Primjena računalnog vida u očitavanju pogotka na metama

Hodžić, Adnan

Sveučilište Jurja Dobrile u Puli

ADNAN HODŽIĆ

Primjena računalnog vida u očitavanju pogotka na metama

Završni rad

Pula, kolovoz, 2020.

Sveučilište Jurja Dobrile u Puli

ADNAN HODŽIĆ

Primjena računalnog vida u očitavanju pogotka na metama

Završni rad

JMBAG: 0303076468, redoviti student

Studijski smjer: Informatika

Github repository: https://github.com/adhodzic/TargetScanner

Predmet: Poslovni informacijski sustavi

Znanstveno područje: Tehničke znanosti

Znanstveno polje: Računarstvo

Znanstvena grana: Informacijski sustavi

Mentor: doc. dr. sc. Darko Etinger, doc. dr. sc. Nikola Tanković

Pula, kolovoz, 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Adnan Hodžić, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Adnan Hodžić

U Puli, kolovoza, 2020. godine

IZJAVA

o korištenju autorskog djela

Ja, Adnan Hodžić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **Primjena računalnog vida u očitavanju pogotka na metama** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 24.08.2020.

Potpis

Adnan Hodžić

Sadržaj

Uvod

Moblinu aplikaciju za prepoznavanje pogodatka na metama i računanje rezultata namjenjena je osobama koje se bave gađanjem iz zračne puše na 10m u stojećem položaju. Aplikacija omogućava brzo skeniranje bez dodatnih podešavanja u jednom kliku pronalazi sve potencijalne pogodke te za svaki pogodak izračunava rezultat. Metu koja aplikacija trenutno podržava je “ISSF 10m air rifle target”

Slika 1: ISSF 10m air rifle target

Slika 2: ISSF 4.5mm pellet

10 Ring	0.5 mm	(±0.1 mm)	5 Ring	25.5 mm	(±0.1 mm)
9 Ring	5.5 mm	(±0.1 mm)	4 Ring	30.5 mm	(±0.1 mm)
8 Ring	10.5 mm	(±0.1 mm)	3 Ring	35.5 mm	(±0.1 mm)
7 Ring	15.5 mm	(±0.1 mm)	2 Ring	40.5 mm	(±0.1 mm)
6 Ring	20.5 mm	(±0.1 mm)	1 Ring	45.5 mm	(±0.1 mm)

Tablica 1: Dimenzije ISSF 10m mete

Korištene tehnologije

Python(Jupyter)

Za razvoj programa koji manipulira slikom i pronalazi hitce korišten je Python. Programski jezik koji nije strogo tipiziran pojednostavio je razvoj i skratio vrijeme na učestalim pokretanjima programa radi konstantnog podešavanja različitih parametara. Jedan od glavnih razloga zašto je korišten Python sa bibliotekom OpenCV koja je zadužena za manipulacijom slike i grafike je taj što se OpenCV koristi apstraktnim tipom podatka „Mat“ koji je višedimenzionalna matrica pixela, a biblioteka koja na jako jednostavan način manipulira višedimenzionalnim poljima i matricama je NumPy. Program koji sam koristio za pisanje i kompajliranje python-a je Jupyter. Vrlo brz i jednostavan te osim toga omogućava upravljanjem datoteka i mapa.

Java(Android Studio)

Android aplikacija razvijana je u programu Android Studio koji koristi Javu kao programski jezik i XML kao skriptni. Postojale su alternative kao što su programski okviri Vue-native, React-native, ali zbog loše podrške OpenCV biblioteke to nije bila opcija. Najveći izazov bio je prebacivanje Python koda u Javu jer je Java za razliku od Pythona strogo tipizirani programski jezik, ali uzimajući u obzir vrijeme koje bi potrošio na direktan razvoj u Javi to nije bilo tako veliki problem.

Slika 3: Android Studio

Slika 41: Jupyter

Razvoj

Algoritam za detekciju krugova

Istražujući algoritme za detekciju krugova i kružnica kao rezultat sam dobio dva najčešće korištena, a to su „Circle Hough Transform“(CHT) i Watershed algoritam.

