdvojiti na 2 faze, a može se i izvršiti u okviru jednog koraka kako bi se dobile bolje performanse, ali po cenu pouzdanosti. Detektori koji rade u 2 faz**e podrazumevaju pronalaženje regiona objekata konvencionalnim metodama računarskog vida ili metodama dubokog učenja, koje je praćeno klasifikacijom zasnovanom na izdvojenim features iz detektovanih regiona i** bounding-box regresijom. Ovi metodi dostižu veliku pouzdanost, ali su sporiji. Najpoznatiji algoritmi ovog tipa su region convolutional neural network (2014), Fast RCNN and Faster RCNN (2015), Mask R-CNN (2017), granulated RCNN (G-RCNN) (2021). Detektori koji rade u jednom koraku predviđaju bounding boxes na slici bez prethodno izdvojenih regiona od interesa. Brži su, strukturno jednostavniji i mogu se koristiti u aplikacima od kojih se očekuje da mogu da rade u realnom vremenu. Najpoznatiji algoritmi iz ove grupe su YOLO (2016) koji ima i nekoliko kasnijih verzija, SSD (2016), i RetinaNet (2017). YOLO i MobileSSD su dali najbolje rezultate. MobileSSD je optimizovan za rad na CPU, a YOLO za GPU.

Kako se upoređuju algoritmi za detekciju objekata? Najpopularniji benchmark je Microsoft COCO dataset. Modeli se obično upoređuju na osnovu postignute Mean Average Precision (MAP) metrike. Na primer, YOLO detektor objekata je više od 1000 puta brži od R-CNN i 100 puta brži od Fast R-CNN.

SSD is a popular one-stage detector that can predict multiple classes. The method detects objects in images using a single [deep neural network](https://viso.ai/deep-learning/deep-neural-network-three-popular-types/) by discretizing the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. The object detector generates scores for the presence of each object category in each default box and adjusts the box to better fit the object shape. Also, the network combines predictions from multiple feature maps with different resolutions to handle objects of different sizes. The SSD detector is easy to train and integrate into software systems that require an object detection component. In comparison to other single-stage methods, SSD has much better accuracy, even with smaller input image sizes.

MobileNet is a single-shot multi-box detection network used to run object detection tasks. This model is implemented using the Caffe framework. The model output is a typical vector containing the [tracked object](https://viso.ai/deep-learning/object-tracking/) data, as previously described.

[18] Detekcija objekata nam daje odgovor na pitanje koji objekti se nalaze na slici i gde se nalaze. Primeri algoritama za detekciju objekata su Haar cascades, HOG + Linear SVM, i detektori bazirani na dubokom učenju kao što su Faster R-CNNs, YOLO, i Single Shot Detectors (SSDs). Važno je napomenuti da su detektori zasnovani na dubokom učenju veoma zahtevni u smislu kompjuterskog posla. Pogledaćemo kako funkcionišu Single Shot Detectors and MobileNets. Kada se koriste zajedno, mogu obezbediti veoma brzu detekciju objekata i na uređajima koji imaju ogrenićene resurse.

Tri najpoznatija detektora koja su zasnovana na dubokom učenju su:

1) Faster R-CNNs (Ren et al., 2015)

2) You Only Look Once (YOLO) (Redmon et al., 2015)

3) Single Shot Detectors (SSDs) (Liu et al., 2015)

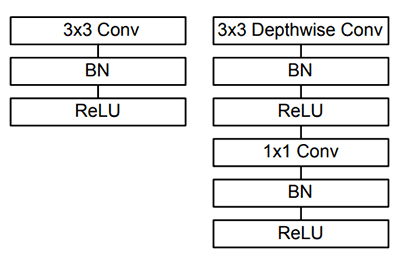
Faster R-CNNs (where the “R” stands for “Region Proposal”) su verovatno najpoznatiji metod za detekciju objekata korišćenjem dubokom učenja, mežutim, ova tehnika može biti teška za razumevanje i implementaciju, i izazovna za treniranje, i veoma spora za izvršavanje (oko 7FPS).

YOLO algoritam je mnogo brži, može dostići izvršenje od 40-90 FPS na Titan X GPU. Najbrže verzije YOLO mogu dostići i do 155 FPS. Problem sa ovim algoritmom je manja preciznost.

SSDs, originalno implementirani od strane Google, su kompromis imeđu 2 prethodno navedena.Algoritam je jednostavniji za razumevanje od Faster R-CNNs. Može se dobiti i veći FPS u zavisnosti od odabira mreže. Pouzdaniji su od YOLO.

**Implementacija**

**MobileNets: Efficient (deep) neural networks**



**Figure 2:** (Left) Standard convolutional layer with batch normalization and ReLU. (Right) Depthwise separable convolution with depthwise and pointwise layers followed by batch normalization and ReLU (figure and caption from Liu et al.).

