Primerjava metod za detekcijo puščic pri klasičnem pikadu

Matic Zgonc[1], **Borut Batagelj**[1]

[1]*Fakulteta za Računalništvo in informatiko, Univerza v Ljubljani*
E-pošta: mz4144@student.uni-lj.si, borut.batagelj@fri.uni-lj.si

Comparison of arrow detection methods for classic darts

In this paper we compared classic methods of computer vision with modern approaches based on machine learning and neural networks. Comparison was made on problem of detecting arrows at steel darts. First we developed system based on classic methods of computer vision and described complete process from calibration to score prediction. Next we developed systems based on machine learning. We used different architectures to find out which give us the best results. After we gathered all results we checked time complexity on all developed systems. At the end we compared systems by detection rate and performance.

1 Uvod

Pikado je šport, ki je pri nas in po svetu vedno bolj priljubljen. V zadnjih letih se opaža velika rast profesionalnih igralcev, gledalcev in denarnih nagrad. Na profesionalnih tekmovanjih se uporablja klasična, saj to predpisujejo standardi. Amaterski igralci, ki se ne udeležujejo tekmovanj pa pogosteje uporabijo elektronsko tarčo. Njena uspešnost zaznavanja je visoka, vendar pa zaradi zasnove prihaja do odbijanja puščic. Omogoča pa avtomatsko štetje rezultata in podpira različne vrste iger, medtem, ko moramo pri klasični vse štetje opraviti ročno. To nas je vodilo k razvoju sistema, za avtomatsko zaznavanje puščic in štetje točk.

Na trgu že obstajajo rešitve, ki rešujejo ta problem. Najbolj znana in razširjena v naši okolici je Scolia [3]. Deluje na principu uporabe več kamer iz različnih zornih kotov, ki služijo zaznavanju puščic. Njegova glavna slabost je ročna kalibracija sistema, ki jo je potrebno ponoviti ob vsaki spremembi stanja tarče oz. pozicije kamer.

Na podlagi obstoječih rešitev smo se odločili razviti sistem, ki bo s pomočjo računalniškega vida zaznaval puščice in beležil rezultate, brez posredovanja igralcev in s čim višjo uspešnostjo. Vzpostavitev sistema in njegova uporaba morata biti dovolj enostavna za povprečnega uporabnika.

V zadnjem obdobju je računalniški vid z uporabo umetne inteligence naredil velik napredek. Zato smo se odločili med razvojem primerjati dva pristopa zaznavanja puščic. Z uporabno klasičnih metod računalniškega vida [12], ter s pomočjo strojnega učenja [10].

2 Klasični sistem

Izgradnjo sistema smo razdelili na 3 sklope, in sicer avtomatsko kalibracijo tarče, iskanje točke udarca puščice, ter pridobivanje rezultata posameznega meta.

2.1 Avtomatska kalibracija tarče

Pri kalibraciji tarče je cilj pridobitev transformacijske matrike. Ta nam vhodno sliko, na kateri je tarča v obliki elipse, pretvori v pravilno okroglo obliko. Uporabimo jo tudi pri pridobivanju točke udarca in rezultata meta.

Transformacijsko matriko pridobimo iz 4 točk, ki se na igralni površini tarče nahajajo 90° ena od druge. Pri pridobivanju transformacijskih točk, smo si pomagali z barvnim prostorom HSV (hue, saturation, value), ter morfološkimi operacijami.

Prvi korak je iskanje elipse, ki obkroža igralno površino. Pridobimo jo z uporabo dveh HSV mask, eno za zeleno barvo in drugo za rdečo, ki jih združimo. Tako pridobimo dva kolobarja z zelenimi in rdečimi polji, ki pa med seboj niso povezani. Z uporabo morfoloških transformacij dilacije in erozije pridobimo kolobar, ki je primeren za iskanje obrisov (angl. contours). Pridobljen obris že kaže podobo elipse, ki pa ni čisto pravilne oblike in vsebuje manjše nepravilnosti. Funkcija fitEllipse iz OpenCV paketa [1], po metodi najmanjših kvadratov izračuna elipso, ki se najbolj prilega na dan nabor 2D točk. Z izračunano elipso na koncu iz vhodne slike izločimo zunanje območje elipse. S tem odstranimo morebitne motnje, ki bi lahko oteževale nadaljnjo kalibracijo.

