Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Vecsei Gábor

ÉRINTÉSMENTES INTERAKTÍV KIJELZŐ

Sebészeknek

KONZULENS

Kundra László

BUDAPEST, 2017

Tartalomjegyzék

Összefoglaló

A célom ezzel a projektel az volt, hogy egy olyan kijelzőt készítsek, amit használni lehet úgy, hogy nem érünk hozzá és nem vagyunk kapcsolatban vele fizikailag, tehát távolról irányítható. Miért pont sebészek? Azért mert miközben egy műtét zajlik, több leletet/röntgenfelvételt meg kell nézni és jó lenne, ha egy könnyen kezelhető kijelzőn ezt kéz mozdulatokkal lehetne operálni.

Ebben a projektben egyaránt megmutatkozik a gépi tanulás és gépi látás, mivel bizonyos szekvenciákat be kell kategorizálni egy gyorsulásmérő alapján, illetve a kijelzőnek „látnia” is kell, hogy pontosan mit mutatunk a kezünkkel. Így általunk definmiált kézjelekkel és mozdulatokkal lehet kezelni majd a kijelzőt.

1 Bevezetés

Szeretném a bemutatni először is a projektet, hogy miért is álltam neki és, hogy miért tartottam fontosnak egy ilyen termék előállítását.

1.1 Motiváció

Elsődleges motivációm az az volt, hogy minél többet tanuljak a *Machine-Learning (Gépi Tanulás)* és *Computer-Vision (Gépi Látás)* terén. Sokszor az volt a gond, hogy a feladatok nagyon elméleti jellegűek és szerettem volna valami olyat készíteni ami ezt az elméleti hátteret ötvözi azzal, hogy egy valós problémára ad megoldást, mivel ezek a problémák sokszor sokkal bonyolultabban kivitelezhetőek és így mind a két részről nagyon sokat tanulhatok, nem kell egy bizonyos szintjére koncentrálnom (mint például az adat megtisztítása), hanem ez magába foglal mindent.

Másik motivációm az volt, hogy az egézségügyben dolgozók munkáját szeretném megkönnyíteni egy ilyen termékkel, így jobban lehet koncentrálni a fontos dolgokra illetve ez könnyítané a minden napokban előforduló problémákat amiket az „analóg rendszer” felfed. A fitnessz karkötők és webkamerák egyáltalán nem nagy befektetés, így minden különlegesebb felszerelés nélkül üzembe lehetne ilyen kijelzőket helyezni.

1.2 A rendszer elemeinek összefoglalása

A rendszernek több eleme van mivel több technológiát vegyítve lehet csak olyan alkalmazást és hardwaret készíteni ami megfelel az előírásoknak és a kívánt eredménynek.

Az volt az alap ötlet, hogy a fitnessz, sport karkötők, okos órák elterjedésével egy ilyen eszköznek a gyorsulásmérőjét használva információt kapunk arról, hogy az eszköz hogy is mozog ami ugye a kezünkön van, tehát tudni fogjuk a kezünk mozgását.

Ebből az adatból kézmozdulatokat fel lehet ismerni mert mindegyik kézmozdulat más *(x, y, z)* szekvenciákat eredményez.

Természetesen ennyi információ nem elég magában ahhoz, hogy egy ilyen kijelzőt teljesen irányítani tudjunk. Még kell tudnunk azt is, hogy mutatunk e valami kézjelet. Ezzel a módszerrel elég kevesebb kézmozdulat szekvenciát felismerni mivel ugyan az a mozdulat tud mást jelenteni, akkor ha más kézjelet mutatunk közbe. Erre egy példa, hogy ha mutató ujjunk ki van nyújtva és így elhúzzuk a kezünket oldalra akkor az lehet annak a jele, hogy lapozunk, viszont ha ökölbe szorítjuk és úgy húzzuk oldalra az lehet annaka jele, hogy valamilyen elemet „megfogtunk” és mozgatunk a képernyőn.