Kada pravimo mrežu za detekciju objekata obično koristimo neku od postojećih arhitektura, poput VGG ili ResNet, a onda ih koristimo u okviru object detection pipeline. Problem je što ove mrežne arhitekture mogu biti veoma velike (reda 200-500MB). Ovakve arhitekture nisu pogodne za uređaje koji su resursima organičeni zbog previke veličine i potrebnog računanja. Umesto njih, mogu se koristiti MobileNets (Howard et al., 2017). [22] Tako se zovu zato što su dizajnirane za uređaje koji su resursima ograničeni, poput mobilnih telefona. Razlikuju se od tradicionalnih CNNs u korišćenju depthwise separable convolution. The general idea behind depthwise separable convolution is to split convolution into two stages:

A 3×3 depthwise convolution.

Followed by a 1×1 pointwise convolution.

This allows us to actually reduce the number of parameters in our network.

Ovde je problem što se žrtvuje pouzdanost, ali s druge strane, zahtevaju mnogo manje resursa.

Ako ukombinujemo MobileNet arhitekturu i Single Shot Detector (SSD) framework, dobićemo brz i efikasan metod za detekciju objekata zasnovan na dubokom učenju. Model koji ćemo koristiti je Caffe version of the [original TensorFlow implementation](https://github.com/Zehaos/MobileNet) [23] and was trained by chuanqi305 [24]. The MobileNet SSD je prvo trenirana na [COCO dataset](http://cocodataset.org/) (Common Objects in Context) [25] i nakon toga fien fine-tuned on PASCAL VOC dostižući 72.7% mAP (mean average precision). Možemo detektovati 20 različitih klasa objekata (+1 za klasu koja predstavlja pozadinu): avioni, bicikle, ptice, brodovi, flašes, autobusi, automobili, mačke, stolice, krave, stolovi za ručavanje, psi, konji, motori, ljudi, biljke u saksijama, ovce, vozovi, sofe i tv ekrani.

**Kod**

In this section we will use the MobileNet SSD + deep neural network (dnn) module in OpenCV to build our object detector.

DETEKCIJA LICA

Detekcija lica je računarska tehnika koja se koristi za identifikaciju ljudskog lica na digitalnim slikama. Može se posmatrati kao specifičan slučaj detekcije, odnosno, klasifikacije objekata. Kod klasifikacije objekata zadatak je pronaći i locirati sve objekte na slici koji pripadaju određenoj klasi. Npr. pronaći pešake, automobile, znakove pored puta... Detekcija lica se odnosi na ispitivanje da li se na slici ili video snimku nalazi ljudsko lice, gde je locirano i koje je veličine, nezavisno od njegovog položaja i osvetljenja. [1]

Detekcija lica je prvi korak bilo koje obrade lica, kao što su nalaženje karakterističnih tačaka lica, prepoznavanje lica, detekcija pokreta lica, ... Ima široku primenu u različitim oblastima poput biometrije, bezbednosti i zabave. Pronalaženje lica je izazov za računarski vid jer je neophodno uzeti u obzir orijentaciju lica, izraz lica, osvetljenje, prekrivenost lica (npr. naočare, kosa, brada, kapa), rezoluciju slike, kompleksnost pozadine (prisustvo velikog broja objekata), boju kože itd. [2]

Metode i algoritmi za detekciju lica

Metode

[3] [4] Yang, Kriegman, and Ahuja predstavili su u svom radu klasifikaciju metoda za detekciju lica u 4 kategorije. Algoritmi za detekciju lica mogu pripadati dvema ili većem broju kategorija.

1. Feature-based metode lociraju lice izdvajanjem strukturnih karakteristika lica poput nosa, usta i očiju. Najpre se trenira klasifikator koji se nakon toga koristi za razlikovanje regiona na kojima je lice i na kojima nije lice.

-Haar Feature Selection relies on similar properties of human faces to form matches from facial features: location and size of the eye, mouth, bridge of the nose, and the oriented gradients of pixel intensities. There are 38 layers of cascaded classifiers to obtain the total number of 6061 features from each frontal face. You can find some pre-trained classifiers here. Histogram of Oriented Gradients (HOG) is a feature extractor for object detection. The features extracted are the distribution (histograms) of directions of gradients (oriented gradients) of the image.

-Histogram of Oriented Gradients (HOG) is a feature extractor za detekciju objekata. Izdvojeni features su distribution (histograms) of directions of gradients (oriented gradients) of the image. Gradients are typically large round edges and corners and allow us to detect those regions. Instead of considering the pixel intensities, they count the occurrences of gradient vectors to represent the light direction to localize image segments.