Naslednji korak je iskanje točk, ki razmejujejo sektorje na zunanjem krogu. Ponovno smo uporabili HSV maske, vendar tokrat ločeno. Tako pridobimo točke, ki razmejujejo zelene in rdeče sektorje. Za vhod v transformacijsko matriko izberemo štiri točke, ki so med seboj oddaljene 5 sektorjev. Pred izvedbo transformacije, izračunamo še homografsko matriko (angl. Homography) z uporabo funkcije findHomography, ki mapira točke med dvema ravninama.

Zadnji korak je izvedba transformacije z uporabo funkcije warpPerspective. Ta na vhod dobi homografsko matriko, vhodne točke, ter ciljne točke, ki predstavljajo točke na poravnani tarči. Rezultat transformacije je kalibrirana tarča.

2.2 Iskanje točke udarca

Iskanje puščice smo izvedli s principom absolutne razlike med dvema slikama. Prva je posneta pred metom, druga pa po tem, ko puščica prileti v tarčo. Če so svetlobni pogoji v obeh primerih podobni, nam po izračunu razlike, ostane na sliki le puščica brez okolice. Da lahko poiščemo točko udarca, sliko najprej binariziramo, ter nato poiščemo obris puščice. Zaradi oblike puščice, točka udarca predstavlja najnižje ležečo točko na njej (Slika 1). Dobljene koordinate preslikamo v kalibrirano ravnino, da lahko izračunamo rezultat meta.

2.3 Izračun točk

Tarča je razdeljena na 20 sektorjev, kar pomeni, da je vsak širok 18°. Z izračunanim kotom med preslikano točko udarca in središčem kalibrirane tarče, dobimo vrednost zadetih točk. Z uporabo kota lahko pridobimo le točke od 1 do 20, ne pa tudi vrednosti sredinskih krogov, ki sta vredna 25 in 50 točk, ter dvojnega in trojnega multiplikatorja. Ker so dimenzije tarč standardizirane, lahko za izračun teh vrednosti uporabimo razdaljo od točke udarca, do središča tarče. Za izračun končnih točk, enostavno pomnožimo vrednost zadetih točk z multiplikatorjem.

2.4 Prilagajanje sistema

V procesu razvoja smo izvedli različne prilagoditve, da bi izboljšali delovanje sistema in dosegali čim boljšo uspešnost zaznavanja. Primerjali smo različne osvetlitve tarče, različne kote namestitve kamere, uporabo kamer različnih kakovosti in namestitev kamere na različnih lokacijah. V sklopu posamezne prilagoditve, so bili pogoji za vsako testno množico identični.

Pri osvetlitvi tarče smo preizkušali uspešnost zaznavanja v naravni osvetlitvi prostora in z uporabo led trakov. Za ta namen smo izdelali leseno ohišje za tarčo, okoli katerega smo namestili osvetlitev. Uporabili smo dve konfiguraciji led trakov. V prvi smo uporabili trak srednje moči, naravne barve svetlobe, v drugi pa smo poleg dodali še močnejši trak hladne bele svetlobe. Ugotovili smo, da je za zaznavanje najboljša uporaba katerekoli konfiguracije led trakov, saj so bile vse puščice dobro vidne in pravilno zaznane, medtem, ko pri naravni osvetlitvi več kot 15% puščic sistem ni zaznal, zaradi senc in črnih sektorjev tarče.

Pri naslednji prilagoditvi smo iskali kot, pod katerim kamera najboljše zaznava puščice. Identične kamere smo namestili na isto os, ki je bila pravokotna na tarčo. Preizkusili smo tri kote, in sicer 40°, 52°, in 60°. Izkazalo se je, da v uporabljeni testni množici, sistem najboljše zaznava puščice pri kotu 40°. Z večanjem kota se uspešnost zaznavanja znižuje, zaradi povečane zakritosti vidnega polja.

Nazadnje smo preverili, če postavitev kamere na različnih lokacijah okoli tarče vpliva na uspešnost. Izbrali smo postavitev na levi, desno in zgornji strani tarče. Na vse lokacije smo postavili identične kamere, pod istim kotom glede na tarčo. Ugotovili smo, da lokacija postavitve kamere ne vpliva na uspešnost zaznavanja. Na vseh lokacijah se pojavlja problem prekrivanja, njihovo število je naključno, odvisno od pozicije puščic.

Edini problem, ki je ostal po vseh prilagoditvah je prekrivanje puščic. Enostavno in učinkovito rešitev za to težavo, smo dobili pri testiranju zadnje prilagoditve. Čeprav se je na vsaki izmed lokacij pojavljalo prekrivanje, smo ugotovili, da problemi niso bili prisotni na istih primerih. Izkazalo se je, da sta v vsaki situaciji, vsaj dve kameri vrnili točno vrednost. Ker je razmak med njimi približno 120°, obstaja zelo majhna možnost, da bi bila ena puščica zakrita iz dveh lokacij. V primeru, ko vsaj dve kameri vrneta isto vrednost, privzamemo, da je to pravilen rezultat. S tem pristopom smo na isti testni množici dosegli uspešnost 100%.