Ezekkel az elképzelésekkel kezdtem el a projektet.

1.3 Felhasznált eszközök

- Windows 10 operációs rendszer
- Python 3.6.1 – nyelv a fejlesztéshez
- Keras - Deep Learning Framework [3]
- OpenCV 3.2 – Computer Vision library
- PyCharm – Python IDE
- Github verziókövetésre
- Anaconda – Python környezet előállításához
- Jupyter Notebook – data exploration

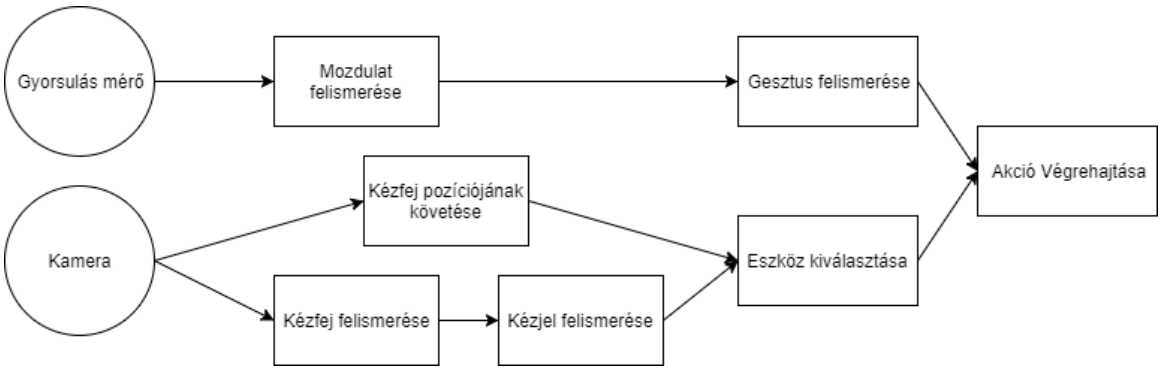
Hardware:

- Webkamera 720p
- Nexus 5X Androidos készülés

2 Rendszer

Ebben a fejezetben arról szeretnék írni, hogy hogy is épül fel egy ilyen rendszer, hogy terveztem meg. A szerencse, hogy minden technológia rendelkezésre áll arra, hogy egy ilyen feladatot elvégezzünk, csak sok adatra van szükség ahhoz, hogy a rendszerünk robosztus legyen.

2.1 Működés ábrája



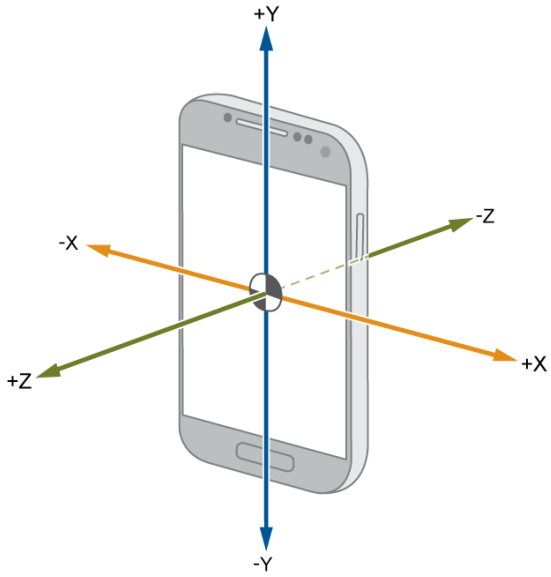
ábra 2-1. *A működés folyamatábrája*

A fenti 2.1 ábrán látható, hogy miként terveztem azt meg, hogy a rendszer felismerje azt, hogy mit is szeretnénk tenni, csupán webkamera és gyorsulásmérő alapján.

3 Mozdulat felismerése

3.1 Gyorsulásmérő

A gyorsulásmérő az ami segítségünkre van abban, hogy bizonyos mozdulatokat felismerjünk csak annyiból, ahogy a kezünk mozog. Természetesen a kezünkön rajta kell elnni egy olyan eszköznek ami ennek a műszernek az adatait szolgáltatja, mint például egy okos óra.