Odlučio sam se za CHT zbog jednostavnosti i mogućnosti prepoznavanja krugova s već određenim radiusom. Generalna ideja je bila koristiti crni krug koji kreće od 4. prstena mete kao polaznu točku za određivanje centra mete i radijusa hitca(u pikselima). Znajući omjer 4. prstena(Slika 1.) i hitca(Slika 2.) moguće ga je primijeniti na piksele i sa time smo dobili radius kruga hitca za CHT algoritam. Nakon izvršavanja CHT algoritma dobijemo konture(Slika 5.) od kojih moramo pronaći centar „mase“ i to uzeti kao naš potencijalni pogodak. Nakon dobivenih centara za svaku konturu izbacujemo one koje ne ulaze u polje hitca, a to se možemo vidjeti u slučaju kada se dva ili više hitaca preklapaju i čine kut. Na kraju nam je ostalo izračunati udaljenost hitca od centra mete uz pomoć Pitagorinog poučka.

Funkcija za očitavanje pogotka na meti

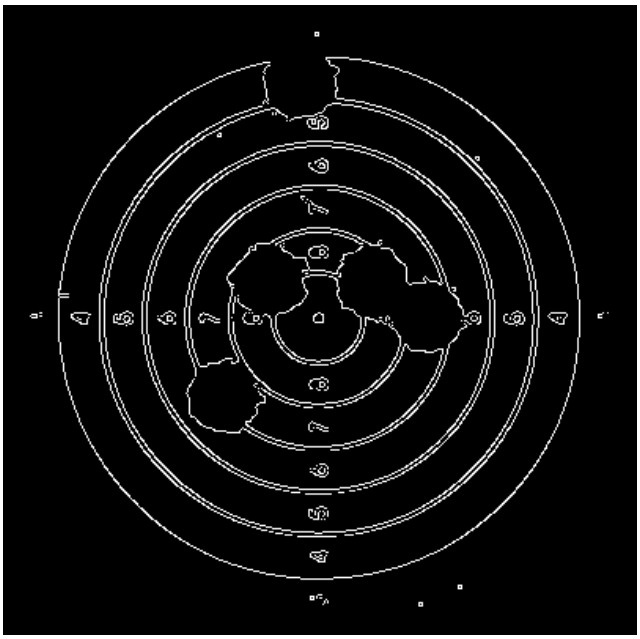
Funkcija koja obrađuje sliku i pronalazi hitce prima dva parametra. Matricu slike i listu rezultata. Kako funkcija koja prikazuje sliku na našem android uređaju ne može primiti Mat tip podatka moramo kreirati tip podatka Bitmap te na kraju funkcije našu Mat matricu konvertirati u Bitmap i takvu je vratiti.

U prvom dijelu koda cilj nam je pronaći konture 4. prstena kako bi mogli odrediti centar mete i kako bi dobili veličinu radijusa hitca u pikselima. Prvo sliku smanjujemo tako da zadržimo format slike ali smanjimo broj piksela. Ovo radimo prvenstveno da ubrzamo procesiranje slike. Nakon smanjivanja slike izvršavamo niz funkcija za manipulaciju slikom, a to su prebacivanje slike u boji u crnobijelu(funkcija cvtColor), filtriranje piksela

po zasićenosti(funkcija threshold), erodiranje(funkcija erode), proširivanje(funkcija dilate) i na kraju zamućivanje(funkcija medianBlur) sljedeći korak je pronaći rubove obrađene slike(funkcija Canny) (Slika 5.) te ih predati funkciji findCountours koja će na u varijablu contours spremiti sve konture rubova među kojima je i 4. prsten. Svi parametri funkcija

Isprobavani su ručno kako bi dobili što neovisniji algoritam o svjetlosti i rezoluciji slike.

```
Mat kernel = Mat.ones(5,5, CvType.CV_8U);
Mat grayC = new Mat(0,0,CvType.CV_8UC1);
Mat hierarchy = new Mat();
List<MatOfPoint> contours = new ArrayList<>();
double w,h;
w = image.cols();
h = image.rows();
double ratio = h/w;
Size size = new Size(1024, 1024*ratio);
Imgproc.resize(image, image, size);
Mat imageO = image.clone();
Mat edge = image.clone();
Imgproc.cvtColor(imageO, grayC, Imgproc.COLOR_BGR2GRAY);
Imgproc.threshold(grayC, grayC,110,255, Imgproc.THRESH_BINARY_INV);
Imgproc.erode(grayC, grayC, kernel);
Imgproc.dilate(grayC, grayC, kernel);
Imgproc.medianBlur(grayC, grayC, 15);
Imgproc.Canny(grayC, edge,50,255);
Imgproc.findContours(edge, contours, hierarchy, Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);
```