3 Detekcija s pomočjo globokih nevronskih mrež

Po izdelanem sistemu, ki temelji na klasičnih metodah računalniškega vida, smo se odločili izdelati sistem, ki temelji na modernem pristopu, in sicer z uporabo strojnega učenja in s pomočjo nevronskih mrež. Tako lahko tudi primerjamo uspešnost klasičnega pristopa z modernimi. Z enakim problemom so se ukvarjali že pisci članka DeepDarts [10], ki so za zaznavanje uporabili arhitekturo za zaznavanje objektov YOLOv4 [4] (angl. You Only Look Once). Na podlagi njihovih dobrih rezultatov, smo se tudi mi odločili da uporabimo to arhitekturo. Izbrali smo novejšo različico YOLOv5. Poleg metod zaznavanja objektov, smo se odločili primerjati še metode instančne in semantične segmentacije. Za instančno segmentacijo smo uporabili knjižnico PixelLib [11], pri semantični pa arhitekturo DeepLabv3 [5]. Moderne pristope smo v sistemu uporabili samo za zaznavanje puščice in pridobivanje točke udarca, preostanek sistema pa je ostal enak.

Za učenje vseh modelov v naslednjih poglavjih smo uporabili učno množico 300 slik, ki smo jih pridobili tekom izgradnje sistema. Vse so posnete pod enakim kotom glede na tarčo, razlikujejo se samo v osvetlitvi in pa namestitvi kamere (levo, desno, zgoraj). Modele segmentacije smo učili na lastnem sistemu, za učenje YOLO uteži smo uporabili oblačno storitev Google Colab.

3.1 Instančna segmentacija - PixelLib

PixelLib je knjižnica, ki omogoča enostavno uporabo instančne in semantične segmentacije objektov. Instačna segmentacija bazira na Mask R-CNN (Mask Region Convolutional Neural Network) arhitekturi nevronskih mrež [7]. Model je preučen na COCO množici fotografij z 80 različnimi kategorijami [9]. Poleg uporabe obstoječih modelov podpira tudi učenje modela na lastni množici fotografij.

Primerjali smo anotacijo celotne puščice, polovice puščice do trupa in samo konice puščice. Učne množice so se razlikovale le v obsegu anotacije. Obsegale so 300 primerov, od tega smo jih 225 uporabili za dejansko učenje, 75 pa kot validacijsko množico. Na učni množici smo opravili še naključne transformacije obračanja ter rahle zameglitve, da smo pridobili bolj raznovrstno množico. Število ponovitev učenja smo nastavili na 150, velikost

regija anotacije	uspešnost
celotna puščica	50%
polovica puščice	60%
konica puščice	78%

Tabela 1: Rezultati instančne segmentacije glede na regijo anotacije.

regija anotacije	uspešnost
konice puščice	67.5%

Tabela 2: Rezultati semantične segmentacije glede na regijo anotacije.

paketa (angl. batch size) pa na 2, saj smo bili omejeni s pomnilnikom grafične kartice.

Metoda testiranja je bila sledeča. S segmentacijo smo pridobili masko puščice, poiskali ekstremno točko, ki predstavlja konico puščice. Z obstoječimi funkcijami iz klasičnega sistema, smo nato pridobili rezultate meta puščice.

Najboljši rezultat (Tabela 1) smo dosegli pri množici anotiranih konic puščice, kjer smo v skoraj vseh primerih zaznali pravilne objekte (Slika 1). Težave so se pojavljale v sami natančnosti maske, glede na konico puščice. Zgornji del konice puščice, ki je širši, je maska lepo zajela. Slabše je bila zaznana konica puščice, saj jo velikokrat ni zaznalo do samega konca, kar vpliva na končni rezultat uspešnosti sistema.

3.2 Semantična segmentacija - DeepLabv3

Za preizkus semantične segmentacije smo uporabili model DeepLabv3 [5], z arhitekturo nevronske mreže Resnet101 [8]. Treniran je na PASCAL VOC množici fotografij [6], ki vsebuje 20 različnih razredov. Prednost modela je dobra natančnost in zmogljivost, ki ju dosega pri izvajanju segmentacije v realnem času.