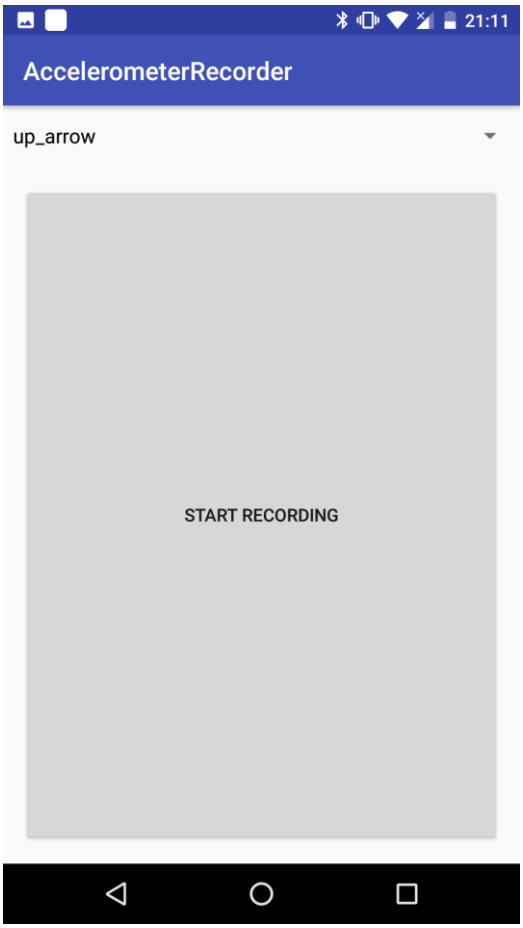
ábra 3-1. *https://www.mathworks.com/help/supportpkg/android/ref/accelerometer.html*

Ebben a projekteben egy Androidos mobil készüléket használtam, mivel ez állt rendelkezésemre.

3.1.1 Applikáció adat rögzítésre

Készítettem egy olyan applikációt mellyel könnyedén lehet „felvenni” és címkézni gesztusokat, mivel ilyen alkalmazást nem találtam. Az egész egy legördülő

menüből és egy nagy gombból áll (nehogy máshol nyomjuk meg), tehát nagyon egyszerű felülete van. A legördülő menüből a címkét lehet kiválasztani a gombbal pedig el lehet indítani a felvételt ami *10 Hz*-es mintavételezéssel rögzíti az *(x, y, z)* koordináták mentén történő gyorsulás változást.



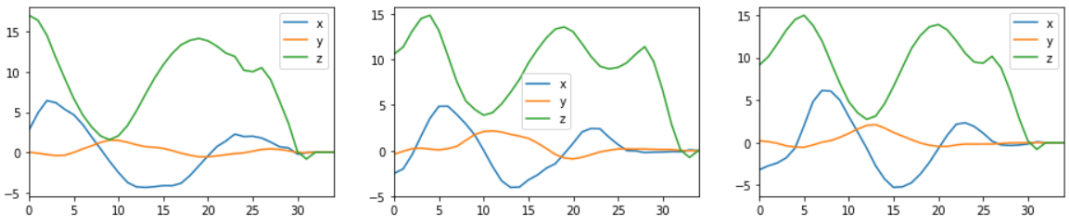
ábra 3-2. *Egyszerű adat rögzítő mobil applikáció*

A mobil a felvett adatot az SD kártyára mentette *.csv* formátumban.

