Obrada slike – Četvrti domaći zadatak

Automatizacija Android igara

Sadržaj

Uvod 3
Automatizacija 3
Zaključak 5
Literatura 6

Uvod

Automatizaciju Android igre ***Pong*** sam uradio uz pomoć ***Open-cv*** biblioteke i ***Python-a***, kao i ***Android debuging bridge-a (ADB)***, koji u ovom slučaju služi za emulaciju ekrana telefona kroz Python. Sama igra je relativno jednostavna, a pravougaonik kojim udaramo lopticu ćemo zvati veslo u daljem izlaganju.

Figure 1 - Pong

Automatizacija

Najpre uspostavljamo ***ADB*** konekciju sa Android uređajem na kome je igra pokrenuta.

```
adb = Client(host='127.0.0.1', port=5037)
devices = adb.devices()

if len(devices) == 0:
    print('no device attached')
    quit()
```

Koristimo ***mss Python*** biblioteku, uz čiju pomoć pravimo snimke ekrana, veoma brzo u beskonačnoj petlji (izuzev navigacionih dugmića na samom androidu, koji nam nisu potrebni).

```
while True:
    scr = sct.grab({
        'left': 2560,
        'top': 1470,
        'width': 910,
        'height': 540
    })
```

Figure 2 – Screenshot

Snimke ekrana konvertujemo u ***numpy array***, a zatim isti prikazujemo uz pomoć ***Open-cv*** biblioteke na ekranu radi debagovanja. Rezultat pri vrhu po sredini, moramo najpre da ***maskiramo***, jer može da bude ***0*** ili ***10***, što će, zbog sličnosti uticati na praćenje loptice koje sledi. ***Open-cv*** koristimo za crtanje pravougaonika na mestu rezultata koji je crne boje (*RGB(0,0,0)*).

cv2.rectangle(img, (420, 70), (480, 180), (0,0,0), -1)

Da bismo detektovali lopticu, najpre snimak ekrana konvertujemo u ***sivi prostor boja*** (grayscale colorspace). Ova igra je već crno bela, ali snimak ekrana dobijamo u RGB prostoru boja, što bi detekciju znatno otežalo, jer treba pronaći lopticu unutar više od 16 miliona boja. Konverzija u sivi prostor boja značajno smanjuje broj mogućih boja kroz koje pronalazimo lopticu.

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

Snimak ekrana u sivom prostoru zatim koristimo za pronalaženje i praćenje kretanja loptice na ekranu. Pojedinačne parametre za pronalaženje loptice sam ustanovio putem testiranja. Ovi parametri su najbolje detektovali lopticu tokom kretanja.

```
circles = cv2.HoughCircles(
    gray, cv2.HOUGH_GRADIENT,
    1, 20,
    param1=50, param2=30,
    minRadius=1, maxRadius=40)
```

Figure 3 - Detekcija loptice

Detekciju loptice lako možemo uočiti crtanjem crvene linije oko njene konture.

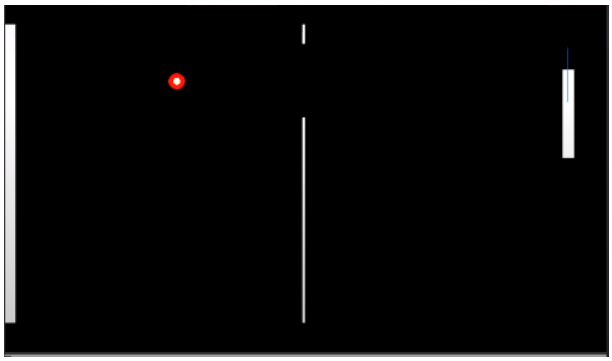

Figure 4 - Detekcija loptice, vizualizacija

Korišćenje ***Threading*** biblioteke istovremeno možemo da pratimo lopticu i ***pomeramo veslo*** ka njoj, simuliranjem ***touchscreen swipe-a*** preko ***ADB protokola*** (*device.shell(f'input touchscreen swipe {x1} {y1} {x2} {y2} 10')*). Tokom praćenja loptice konstantno pamtimo koordinate loptice, a zatim ***prema y koordinati*** pomeramo ***veslo ka loptici***.

Figure 5 - Pomeranje vesla

Zaključak

Open-cv je fenomenalna computer vision biblioteka, jer omogućava veoma laku manipulaciju digitalnim slikama, samim tim i video materijalom.

Uz dosta testiranja ova igra je uspešno automatizovana. Jedini problem tokom igranja se javlja kada loptica poludi i jako brzo menja pravac. Tada do izražaja dolaze limitcaije ADB protokola, koji nije pravljen sa brzinom na umu. Zakasnelo pomeranje vesla ka loptici je veoma moguće u tom slučaju. Ova limitacija ADB protokola se može zaobići, ali bismo morali da Root-ujemo Android uređaj kako bismo ostvarili neograničeno brzu komunikaciju.

Literatura

- https://developer.android.com/studio/command-line/adb
- https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/
- https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html
- OpenCV 4 with Python Blueprints - Dr. Menua Gevorgyan