Za treniranje modela smo pripravili binarne maske, na katerih je iskana regija označena z belo barvo, okolica pa s črno. Za učenje smo vzeli množico fotografij in anotacij, ki obsega samo spodnji del puščice, saj smo pri instačnem segmentiranju na njej dosegli najboljše rezultate. Število ponovitev učenja smo nastavili na 100, pri velikosti paketa 2.

Dosegli smo 67.5% uspešnost (Tabela 2), kar je slabše kot pri instančni segmentaciji. Razliko v uspešnosti lahko pripišemo predvsem različnemu delovanju segmentacij. Ker semantična segmentacija ne loči med posameznimi instancami objektov, so bile puščice, ki so se prekrivale med seboj, zaznane kot enotna maska. Zaradi tega smo imeli težave pri iskanju ekstremnih točk konic puščic, saj je bila natančnost zaznavanja v tem primeru zelo slaba ali pa točk sploh nismo mogli prepoznati. Slabša je bila tudi natančnost zaznave, saj smo pri puščicah na robu sektorjev velikokrat dobili napačen rezultat (Slika 1).

3.3 YOLOv5

YOLO [4] (angl. You Only Look Once) je algoritem, ki z uporabo konvolucijskih nevronskih mrež, omogoča

regija anotacije	uspešnost
konice puščic	95.1%
območje udarca	88.6%

Tabela 3: Rezultati YOLOv5 modela glede na regijo anotacije.

detektiranje in prepoznavanje objektov na sliki v realnem času. Izbrali smo različico v5, ki je v uporabi od leta 2020. Čeprav ni objavljene uradne dokumentacije ali znanstvenega članka, je arhitektura v splošnem podobna kot pri različici v4. Je enostopenjski algoritem, kar pomeni, da v enem prehodu skozi nevronsko mrežo pridobimo lokacijo mejnega pravokotnika kot tudi klasifikacijo objekta. Ostali znani algoritmi kot so RCNN, Mask RCNN in Fast RCNN so dvostopenjski. V prvem koraku poiščejo regijo, kjer bi se objekt lahko nahajal, v drugem koraku pa se opravi klasifikacija za posamezno regijo. Ti algoritmi so bolj natančni, vendar počasnejši, zato niso najbolj primerni za delovanje v realnem času. Zaradi združitve obeh korakov se močno poveča hitrost algoritma. Tako YOLOv5 tudi presega hitrost 45 slik na sekundo.

Uporabili smo učni množici, ki sta bili sestavljeni iz identičnih fotografij kot pri prejšnjih preizkusih. Uporabili smo množico z anotiranimi spodnjimi deli puščic, dodali pa smo še množico z anotacijo okolice točke udarca puščice v tarčo, kar so uporabili tudi pri projektu DeepDarts. Anotirane datoteke smo uvozili v spletno storitev roboflow [2], ki omogoča izvoz datotek v obliko, ki je primerna za YOLOv5 učenje. Učni množici lahko dodamo še naključne transformacije, da dobimo čim bolj raznoliko množico. V našem primeru smo dodali rotacijo levo in desno za 36° ter striženje v vertikalni in horizontalni smeri.

Rezultati (Tabela 3) so bili boljši od pričakovanih, saj smo se pri utežeh, treniranih na konicah puščic, približali uspešnosti, ki smo jo dosegli pri klasičnem sistemu. Pri množici z anotacijo območja udarca so se pojavljale napake pri zaznavi, saj je bila velikokrat kot rezultat zaznana kovinska žica, ki ločuje sektorje na tarči, ne pa sama točka udarca s konico puščice (Slika 1). Kot smo zasledili v literaturi, je lahko to posledica premajhnih območij anotacij, saj za YOLO algoritem v splošnem velja slabša zanesljivost na majhnih objektih. Napake pri utežeh, treniranih samo na konicah puščice, so bile na zakritih puščicah, kjer so bile skrite konice puščic. V ostalih primerih je bila napačna zaznava, oziroma ni bilo zaznave.

Zaradi dobrega doseženega rezultata smo preverili, kako bi se odrezal sistem z več kamerami, ki bi zmanjšal možnost prekrivanja puščic. Uporabili smo obstoječe množice, ki smo jih zajeli pri razvoju klasičnega sistema, pri testiranju različnih postavitev kamer. Pri vsakem metu je bil pravilen rezultat napovedan vsaj pri eni poziciji. Če vzamemo za pravilen rezultat primere, kjer vsaj 2 poziciji vrnejo isti rezultat, bi dosegli 98.4% uspešnost.