3.1.2 Rögzített adat

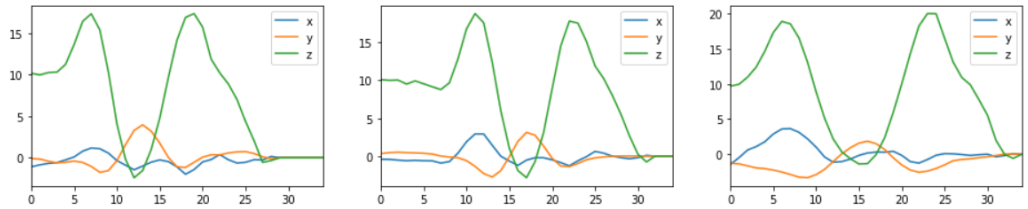
A rögzített adatokat megvizsgálva bebizonyosodott, hogy külömböző gesztusok ugyan olyan mintát mutatnak. Ezt *jupyter notebook* segítségével vizualizáltam. Az alsó ábrákon láthatunk erre példát.

```
In [7]: fig, axs = plt.subplots(1,3, figsize=(16,3))
        plt.grid()
        processed_dataframes["circle"][0].plot(ax=axs[0])
        processed_dataframes["circle"][1].plot(ax=axs[1])
        processed_dataframes["circle"][2].plot(ax=axs[2])
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x13770631860>
```



ábra 3-3. *Kör rajzolása a levegőben*

```
In [8]: fig, axs = plt.subplots(1,3, figsize=(16,3))
        plt.grid()
        processed_dataframes["up_arrow"][0].plot(ax=axs[0])
        processed_dataframes["up_arrow"][1].plot(ax=axs[1])
        processed_dataframes["up_arrow"][2].plot(ax=axs[2])
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x137707fa2b0>
```



ábra 3-4. *Felfelé mutató nyíl rajzolása a levegőben*

Az adatokat úgy készítettem elő, hogy összesen maximum 35 *„Timestemp”* lehetséges egy adott esetban ami azt jelenti, hogy 35 „időpont” az amit megtartunk. Ha kevesebb volt eredetileg mint 35, akkor 0-ákkal kiegészítjük a végén. Így szinte ugyan olyan ábrákat is kaptunk, minimális eltérésekkel.

Az, hogy így vizuálisan könnyen el tudjuk dönteni, hogy különbözőek azért jó, mert valószínüleg így egy Neurális Hálózatnak se lesz olyan nehéz a különbséget észrevenni.

3.1.2.1 Miért nem jó a gyűjtött adat?

Sajnos az idő rövidsége miatt, csak saját magamtól tudtam adatot gyűjteni ami nem jó, mivel így lehet, hogy a majd megépítésre kerülő neurális háló csak akkor tudja

jól felsimerni a mintákat ha azok tőlem származnak. Ahhoz, hogy ez robosztusabb legyen, nagyon sok adatot kellene gyűjtenem nagyon sok féle embertől.

3.1.3 Neurális Háló - GRU

A szekvencia felismeréséhez egy neurális hálózatot tanítottam be amit *Gated Recurrent Unit*, rövidítve *GRU*-nak neveznek. Sok keresés után azt találtam, hogy a *GRU* jobb mint a *Long Short Term Memory (LSTM)* mivel ez egy újabb megvalósítása igazából az *LSTM*-nek. Természetesen nem csak ez szól mellette, hanem gyorsabban lehet trainelni, és kevesebb adat kell ahhoz, hogy ugyan azt a pontosságot érjük el mint például egy LSTM hálóval [1].

Sajnos arra nem maradt időm és nem volt megfelelő hardware-em sem, és még adatom se, hogy a hálót betanítsam, így ez a jövőbeli énem feladata.

3.1.4 Kiértékelése

Megfelelően felvett és megtisztított adattal, egyszerű szekvenciákank (mint például a kör rajzolása) egyszerűen felismerhetőnek kellene lennie. A fentebb említett okok miatt, csak tippelni tudok, hogy mennyire működne.

4 Kézfej követése és felismerése

Egy kamerának a képén a kézfejet követnünk kell és fel kell ismernünk, mivel kézjeleket is szereznék detektálni egy másik modullal, így ez egy kritikus része az egésznek.