Slika 5: Kontura 4. prstena i svih unutar njega

Kada imamo polje svih kontura potrebno je pronaći pravokutnike koji opisuju te konture. Za svaki pravokutnik izdvajamo širinu i visinu te dodajemo pravokutnik u novo polje samo ako je veći od 200 piksela i manji od 520 tako smo se riješili šuma i nepotrebnih podataka.

Pravokutnike još sortiramo po razlici između širine i visine kao bi dobili pravokutnik što bliže kvadratu ili točno kvadrat. Taj pravokutnik će dalje poslužiti kao referenca za obrezivanje slike te određuje radijus hitca u omjeru 4. prstena i hitca (Slika 6.). Po izračunu omjer bi trebao biti 1:6.78, ali kako metak koji prođe kroz metu ostavi manju rupu s obzirom na svoju veličinu omjer na kraju iznosi cca. 1:7.3

```
MatOfPoint2f[] contoursPoly  = new MatOfPoint2f[contours.size()];
Rect[] boundRect = new Rect[contours.size()];
double cx = 0,cy = 0,cw = 0,ch = 0;
List<Rect> rectArr = new ArrayList<>();
for (int i = 0; i < contours.size(); i++) {
    contoursPoly[i] = new MatOfPoint2f();
```

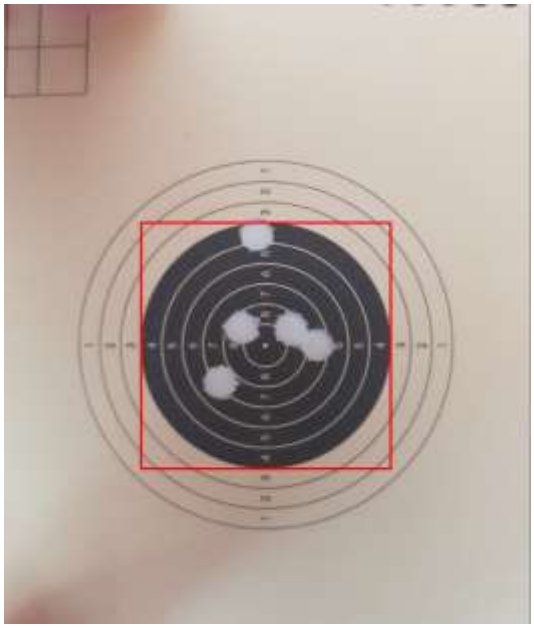
```
    Imgproc.approxPolyDP(new MatOfPoint2f(contours.get(i).toArray()),
contoursPoly[i], 3, true);
    boundRect[i] = Imgproc.boundingRect(new MatOfPoint(contoursPoly[i].toArray()));
    cx = boundRect[i].tl().x;
    cy = boundRect[i].tl().y;
    cw = boundRect[i].br().x - cx;
    ch = boundRect[i].br().y - cy;
    if(cw > 200 && cw < 520 && ch > 200 && ch < 520){
        rectArr.add(boundRect[i]);
    }
}

Collections.sort(rectArr, new Comparator<Rect>() {
    @Override
    public int compare(Rect o1, Rect o2) {
        double h1 = o1.br().y - o1.tl().y;
        double w1 = o1.br().x - o1.tl().x;
        double h2 = o2.br().y - o2.tl().y;
        double w2 = o2.br().x - o2.tl().x;
        int result = Double.compare(Math.abs(w1-h1), Math.abs(w2-h2));
        return result;
    }
} );

Rect resizeRect;
try{
    resizeRect = rectArr.get(0);
}catch (Exception e){
    return null;
}
```