1. Knowledge-based metoda zavisi od skupa pravila zasnovanih na ljudskom znanju. Na primer, lice obično sadrži dva oka, nos i usta koji se nalaze na određenom rastojanju i poziciji jedni u odnosu na druge. Problem kod ovog modela je kreirati adekvatan skup pravila. Ako su pravila suviše opšta, rezultovaće dobijanje velikog broja lažno pozitivnih lica. Slično, ako su pravila suviše detaljna, rezultovaće dobijanje velikog broja lažno negativnih lica. Nisu primenljivi za sve boje kože i zavise od osvetljenja koje može znatno uticati na nijansu kože osobe na slici.
2. Template matching metoda koristi predefinisane ili parametrizovane šablone lica za lociranje i detekciju lica, računajući korelaciju između šablona i ulazne slike. Npr. šablon može pokazati da se lice podeljeno na regione kao što su nos, usta, oči i kontura lica. Takođe, može biti sačinjen samo od ivica i onda se može koristiti metod za detekciju ivica-implementacija ovog pristupa je jednostavna, ali nedovoljno dobra za detekciju lica. Ovi metodi ne podržavaju varijacije u pozi, skaliranju i obliku.
3. Appearance-based metoda zavisi od skupa trening slika koje se koriste za nalaženje modela lica. Zasniva se na mašinskom učenju i statističkim metodama kako bi se odredile relevantne karakteristike lica. Ove metode su pokazale znatno bolje performanse u poređenju sa ostalim.

-Viola-Jones detektor lica imao je najviše uticaja na istraživanje u ovoj oblasti. Široko je korišćen u aplikacijama i omogućava detekciju lica u realnom vremenu sa visokom pouzdanošću. Zasniva se na treniranju modela da razazna šta je lice, a šta ne. Jednom istreniran, model se čuva u fajl tako da karakteristike sa novih slika mogu biti upoređene sa prethodno sačuvanim karakteristikama. Ako slika prođe kroz sve faze upoređivanja feature-a, lice je detektovano. Ovaj metod neće raditi ako je lice prekriveno nečim, ili nije adekvatno orijentisano.

This method unites several algorithms:

* 1. Eigenface-based algoritam koji efikasno predstavlja lica korišćenjem Principal Component Analysis (PCA). PCA se primenjuje nad skupom slika kako bi se smanjila dimanzija dataset-a, najbolje opisujući varijansu podataka. Lice može biti modelovano kao linearna kombinacija eigenfaces (set of eigenvectors). Prepoznavanje lica je zasnovano na upoređivanju koeficijenata linearne reprezentacije.
  2. Distribution-based algorithms like PCA and Fisher’s Discriminant define the subspace representing facial patterns. They usually have a trained classifier that identifies instances of the target pattern class from the background image patterns.
  3. Neural networks, such as GANs, are among the most recent and most powerful methods for detection problems, including face detection, emotion detection, and face recognition.
  4. Support Vector Machines are linear classifiers that maximise the margin between the decision hyperplane and the examples in the training set. Osuna et al. first applied this classifier to face detection
  5. Sparse Network of Winnows defines two linear units or target nodes: one for face patterns and the other for non-face patterns. It is less time consuming and efficient.
  6. Naive Bayes Classifiers compute the probability of a face to appear in the picture based on the frequency of occurrence of a series of the pattern over the training images. The classifier captured the joint statistics of local appearance and position of the faces
  7. Hidden Markov Model is a standard method for detection tasks. Its states would be the facial features, usually described as strips of pixels.
  8. Inductive learning uses such algorithms as Quinlan’s C4.5 or Mitchell’s FIND-S to detect faces starting with the most specific hypothesis and generalizing.
  9. Video Processing: Motion-based face detection. In video images, you can use movement as a guide. One specific face movement is blinking, so if the software can determine a regular blinking pattern, it determines the face. Various other motions indicate that the image may contain a face, such as flared nostrils, raised eyebrows, wrinkled foreheads, and opened mouths
  10. Information Theoretical Approach:- Markov Random Fields (MRF) can use for face pattern and correlated features. The Markov process maximises the discrimination between classes using Kullback-Leibler divergence. Therefore this method can be used in Face Detection

**Odabir detektora lica [5] [6]**

U ovom poglavlju biće dat pregled nekoliko najpoznatijih detektora koji se mogu koristiti za detekciju lica, kao što su Haar cascades i DNN detektor iz OpenCV biblioteke, HOG + Linear SVM i CNN iz dlib biblioteke i MTCNN (Multi-task Cascaded Convolutional Neural Network).