Slika 1: Primer izhodnih fotografij zaznave: klasični sistem, semantična segmentacija, instančna segmentacija, YOLO

3.4 Primerjava časovne zahtevnosti in povzetek rezultatov

Časovna zahtevnost je pri našem problemu zaznavanja puščic pomembna iz razloga, da mora biti vsak met zaznan pred naslednjim metom. Po pregledovanju posnetkov iz turnirskih obračunov smo izmerili, da trajanje med posameznima metoma traja v povprečju med 2-3 sekunde.

Najboljšo uspešnost z najhitrejšim zaznavanjem smo dosegli s klasičnim sistemom, ki je opremljen z dvema ali tremi kamerami. Ta dosega uspešnost blizu ali celo 100%. Podobno uspešnost, z malo daljšim časom zaznavanja, dobimo s kombinacijo YOLO sistema z več kamerami, pri kateri dosegamo uspešnost okoli 99%. Sledita oba sistema v kombinaciji z eno kamero in uspešnostjo 95-96%, ki je odvisna tudi od prekrivanja puščic. Sistema, pri katerih smo implementirali segmentiranje, nista primerna za uporabo. Sistem z instančno segmentacijo je sicer dovolj zmogljiv, vendar pa je natančnost zaznavanja prenizka. Pri sistemu s semantično segmentacijo sta tako zmogljivost kot tudi natančnost preslaba za dejansko uporabo.

Primerjali smo tudi rezultate projekta DeepDarts, ki uporablja podoben YOLOv4 algoritem, vendar z drugačnim potekom zaznavanja. Pri modelu, treniranem na večji množici D1 (15.000 posnetkov), so dosegli 94.7% uspešnost zaznavanja, kar je primerljivo z našim rezultatom, ki smo ga dosegli s sistemom YOLOv5. Pri drugem modelu, ki je treniran na D2 množici (1050 posnetkov), pa so dosegli 84.0% uspešnost. V članku navajajo, da je več kot polovica napak posledica prekritih puščic, kar se je izkazalo tudi pri vseh naših sistemih, kjer smo uporabljali samo eno kamero. Sledijo napake, ko je puščica na robu sektorja, in na koncu še napaka pri kalibraciji tarče. Obeh najpogostejših napak smo se pri našem razvoju znebili z uporabo več kamer, kar nakazuje, da je to pravilen pristop za razvoj zanesljivega sistema.

4 Zaključek

V članku smo primerjali dva pristopa pri reševanju problema zaznavanja puščic pri klasičnem pikadu. Najprej smo razvili sistem s klasičnimi metodami računalniškega vida, ki smo ga primerjali s pristopom, ki temelji na strojnem učenju in nevronskih mrežah. Najboljše rezultate smo dosegli z uporabo YOLO algoritma, ki je po uspešnosti primerljiv s sistemom, razvitim na klasičen. Je časovno zahtevnejši, a še vedno dovolj zmogljiv za reševanje našega problema. Tekom razvoja smo ugotovili, da uporaba več kamer občutno izboljša uspešnost v vseh sistemih, saj se možnost prekrivanja puščic skoraj izniči.

sistem	uspešnost	čas zaznave [s]
klasični	96%	0.37
klasični (več kamer)	100%	0.37
instančna seg.	78%	1.65
semantična seg.	67.5%	3.65
YOLOv5	95.1%	0.96
YOLOv5 (več kamer)	99%	0.96
DeepDarts D1	94.7%	/
DeepDarts D2	84%	/

Tabela 4: Povzetek končnih rezultatov uspešnosti pravilne zaznave meta in povprečnega časa.

Literatura

[1] OpenCV. Dosegljivo: https://opencv.org/about/, 2022. [Dostopano 1.6.2022].

[2] Roboflow. Dosegljivo: https://roboflow.com/, 2022. [Dostopano: 12. 7. 2022].

[3] Scolia. Dosegljivo: https://scoliadarts.com/, 2022. [Dostopano 31.8.2022].

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020.

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[6] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Chris K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, jan 2015.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[10] William J. McNally, Pascale Walters, Kanav Vats, Alexander Wong, and John McPhee. Deepdarts: Modeling keypoints as objects for automatic scorekeeping in darts using a single camera. *CoRR*, abs/2105.09880, 2021.

[11] Ayoola Olafenwa. Simplifying object segmentation with pixellib library. 2021.

[12] Matic Zgonc and Borut Batagelj. Detekcija puščic pri klasičnem pikadu. page 115–123. Univerza v Mariboru, Univerzitetna založba, 2023. Nasl. z nasl. zaslona.