U sljedećim djelu koda uzimamo kvadrat 4. prstena te dobivamo radijus R i sliku (Slika 7.) koja je izrezana s obzirom na kvadrat plus dodani pomak kako bi dobili sliku cijele mete, a ne samo crni dio. Dobivena slika je dalje spremna za obradu hitaca.

```
double wR = ((resizeRect.br().y - resizeRect.tl().y) + (resizeRect.br().x -
resizeRect.tl().x)) /2;
double R = wR/2/7.3;
double aW = resizeRect.br().x - resizeRect.tl().x;
double aH = resizeRect.br().y - resizeRect.tl().y;
int offsetX = (int)aW / 2;
int offsetY = (int)aH / 2;
Rect offSet = new Rect((int)resizeRect.tl().x-offsetX, (int)resizeRect.tl().y-
offsetY, (int)aW + (offsetY*2), (int)aH + (offsetY*2));
Mat cropped = image.submat(offSet);
```



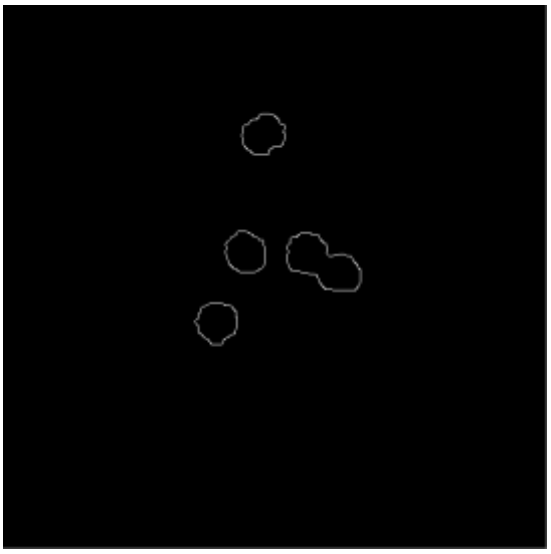
Slika 6: Pronađen 4. prsten

Slika 7. Izrezana slika

Prethodno izrezanu sliku prebacujemo iz RGB vrste u HSV kako bi primijenili filtriranje po bojama i dobili samo boju koja se nalazi ispod nasi pogodaka. Vrlo je bitno prilikom slikanja koristiti obični A4 papir kako se rezultati ne bi razlikovali. Boja koju mi koju mi filtriramo nalazi se u području plave te smo s obzirom na HSV spektar tako postavili parametre. Kako bi još smanjili šum podigli smo donju granicu zasićenosti boje S (Saturation) i jačinu V(Value). Nakon filtriranja boja i par funkcija za manipulaciju slikom dobili smo masku koja je crno-bijelog tipa i točno prikazuje konture hitaca (Slika 8.) ostaje još primijeniti Canny funkciju koja će nam pronaći rubove (Slika 9.)

```
Scalar lower = new Scalar(80, 25, 108);
Scalar upper = new Scalar(179, 255, 255);
Mat hierarchyH = new Mat();
List<MatOfPoint> contoursH = new ArrayList<>();
Mat hsv = new Mat();
Mat mask = new Mat();
Mat output = new Mat();
Mat gray = new Mat();
Mat erosion = new Mat();
Mat dil = new Mat();
Mat edgeH = new Mat();
Imgproc.cvtColor(cropped,hsv,Imgproc.COLOR_RGB2HSV);
Core.inRange(hsv, lower, upper, mask);
Mat kernel1 = Mat.ones(3,3, CvType.CV_8U);
Imgproc.cvtColor(cropped,cropped,Imgproc.COLOR_BGR2RGB);
Imgproc.erode(mask, erosion, kernel);
Imgproc.dilate(erosion, dil, kernel);
Imgproc.medianBlur(dil, dil, 11);
Imgproc.Canny(dil, edgeH, 50, 255);
Imgproc.findContours(edgeH, contoursH, hierarchyH, Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);
```