Ehhez a részhez készítettem egy külön Python modult amit nyílt forráskódú és szín alapján lehet vele tárgyakat követni és felismerni [2]. Githubon elérhető bárki számára.

Azért szín alapján detektálok, mert egyenlőre ez tűnt egy olyan egyszerű és robosztus megoldásnak amit olcsó nem különleges hardware-el kivitelezni lehet robosztus módon, úgy hogy nagy pontossággal fog működni. Ha maradunk a sebészetnél akkor ott úgy is kell gumi kesztűt hordani. Annak a színére (ha elüt a környezettől) be lehett állítani a modult, hogy trackelje a használó kezét. Ennek a modulnak a kimenete egy *(x, y)* koordináta sorozat (a hosszát ennek a listának meg lehet határozni és akkor csak a legfrissebb X darabot fogja megadni) ami a 2D-s térben a kéz pozícióját muttatja. Másik kimenete pedig egy olyan al-kép ami az eredetiből van kivágva de csak a kéz látható rajta.

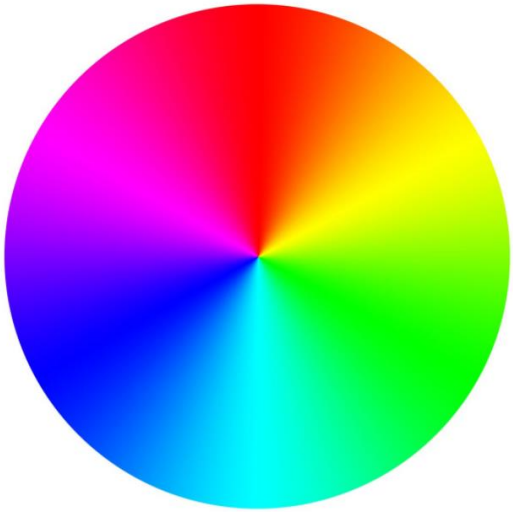
Ezzel a módszerrel könnyen lehet szegmentálni egy képről a kezet, minden nehézség nélkül. A hátránya ennek a módszernek, hogy a környezetben nem lehet oylan tárgy ami ugyan olyan színű és nagyobb annál a tárgynál amit detektálni szeretnénk.

ábra 4-1. *Sárga szín alapján gördeszka trackelése a modullal*

4.1 Módszer ismertetése

A detektálás mint említettem a színen alapszik. Egy kép amit a webkamerából kapunk RGB (Red, Green, Blue) csatornákból áll és a színkeverés alapján jönnek ki a színek a képen. Ha egy színt ki szeretnénk választani akkor külön meg kall adni a Piros, Zöld és Kék csatornák értékét.

Mindeközben a HSV (Hue, Saturation, Value) egy olyan reprezentálása a színeknek ahol ki tudom választani külön a színárnyalatot, azt, hogy mennyire fehér a választott szín és, hogy mennyire „könnyű”. Ezzel szín detektáláskor sokkal egyszerűbb dolgozni mivel így a piros több árnyalatát egyszerre le tudom fedni és nem csak 1 bizonyos árnyalatot mint RGB esetén.



ábra 4-2. *HSV színválasztó kör*

Ezt OpenCV segítségével valósítottam meg.

4.1.1 Lépései

1. Kamera képének betöltése a memóriába
2. BGR (Blue Green Red) színtérből HSV-be konvertálni
    a. Azért van itt BGR mert az OpenCV-nek ez a visszafelé irányított színér az alap beállítása. Szerencsére egyszerűen lehet bármilyen más színtérbe konvertálni.
3. Egy felső és egy alsó korlát meghatározása a kívánt színhez
4. Maszk készítése a korlátok alapján
    a. Ami a HSV képen beleesik a korlátok közé azt egy új képen (ugyan olyan méretű mint az eredeti kép) 1-essel jelöljük a többit pedig 0-ással. Így kapunk egy mátrixot aminek az elemei csak [0,1] egész számok. Amit az 1-es jelöl az potenciálisan az objektum amit detektálni szeretnénk a többi nem érdekes az algoritmus számára.
5. Morfológiai műveletek végzése a maszkon