1. OpenCV and Haar cascades

Ovaj model je prvi put predstavljen 2001. godine od strane autora Paul Viola and Micheal Jones. Veoma je brz i izdvaja mnogo features sa slike, ali i zahteva veliki broj pozitivnih i negativnih slika za treniranje. Najbolje features su nakon toga selektovani pomoću Adaboost. Ovim se originalnih 160000+ features redukuje na 6000 features. Primena svih ovih features u klizećem prozoru bi zahtevala mnogo vremena, te je preložena kaskada klasifikatora, gde su features grupisani. Ako prozor otpadne u prvoj fazi, preostali features u kaskadi se ne procesiraju. Ako prođe, onda se testira sledeći feature i postupak se ponavlja. Ako prozor prođe sve features onda se klasifikuje kao region koji sadrži lice. [7]

Ovo je originalni detektor lica koji je isporučen uz OpenCV biblioteku, istovremeno i najpoznatiji detektor. Veoma je brz i pogodan za izvršavanje u realnom vremenu, nema velike hardverske zahteve i ima mali model (oko 400KB). Sa druge strane, podležan je lažno pozitivnim detekcijama, zahteva ručno podešavanje parametara i mnogo manje je pouzdan nego ostali navedeni detektori. Treba ga koristiti kada je brzina izvršavanja prioritet i spremni smo da žrtvujemo pouzdanost.

1. Dlib’s HOG + Linear SVM implementation

Dlib je biblioteka toolkit koja sadrži algoritme mašinskog učenja za rešavanje problema iz realnog sveta. Iako je napisana u jeziku C++, moguće je izvršavati i u jeziku Python. Ovaj detektor radi tako što features izdvojene preko Histogram of Oriented Gradients (HOG) prosleđuje SVM. In the HOG feature descriptor, the distribution of the directions of gradients is used as features. Prvi put predstavljen od strane Dalal and Triggs u njihovom radu iz 2005, Histograms of Oriented Gradients for Human Detection [8]. Slično kao Haar cascades, ovaj algoritam se zasniva na piramidama slika i klizećem prozoru (napomena). Pouzdaniji je od Haar cascades, ima stabilniju detekciju i dobro je dokumentovan. Nedostaci su što radi samo sa frontalnim licima jer HOG deskriptor ne toleriše promene u rotaciji ili uglu gledanja, zahteva instaliranje dlib biblioteke, nije toliko pouzdan kao detektori zasnovani na dubokom učenju i skup je u vidu zahteva za računanjem jer radi pyramid construction, sliding windows, and computing HOG features at every stop of the window.

Napomena: Piramida slika je multi-scale reprezentacija slike. Omogućava nam da pronađemo objekte na slici posmatrajući njene skalirane verzije. Na dnu piramide je slika u originalnoj veličini, a u narednim slojevima resized slike koje su opciono smoothed (obično korišćenjem Gaussian blurring). Slika se smanjuje dok ne dostigne određenu minimalnu veličinu. [9]

Klizeći prozor je pravougaoni region fiksne širine i visine koji klizi preko slike sa leva na desno i odozgo na dole. Za svaki od prozora obično se primenjuje klasifikator za određivanje da li je objekat od interesa sadržan u prozoru. [10]

1. Dlib’s CNN face detector (A Max-Margin (MMOD) CNN face detector [12]

Davis King, kreator dlib-a, trenirao je CNN face detector na osnovu svog rada [max-margin object detection](https://arxiv.org/abs/1502.00046) [11]. Metoda je veoma pouzdana, zahvaljujući samom dizajnu algoritma i kvalitetnom trening skupu. Model je mali (manji od 1MB), veoma dobro implementiran i dokumentovan. Sa druge strane, zahteva da dlib bude instalirana, zahteva konvertovanje prozora za oukviravanje lica ako se koristi OpenCV (napomena), nemoguće ga je koristiti u realnom vremenu bez GPU ubrzanja. Preporuka je da se koristi kada ne treba brinuti o performansama u realnom vremenu.

Napomena: U OpenCV-u, uokvirujući pravougaonik je predstavljen uređenom četvorkom ( početna x koordinata, početna y koordinata, širina, visina), dok je u dlib-u predstavljen objektom pravougaonika sa left, top, right, and bottom atributima.

1. MTCNN [13]

Predstavljen je 2016. godine u paper, “Joint Face Detection and Alignment Using Multi-task Cascaded Convolutional Networks.” Pored detekcije lica, detektuje i 5 karakterističnih tačaka lica. Koristi kaskadnu strukturu sa 3 nivoa CNN-a. Prvo, koristi potpunu konvolucionu mrežu da prikupi prozore-kandidate i njihove regresione vektore, koji su preklopljeni korišćenjem on-maximum suppression (NMS). Dalje, ovi kandidati se prisleđuju drugoj CNN koja eliminiše veliki broj lažno pozitivnih i kalibriše obuhvatajuće pravougaonike. U poslednjoj fazi obavlja se detekcija karakterističnih tačaka lica.