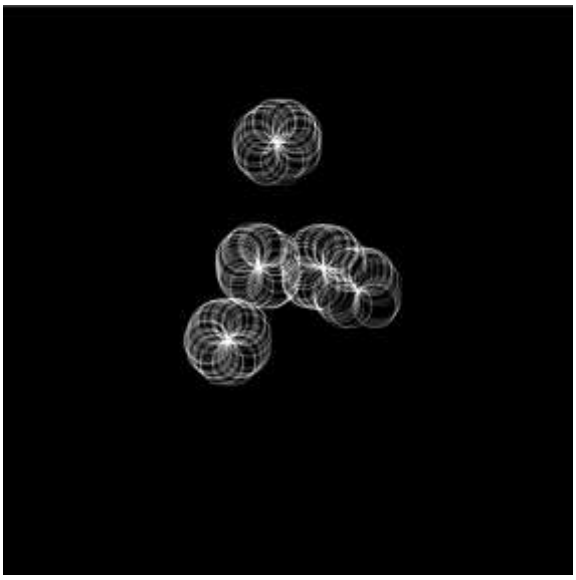
Slika 8: Izdvojeni hitci



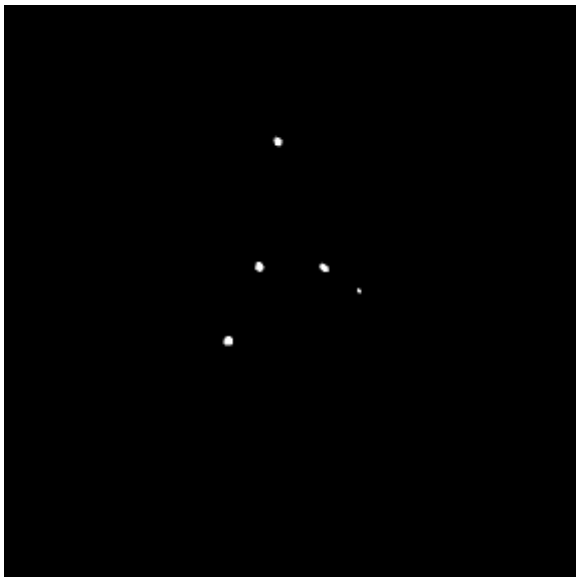
Slika 9: Rubovi hitaca

Dolazimo do dijela gdje zapravo primjenjujemo CHT algoritam kada prolazimo kroz svaki piksel i crtamo kružnicu one veličine radijusa R kojega smo prethodno dobili. Tako iscrtane kružnice (Slika 10.) provlačimo kroz filter (threshold) gdje izostavljamo ne dovoljno bijele piksele što znači da se u tim točkama nisu događala presijecanja kružnica i samim time nam ti pikseli ne sadrže veliku vrijednost. Primjenom erozije i proširivanja dobivamo konture koje nam imaju veliku vrijednost te neke od tih kontura će biti kandidati za naše pogotke (Slika 11.). Kako bi precizno odredili centar svake konture potrebno je koristiti metodu Moments koja će nam poslužiti za jednostavno određivanje centra mase svake konture te osim toga potrebno je provjeriti ako se taj centar nalazi barem unutar kontura hitaca. To provjeravamo tako da pogledamo vrijednost piksela maske (Slika 8.). Ako piksel od maske na poziciji centra hitca ima vrijednost 255 (bijeli piksel) sa sigurnošću možemo reći da se centar nalazi unutar konture hitaca i time ga uzeti u obzir.

```
Mat bg = Mat.zeros(cropped.size(), CvType.CV_8UC1);
Mat bg1 = Mat.zeros(cropped.size(), CvType.CV_8UC1);
MatOfPoint2f approxCurve = new MatOfPoint2f();
List<MatOfPoint> contoursC = new ArrayList<>();
List<Point> hits = new ArrayList<>();
for(MatOfPoint c:contoursH){
    for(Point p:c.toList()){
        Imgproc.circle(bg1, p, (int)R, new Scalar(55,55,55), 1,8,0);
        Core.addWeighted(bg,1,bg1,1,0,bg);
        bg1 = Mat.zeros(gray.size(), CvType.CV_8UC1);
    }
    Mat k = new Mat();
    Imgproc.threshold(bg, bg, 254, 255, Imgproc.THRESH_BINARY);
    Imgproc.dilate(bg, bg, kernel1);
    Imgproc.medianBlur(bg,bg,3);
    Imgproc.erode(bg, bg, kernel1);
    Imgproc.medianBlur(bg,bg,3);
    Imgproc.findContours(bg, contoursC, hierarchyH, Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);
    Point centeroid = new Point();
    for(MatOfPoint cC:contoursC){
        Moments M = Imgproc.moments(cC);
        if(M.get_m00()!=0){
            centeroid.x = M.get_m10() / M.get_m00();
            centeroid.y = M.get_m01() / M.get_m00();
            double[] colo = dil.get((int)centeroid.y, (int)centeroid.x);
            Double sth = colo[0];
            if(sth > 50){
                if(!hits.contains(new Point(centeroid.x, centeroid.y))){
                    hits.add(new Point(centeroid.x, centeroid.y));
                }
            }
        }
    }
    bg = Mat.zeros(gray.size(), CvType.CV_8UC1);
}
```