    a. Morfológiai zárás művelete
    b. Morfológiai nyitás művelete
6. Legnagyobb területű összefüggő 1-es csoport megkeresése. Ezt hívják kontúr keresésnek. Mivel ez a legnagyobb és ideális esetben nincs más ilyen színű objektum a képen így ez a keresett része a képnek.
7. Legnagyobb kontúrból egy határoló doboz (Bounding Box) számítása
    a. Ebben a dobozban, (ami a képnek egy téglalap alakú része) megtalálható a keresett objektum
8. Ennek a határoló doboznak a középpontját eltárolni a pozíció követéséhez
9. Előről kezdődhet egy új képpel amit a webkamerából nyerünk

4.2 Color-Tracker Modul

Az elkészített objektum követő modullal minimális kóddal lehet könnyedén detektálni a kezet szín szerint.

```
import cv2
import color_tracker


def tracking_callback():
    frame = tracker.get_frame()
    debug_frame = tracker.get_debug_image()
    object_center = tracker.get_last_object_center()

    cv2.imshow("original frame", frame)
    cv2.imshow("debug frame", debug_frame)
    key = cv2.waitKey(1)
    if key == 27:
        tracker.stop_tracking()
    print("Object center: {0}".format(object_center))


webcam = color_tracker.WebCamera(video_src=0)
webcam.start_camera()

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))

tracker = color_tracker.ColorTracker(camera=webcam, max_nb_of_points=20, debug=True)
tracker.set_tracking_callback(tracking_callback=tracking_callback)
tracker.track(hsv_lower_value=(0, 100, 100),
              hsv_upper_value=(10, 255, 255),
              min_contour_area=1000,
              kernel=kernel)

webcam.release_camera()
```

ábra 4-3. *Minimális kód objektum követéséhez*

5 Kézjel felismerése

Sajnos időm nem marad foglalkozni ezzel a résszel így csak a gondolataimat tuzdom megosztani. Ez is a jövőbeli énem feladata, hogy elkészítse a rendszernek ezt a modulját.

5.1 Gondolatok a megvalósításról

A kezet már tudjuk detektálni, az előző fejezetben (4) említett algoritmus szerint. Most pedig fel kellene ismeri, azt hogy a kéz mutat e valamilyen jelet amit fel kellene ismernünk (például ökölbe szorított kéz).

Ezt a feladatot úgy oldanám meg, hogy megint készítenék egy nagyon egyszerű applikációt ami az előző modullal követi a kezem és felvételeket csinál csak a kézfejemből miközben én (és még mások is) mutatnának jeleket. Ezután számítógépen címkézném a jelekez és így kapnék egy tanító adathalmazt.

Miután adatok van, egy Konvolúciós Neurális Hálót építenék, de nem 0-ról, hanem az *InceptionV3* modellt [4] tanítanám újra, vagyis annak az utolsó két *Inception blokkját*. Így nem kellene előről egy nagy hálót betanítani, ami nagyon sok idő és nem is rendelkezek a kellő hardware-el.

Egy ilyen hálóval minden egyes képen meg lehetne mondani, hogy milyen kézjelet látunk.

Irodalomjegyzék

[1] Rahul Dey and Fathi M. Salem: Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks

[2] https://github.com/gaborvecsei/Color-Tracker

[3] https://github.com/fchollet/keras

[4] Rethinking the Inception Architecture for Computer Vision, 2015

Függelék

A projektehez tartozó forráskód megtalálható Github-on:

- https://github.com/gaborvecsei/Surgeon-Board
    - A *surgeon_board* mappán belül megtalálható a 3 modul amiből összeáll a kész projekt
    - Az Androidos applikáció kódja is megtalálható a hozzá tartozó modulon belül
- https://github.com/gaborvecsei/Color-Tracker