1. Detektor lica baziran na dubokom učenju iz OpenCV biblioteke

U današnje vreme duboko učenje je jedna od najpopularnijih i najbrže rastućih oblasti u računarskom vidu. Od verzije 3.1 OpenCV biblioteke postoji DNN modul, a verziji 3.3 je prebačen u glavni repozitorijum i znatno je poboljšan. Ovaj modul podržava veliki broj framework-a za duboko učenje, uključujući Caffe, TensorFlow i Torch/PyTorch. [14]

U pitanju je Caffe model zasnovan na Single Shot Detector (SSD) i ResNet mreži, što ga čini i brzim i pouzdanim. Može se izvršavati u realnom vremenu na modernim laptopovima i desktop računarima, model nije preveliki (oko 10MB) i deo je OpenCV biblioteke. Pouzdaniji je od Haar cascades i HOG + Linear SVM, ali manje pouzdan od dlib’s CNN MMOD detektora. Najbolji izbor je za većinu primena, ne zahteva dodatne biblioteke i uključen je u OpenCV. Jedan od najvećih nedostataka je što je manje pouzdan za detekciju lica sa tamnijom bojom kože, ali to se može rešiti treniranjem modela skupom slika koji sadrži slike ljudi različitih rasa. Postoji i kvantovana Tensorflow verzija. More specifically, the model used (opencv\_face\_detector\_uint8.pb) has been quantized (with the TensorFlow library) on 8-bit unsigned int to reduce the size of the training model (2.7 mo vs 10.7 mo for res10\_300x300\_ssd\_iter\_140000.caffemodel). U poređenju sa prethodno navedenim metodama, DNN je dao najbolje rezultate prilikom testiranja real time videa (članak). Ovaj model radi dobro kada je lice delimično prekriveno, kada postoje brzi pokreti glave, može detektovati bočna lica, i ima najbrži fps.

**Kod:**

Na početku, potrebno je instalirati OpenCV biblioteku komandom:

pip install opencv-python

Nakon toga, treba učitati mrežu i proslediti slojeve modela i težine. Težine za slojeve se mogu preuzeti sa linka: <https://github.com/opencv/opencv_3rdparty/raw/dnn_samples_face_detector_20170830/res10_300x300_ssd_iter_140000.caffemodel>, a slojevi modela (arhitektura mreže) sa linka: <https://github.com/opencv/opencv_extra/blob/4.x/testdata/dnn/opencv_face_detector.prototxt>.

Kvantovana verzija:

<https://github.com/opencv/opencv/blob/4.x/samples/dnn/face_detector/opencv_face_detector.pbtxt>

<https://github.com/opencv/opencv_3rdparty/raw/8033c2bc31b3256f0d461c919ecc01c2428ca03b/opencv_face_detector_uint8.pb>

Da bi se postigla najbolja pouzdanost modelu treba proslediti BGR slike dimenzija 300x300 nad kojima je primenjeno oduzimanje srednje vrednosti (104, 177, 123) za svaki od B, G i R kanala, respektivno. Ovo je odrađeno korišćenjem funkcije blobFromImage. Da bismo dobili korektne predikcije na osnovu duboke neuronske mreže, potrebno je da preprocesiramo podatke. Oduzimanje središnje vrednosti se koristi kako bi se umanjio efekat promene osvetljenja u slikama iz skupa slika (dataset). Pre nego što se krene sa treniranjem duboke neuronske mreže, izračunaju se prosečni intenziteti piksela nad celim trening skupom za svaki od kanala. Rezultat je uređena trojka. Pre propuštanja slike kroz mrežu, oduzmemo usrednjene vrednosti za svaki od kanala ulazne slike. Možemo imati i faktor skaliranja koji se dodaje pri normalizaciji.

Funkcija cv.dnn.blobFromImage(image[, scalefactor[, size[, mean[, swapRB[, crop[, ddepth]]]]]]) ->retval kreira 4-D blob na osnovu slike. Opciono menja dimenziju i crops image from center, oduzima srednje vrednosti, skalira vrenosti na osnovu faktora skaliranja, vrši zamenu plavog i crvenog kanala. [15]

1. Image – ulazna slika
2. Scalefactor- After we perform mean subtraction we can optionally scale our images by some factor
3. Size – veličina izlazne slike
4. Mean - scalar with mean values which are subtracted from channels. Values are intended to be in (mean-R, mean-G, mean-B) order if image has BGR ordering and swapRB is true
5. swapRB - OpenCV assumes images are in BGR channel order; however, the `mean` value assumes we are using RGB order. To resolve this discrepancy we can swap the R and B channels in image by setting this value to `True`.
6. crop – flag which indicates wheather image will be cropped after resize or not
7. ddepth – depth of output blob, Choose CV\_32F or CV\_8U