Slika 10: Iscrtan krug za svaki piksel konutre



Slika 11: Konure potencijalnih hitaca

U finalnom dijelu funkcije ocrtavamo kružnice na slici kako bi prikazali hitac i njegov broj za lakše raspoznavanje (Slika 12.). Te centar kao zelenu točku. Osim grafike računamo i udaljenost svakog centra hitca od sredine dobivajući rezultat za svaki hitac pojedinačno. Na kraju je ostalo matricu tipa Mat konvertirati u Bitmap tip i tu varijablu vratiti.

```
double segment = aW/2/6;
int index = 0;
for (Point h1:hits){
    index++;
    Imgproc.circle(cropped, h1, (int) R, color, 1, 8,0);
    Imgproc.putText(cropped, String.valueOf(index), new Point(h1.x - (R/3.5), h1.y
+ (R/3.5)),
    Imgproc.FONT_HERSHEY_COMPLEX, R/35,  color, 1, Imgproc.LINE_AA);
    double aa = h1.x - aW;
    double bb = h1.y - aW;
    double d = (aa*aa) + (bb*bb);
    d = Math.sqrt(d);
    scores.add((10-((d-R)/segment)));
}
try {
```

```
    bmp = Bitmap.createBitmap(cropped.cols(), cropped.rows(),
Bitmap.Config.ARGB_8888);
    Utils.matToBitmap(cropped, bmp);
}
catch (CvException e){
    return null;
}
    return bmp;
```

Slika 12: Numerirani hitci i centar

Grafičko sučelje aplikacije

Grafičko sučelje je dosta jednostavno zato što je ideja bila brza uporaba aplikacije sa što manje interakcije i podešavanja. Aplikacija se sastoji od dvije aktivnosti koje se međusobno pozivaju ovisno o događajima koje korisnik pokrene. Svako pozivanje aktivnosti prosljeđuje svoj identifikacijski broj i status izvršavanja (uspješno, neuspješno):

- Glavna aktivnost – Sadrži jedan gumb koji služi za pokretanje kamere za slikanje mete te ima zadatak spremanja slike na privremenu lokaciju i pokretanje aktivnosti za prikaz mete i rezultata. Također sadrži tekstualni okvir koji se prikazuje u slučaju da funkcija za pronalazak hitaca nije pronašla metu u tom slučaju se ispisuje odgovarajuća poruka.

- Aktivnost za prikaz mete i rezultata – Glavna zadaća je očitavanje slike koju je glavna aktivnost spremila te pokretanje gore opisane funkcije za obradu slike i pronalazak hitaca. Ako je funkcija uspješno detektirala metu vraća tip podatka Bitmap koji se prosljeđuje grafičkoj komponenti „ImageView“ kako bi je prikazali. Osim Bitmape funkcija sprema izračunate pogotke u javnu varijablu iz koje se kasnije rezultati prosljeđuju komponenti „TextView“ za prikaz.