Funkcija vraća blob koji predstavlja ulaznu sliku nakon oduzimanje središnjih vrednosti, normalizacije i zamene kanala. [16]

Rezultat prosleđivanja bloba mreži je 4-D niz koji sadrži informaciju o pouzdanosti i koordinatama lica skaliranih na opseg [0, 1], te ćemo množenjem tih brojeva sa originalnom širinom i visinom slike dobiti predikciju za originalnu sliku. The 3rd dimension iterates over the detected faces. (i is the iterator over the number of faces)

The fourth dimension contains information about the bounding box and score for each face. For example, detections[0,0,0,2] gives the confidence score for the first face, and detections[0,0,0,3:6] give the bounding box

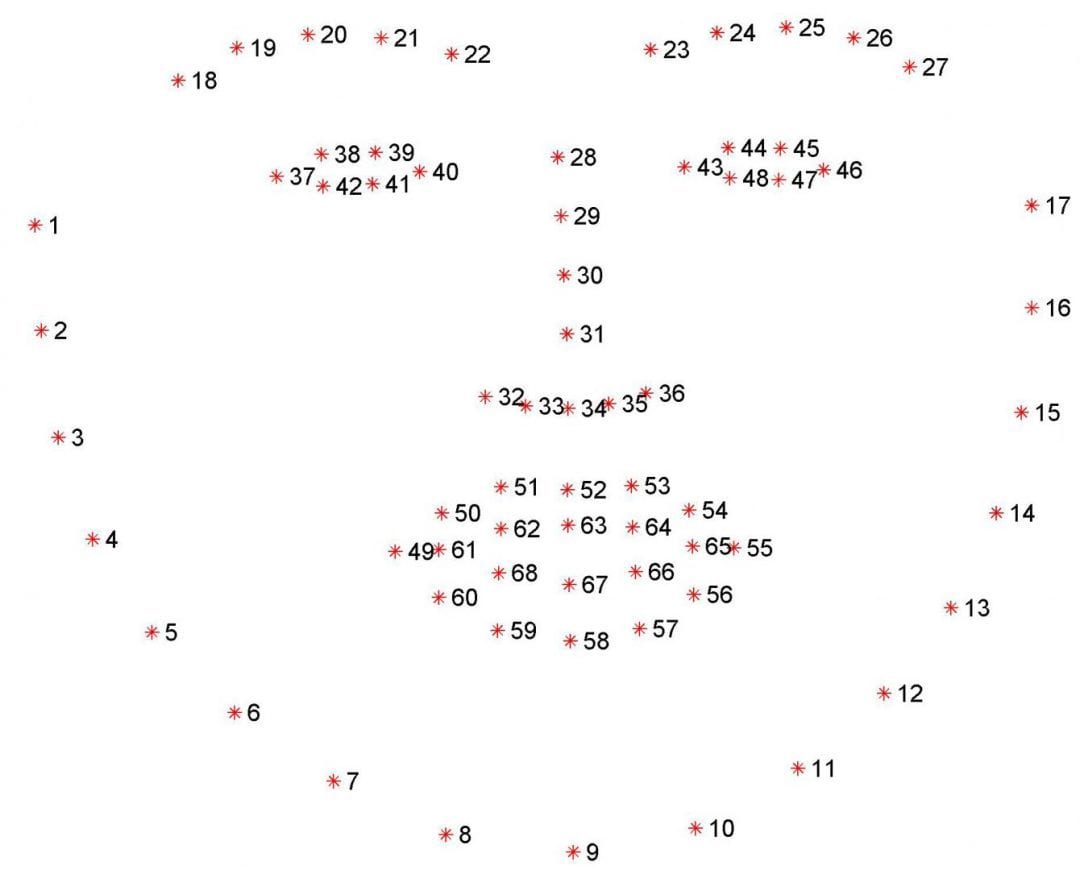
DETEKCIJA KARAKTERISTIČNIH TAČAKA LICA [26]

http://dlib.net/files/shape\_predictor\_68\_face\_landmarks.dat.bz2"

Detekcija karakterističnih oznaka lica je podskup problema predikcije oblika. Na osnovu ulazne slike (na kojoj je region od interesa), prediktor oblika locira ključne tačke za taj oblik. Oznake lica se koriste za lokalizaciju i istaknutih delova lica, poput očiju, obrva, nosa, usta i vilice. Uspešno se primenjuju za poravnanje lica, određivanje poze glave, detekciju treptaja itd. Mi ćemo za detekciju karakterističnih oznaka lica koristiti dlib, OpenCV, and Python. Detekcija se obavlja u 2 koraka: Pronalaženje lica na slici, a nakon toga pronalaženje karakteristika lica u regionu od interesa. Prvi korak se može obaviti na više načina. Kao što je opisano u prethodnom poglavlju, mi ćemo za detekciju lica koristiti OpenCV DNN model. Detektor lica nam vraća region od interesa predstavljen obuhvatajućim pravougaonikom.

Detektor oznaka lica iz biblioteke dlib

Detektor uključen u dlib je implementacija One Millisecond Face Alignment with an Ensemble of Regression Trees rada autora Kazemi and Sullivan (2014).

Metoda kreće sa korišćenjem trening skupa obeleženih oznaka lica na slici. Ove slike su ručno označene, specificiranjem (x, y) koordinata regiona koji okružuju svaku od struktura lica. Imajući trening podatke, an ensemble of regresionih stabala je treniran da ustanovi pozicije oznaka lica direkno na osnovu intenziteta piksela (bez izdvajanja features). Ovaj detektor može detektovati oznake lica u realnom vremu sa veoma dobrim rezultatom. Detektuju se koordinate 68 tačaka koje se mapiraju na strukturu lica. Indeksi svih 68 tačaka prikazani su na slici:

Ova anotacija je deo iBUG 300-W dataset na kojem je dlib detektor treniran. Dlib detektor oznaka lica je najpoznatiji zbog svoje brzine i pouzdanosti dlib biblioteke. Dlib nudi i detektor koji pronalazi 5 karakteritičnih tačaka lica i brži je od 68-tačaka varijante. Preporučuje se za korišćenje ako su nam samo potrebne likacije nosa i očiju.

Jedan od najpoznatijih novijih detektira je iz MediaPipe biblioteke koji je sposoban da detektuje 3D mrežu lica. Ukoliko želimo da koristimo samo OpenCV biblioteku, i ona nudi ugrađeni detektor karakterističnih tačaka lica.

PORAVNANJE LICA

There are many forms of face alignment.

Some methods try to impose a (pre-defined) 3D model and then apply a transform to the input image such that the landmarks on the input face match the landmarks on the 3D model.

Other, more simplistic methods (like the one discussed in this blog post), rely only on the facial landmarks themselves (in particular, the eye regions) to obtain a normalized rotation, translation, and scale representation of the face.

The reason we perform this normalization is due to the fact that many facial recognition algorithms, including Eigenfaces, LBPs for face recognition, Fisherfaces, and deep learning/metric methods can all benefit from applying facial alignment before trying to identify the face.

Thus, face alignment can be seen as a form of “data normalization”. Just as you may normalize a set of feature vectors via zero centering or scaling to unit norm prior to training a machine learning model, it’s very common to align the faces in your dataset before training a face recognizer.

By performing this process, you’ll enjoy higher accuracy from your face recognition models.

The purpose of this blog post is to demonstrate how to align a face using OpenCV, Python, and facial landmarks.

Given a set of facial landmarks (the input coordinates) our goal is to warp and transform the image to an output coordinate space.

In this output coordinate space, all faces across an entire dataset should:

Be centered in the image.

Be rotated that such the eyes lie on a horizontal line (i.e., the face is rotated such that the eyes lie along the same y-coordinates).

Be scaled such that the size of the faces are approximately identical.

To accomplish this, we’ll first implement a dedicated Python class to align faces using an affine transformation. I’ve already implemented this FaceAligner class in imutils.

Note: Affine transformations are used for rotating, scaling, translating, etc. We can pack all three of the above requirements into a single cv2.warpAffine call; the trick is creating the rotation matrix, M .

We’ll then create an example driver Python script to accept an input image, detect faces, and align them.

Finally, we’ll review the results from our face alignment with OpenCV process.

desiredLeftEye : An optional (x, y) tuple with the default shown, specifying the desired output left eye position. For this variable, it is common to see percentages within the range of 20-40%. These percentages control how much of the face is visible after alignment. The exact percentages used will vary on an application-to-application basis. With 20% you’ll basically be getting a “zoomed in” view of the face, whereas with larger values the face will appear more “zoomed out.”

Next, let’s will compute the center of each eye as well as the angle between the eye centroids.

This angle serves as the key component for aligning our image.

On Lines 34 and 35 we compute the centroid, also known as the center of mass, of each eye by averaging all (x, y) points of each eye, respectively.

Given the eye centers, we can compute differences in (x, y)-coordinates and take the arc-tangent to obtain angle of rotation between eyes.

This angle will allow us to correct for rotation.

To determine the angle, we start by computing the delta in the y-direction, dY . This is done by finding the difference between the rightEyeCenter and the leftEyeCenter on Line 38.