Glavna aktivnost

Prva funkcija koja se izvodi prilikom pritiska na gumb „SCAN TARGET“(Slika 13.) izvršava provjeru dopuštenja aplikacije za korištenje kamere te memorije uređaja. U slučaju da aplikacija nema dopuštenja prikazuju se iskočni prozor koji traži od korisnika potvrdu. Ako su dopuštenja zadovoljena poziva se funkcija „openCamera()“.

```
btn.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v){
        if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
            if(checkSelfPermission(Manifest.permission.CAMERA) ==
            PackageManager.PERMISSION_DENIED ||
            checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE) ==
            PackageManager.PERMISSION_DENIED){
                String[] permission = {Manifest.permission.CAMERA,
                                       Manifest.permission.WRITE_EXTERNAL_STORAGE};
                requestPermissions(permission, PERMISSION_CODE);
            }else{
                  openCamera();
            }
        }else{
              openCamera();
        }
    }
});
```

Dio koda iz XML datoteka koju Android studio koristi kako bi opisao poziciju, veličinu, boju za određenu komponentu na sučelju jedne aktivnosti. U našem slučaju imao dvije komponente gumb i tekstualno polje.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="TextView"
    android:textColor="@color/colorAccent"
    android:textAlignment="center"
    android:visibility="invisible"
    android:gravity="center_horizontal" />
<Button
    android:id="@+id/button_camera"
    android:layout_width="133dp"
    android:layout_height="71dp"
    android:text="Scan target" />
```

Slika 13: Glavna aktivnost

Pozivanjem funkcije „openCamera()“ spremni smo za pokretanje kamere na uređaju te spremanje slike kao privremene datoteke. Ukoliko uspješno kreiramo datoteku u našoj memoriji onda slijedi otvaranje aplikacije kamere i na kraju spremanje slike na mjesto datoteke. Kada aplikacija kamere odradi svoj posao poziva se njezina metoda koja šalje rezultate među kojima se nalaze i greške ako ih je bilo.

```
private void openCamera() {
    Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if(cameraIntent.resolveActivity(getPackageManager())!=null){
        File imageFile = null;
        try {
            imageFile = getImageFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        if(imageFile!=null){
            imageUri = FileProvider.getUriForFile(this,
"com.example.android.fileprovider", imageFile);
            cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
            startActivityForResult(cameraIntent, IMAGE_CAPTURE_CODE);
        }
    }
}
```

Metoda prima tri parametra od kojih nas zanimaju samo dva, a to su „requestCode“ i „resultCode“. „RequestCode“ nam govori od koje aktivnosti nam dolaze rezultati kako bi znali kako dalje postupi s podatcima dok nam „resultCode“ sadrži podataka o uspješnosti obavljanja operacije npr. uspješno, greška, prekinuto. U našem slučaju se izvršava dio koda kojem „requestCode“ ima vrijednost varijable „IMAGE_CAPTURE_CODE“ za kameru i „SECOND_ACTIVITY“ za aktivnost koja nam prikazuje mete i rezultate. Osim provjere identiteta provjeravamo i status izvršavanja aktivnosti.

```
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == SECOND_ACTIVITY && resultCode == 2){
        msg.setVisibility(View.VISIBLE);
        msg.setText("Target not found. Try again!!!");
    }
    if (requestCode == IMAGE_CAPTURE_CODE && resultCode == RESULT_OK){
        BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
        bmpFactoryOptions.inPreferredConfig = Bitmap.Config.ARGB_8888;
        Bitmap bmp = null;
        try {
            bmp = MediaStore.Images.Media.getBitmap(
                    this.getContentResolver(),
```

```
                    imageUri);
        } catch (IOException e) {
            e.printStackTrace();
        }
        Matrix matrix = new Matrix();
        matrix.postRotate(90);
        Mat obj = new Mat();
        bmp = Bitmap.createBitmap(bmp, 0, 0, bmp.getWidth(), bmp.getHeight(),
matrix, true);
        Utils.bitmapToMat(bmp, obj);
        long addr = obj.getNativeObjAddr();
        Intent i = new Intent(MainActivity.this, ProcImage.class);
        i.putExtra("mat", addr);
        startActivityForResult(i, SECOND_ACTIVITY);
    }
}
```

Aktivnost za prikaz mete i rezultata

Kreiranjem aktivnosti koju je pozvala glavna aktivnost dohvaćamo adresu slike koja nam je proslijeđena te uz pomoć konstruktora za tip podatka Mat dohvaćamo sliku i spremamo je u varijablu „tempImg“. Sliku dalje kopiramo u „img“ varijablu i prosljeđujemo našoj funkciji „findHits“ koja će pronaći pogotke i rezultate te vratiti tako obrađenu sliku. Ako funkcija vrati null vraćamo se na glavnu aktivnost sa neuspješnim statusom. Dobivenu sliku postavljamo na našu GUI komponentu „ImageView“ . Kako nam se rezultati hitaca nalaze u polju tipa „double“ potrebno je podatke pretvoriti u tip „String“ i konkatenirati svaki rezultat kako bi sve rezultate prikazali kao jedan „Sting“ te ga postavili u komponentu za prikaz teksta (Slika 14.).

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_proc_image);
    ImageView imageView = (ImageView) findViewById(R.id.imageView);
    TextView ScoreTable = (TextView)findViewById(R.id.textView);
    long addr = getIntent().getLongExtra("mat", 0);
    Mat tempImg = new Mat(addr);
    Mat img = tempImg.clone();
    Bitmap img2 = findHits(img, scores);
    if (img2 == null){
        Intent resultIntent = new Intent();
        resultIntent.putExtra("Msg", "Target not found");
        resultIntent.putExtra("hits", scores);
        setResult(2, resultIntent);
        finish();
    }
    imageView.setImageBitmap(img2);
    String strScores = "SCORE TABLE\n\n";
    int count = 0;
    double total = 0;
    double totalRound = 0;
    for (Double s:scores) {
        total = total + s;
        totalRound = totalRound + Math.floor(s);
        count++;
        String formatS = String.format("%.2f", s);
        if(s %7==0) {
            strScores = strScores + count + ": " + formatS;
        }else{
            strScores = strScores + count + ": " + formatS + "\n";
        }
    }
    strScores = strScores + "\nTOTAL: " + totalRound + "(" +String.format("%.2f",
total) + ")";
    System.out.println(strScores);
    if (strScores != ""){
```

```
        ScoreTable.setVisibility(View.VISIBLE);
        ScoreTable.setText(strScores);
    }
}
```

Slika 14: Aktivnost obrađene slike i rezultata

Zaključak

Tema ovog završnog rada bila je primjena računalnog vida za detekcija pogodaka na metama. Veliku ulogu u razvoju imala je biblioteka OpenCV koja je sadržavala sve potrebne metode za manipulaciju slikom i matricama. Glavni dio funkcionalnosti odrađen je u Pythonu, ali zbog jednostavnosti izrade android aplikacije program je prebačen u programski jezik Java. Kako je ideja bila napraviti jednostavnu aplikaciju sa što manje interakcije, sučelje sadrži samo dvije aktivnosti te međuaktivnost koja otvara kameru. Aplikaciju je moguće još grafički urediti te je moguće dodati još sitnih funkcionalnosti kao što je spremanje obrađene slike i nekakve jednostavne evidencije slikanih meta. Postoji prostor za proširenje podrške drugih vrsta meta i kalibara.

Literatura

https://opencv.org/

https://developer.android.com/docs

https://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf

https://www.issf-sports.org/getfile.aspx?mod=docf&pane=1&inst=458&file=1.%20ISSF%20General%20Technical%20Rules.pdf

Slike i tablice

Tablica i Slika 1: https://www.issf-sports.org/getfile.aspx?mod=docf&pane=1&inst=458&file=1.%20ISSF%20General%20Technical%20Rules.pdf

Slika 2: https://www.google.com/url?sa=i&url=https%3A%2F%2Fcommons.wikimedia.org%2Fwiki%2FFile%3A4.5_mm_(.177_in)_match_air_gun_pellet.jpg&psig=AOvVaw0W6Mc7yfGSK-SsSB6tdNoa&ust=1598173477824000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCICHsqa6rusCFQAAAAAdAAAAABAE

Slika 3: https://www.google.com/url?sa=i&url=https%3A%2F%2Fcommons.wikimedia.org%2Fwiki%2FFile%3AAndroid_Studio_icon.svg&psig=AOvVaw3HcVpArfXHLlZy9_Z0hrXS&ust=1598113975394000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCPjm_NHcrOsCFQAAAAAdAAAAABAD

Slika 4: https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FProject_Jupyter&psig=AOvVaw0Bu2BB8M6dlwAg9QezhhqM&ust=1598173862623000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCNiAvN67rusCFQAAAAAdAAAAABAD