Similarly, we compute dX , the delta in the x-direction on Line 39.

Next, on Line 40, we compute the angle of the face rotation. We use NumPy’s arctan2 function with arguments dY and dX , followed by converting to degrees while subtracting 180 to obtain the angle.

In the following code block we compute the desired right eye coordinate (as a function of the left eye placement) as well as calculating the scale of the new resulting image.

On Line 44, we calculate the desired right eye based upon the desired left eye x-coordinate. We subtract self.desiredLeftEye[0] from 1.0 because the desiredRightEyeX value should be equidistant from the right edge of the image as the corresponding left eye x-coordinate is from its left edge.

We can then determine the scale of the face by taking the ratio of the distance between the eyes in the current image to the distance between eyes in the desired image

First, we compute the Euclidean distance ratio, dist , on Line 50.

Next, on Line 51, using the difference between the right and left eye x-values we compute the desired distance, desiredDist . We update the desiredDist by multiplying it by the desiredFaceWidth on Line 52. This essentially scales our eye distance based on the desired width.

Finally, our scale is computed by dividing desiredDist by our previously calculated dist .

Now that we have our rotation angle and scale , we will need to take a few steps before we compute the affine transformation. This includes finding the midpoint between the eyes as well as calculating the rotation matrix and updating its translation component On Lines 57 and 58, we compute eyesCenter , the midpoint between the left and right eyes. This will be used in our rotation matrix calculation. In essence, this midpoint is at the top of the nose and is the point at which we will rotate the face around:

To compute our rotation matrix, M , we utilize cv2.getRotationMatrix2D specifying eyesCenter , angle , and scale (Line 61). Each of these three values have been previously computed, so refer back to Line 40, Line 53, and Line 57 as needed.

A description of the parameters to cv2.getRotationMatrix2D follow:

eyesCenter : The midpoint between the eyes is the point at which we will rotate the face around.

angle : The angle we will rotate the face to to ensure the eyes lie along the same horizontal line.

scale : The percentage that we will scale up or down the image, ensuring that the image scales to the desired size.

Now we must update the translation component of the matrix so that the face is still in the image after the affine transform.

On Line 64, we take half of the desiredFaceWidth and store the value as tX , the translation in the x-direction.

To compute tY , the translation in the y-direction, we multiply the desiredFaceHeight by the desired left eye y-value, desiredLeftEye[1] .

Using tX and tY , we update the translation component of the matrix by subtracting each value from their corresponding eyes midpoint value, eyesCenter (Lines 66 and 67).

We can now apply our affine transformation to align the face:

For convenience we store the desiredFaceWidth and desiredFaceHeight into w and h respectively (Line 70).

Then we perform our last step on Lines 70 and 71 by making a call to cv2.warpAffine . This function call requires 3 parameters and 1 optional parameter:

image : The face image.

M : The translation, rotation, and scaling matrix.

(w, h) : The desired width and height of the output face.

flags : The interpolation algorithm to use for the warp, in this case INTER\_CUBIC . To read about the other possible flags and image transformations, please consult the OpenCV documentation.

Finally, we return the aligned face on Line 75.

1. **ZAKLJUČAK**

**LITERATURA**

1. <https://en.wikipedia.org/wiki/Face_detection>
2. <https://www.researchgate.net/publication/326667118_Face_Detection_Techniques_A_Review>
3. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.642.4548&rep=rep1&type=pdf>
4. <https://medium.com/sciforce/face-detection-explained-state-of-the-art-methods-and-best-tools-f730fca16294>
5. <https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c>
6. <https://pyimagesearch.com/2021/04/26/face-detection-tips-suggestions-and-best-practices/>
7. <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>
8. <https://hal.inria.fr/inria-00548512/document>
9. <https://pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/>
10. <https://pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/>
11. <https://arxiv.org/abs/1502.00046>
12. <https://towardsdatascience.com/cnn-based-face-detector-from-dlib-c3696195e01c>
13. https://arxiv.org/abs/1604.02878
14. <https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV>
15. <https://docs.opencv.org/3.4/d6/d0f/group__dnn.html#ga29f34df9376379a603acd8df581ac8d7>
16. <https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/>
17. https://en.wikipedia.org/wiki/Object\_detection
18. <https://pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/>
19. <https://arxiv.org/abs/1506.01497>
20. <https://arxiv.org/abs/1506.02640>
21. <https://arxiv.org/abs/1512.02325>
22. <https://arxiv.org/abs/1704.04861>
23. https://github.com/Zehaos/MobileNet
24. <https://github.com/chuanqi305/MobileNet-SSD>
25. <https://cocodataset.org/#home>
26. https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/