Dokumentáció a Gépi Látási beadandóra

Vörös Bence László

C3EC1Z

Tartalomjegyzék

Bevezetés 2

A megoldáshoz szükséges elméleti háttér 2

A megvalósítás terve és kivitelezése 4

Tesztelés 7

Felhasználói leírás 9

Irodalomjegyzék 11

Bevezetés

Az általam választott és ki dolgozott téma a felhasználók által bevitt fényképen szereplő rendszám felismerése, detektálása és le olvasása volt. A feldolgozott probléma az élet számos pontján applikálható, legyen szó akár egy autópálya használati jogosultságot ellenőrző rendszerről, adott autó vagy az azt használó személy mozgását (térfigyelő kamerák segítségével) meghatározó, követő rendszerről, a traffipaxok képei alapján a gyorshajtó autók detektálásáról, de akár csak egy egyszerű beléptető rendszerről mely a parkolni kívánó járművek rendszáma alapján meghatározhatja a fizetendő összeget vagy a használatra való jogosultságot.

Mivel a képek forrása, minősége és a rendszámtáblához viszonyított nézőpontja és dőlése ismeretlen, így ezt a szoftvernek is képesnek kell lennie ezt lekezelni és a helyes megoldást megtalálni különböző bevitelek esetén.

A megoldáshoz szükséges elméleti háttér

A feladat megoldásához több elméleti megoldást is használtam melyek megtalálhatók az openCV Python könyvtárában beépítve.

A **medián zajszűrő** működése során a pixelek által felvett érték meghatározásához annak adott méretű környezetében lévő pontok értékeinek mediánját használja fel. Ehhez szükség van egy értékre mely a környezet szélességét és magasságát definiálja, az értéknek mindenképp egy 1-nél nagyobbnak és páratlan számnak kell lennie.

A kép széléin szereplő pixelek egyértelmű okokból problémások (a szomszédsági négyzet elhelyezéséhez nincs elegendő hely) ennek áthidalása végett a szűrő openCV implementációja a „Border Replicate” módszert használja. Itt abban az esetben mikor a szomszédsági négyzet indexei a képek határain kívülre esnek úgy a határon kívüli elemek értéke megegyezik a kép határain helyezkedő, hozzályuk legközelebb eső elemével (pl.: „aaa|abcdefgh|hhh”).

Az érték meghatározásához használt képlet:

$$J(x,y) = med\{I(i,j)|I(i,j) \in S(x,y)\}$$

A **Canny** egy olyan algoritmus amelynek célja az adott képen automatikusan történő, élek detektálása. Jelen esetben az élt mint a kép intenzitásában hirtelen, nagy mértékű megváltozás definiáljuk.

A Canny lefutása 4 fő lépésből áll.

1. **Zaj redukció**: egy 5x5-ös kernelt használó Gauss filter segítségével a képen esetleg felmerülő zajt távolítja el.
2. **Az intenzitás gradiensek megtalálása**: egy Sobel kernel használatával mind horizontális, mind vertikális irányban egy filterezést végzünk. Az így megkapott deriváltakkal majd meghatározzuk az él gradienseket. Ennek képlete:

$$\text{Él } Gradiens\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Szög\ (\theta) = tan^{-1}(\frac{G_y}{G_x})$$

3. **A nem maximum értékek elnyomása**: minden egyes pixelre egy olyan tesztet végzünk amely eldönti, hogy a pixel által felvett érték a környezetében egy lokális maximum értéket képvisel-e. Ha igen akkor megtartja az értékét, ha nem akkor az értékét 0-ra csökkentik.
4. **Hiszterézis küszöbérték szűrés**: ennek elvégzéséhez 2 küszöbértékre van szükség, egy minimum és egy maximum határra. Amennyiben egy pixel értéke eléri a maximum értéket úgy biztosak lehetünk benne, hogy egy élről van szó, ha viszont nem éri el még a minimum értéket sem úgy szinte biztosak lehetünk hogy nem egy él eleme. Azokban az esetekben, ahol a pixel értéke a két határérték közé esik, ott a döntést a környezete alapján kerül meghozatalra. Ha az összekötésben áll egy olyan elemmel ami már biztosan egy él (azaz, az értéke > a felső korlátnál vagy már korábban ő is a 2 határ közé esett de a döntés él elemeként határozta meg) akkor ezt is él elemének tekintjük, ha környezetében nincs „valós” él vagy ahhoz nem csatlakozik a vizsgált elem úgy azt nem tekintjük élnek és az értéke szintén nullázásra kerül.

Ennek végeztével egy olyan képet kapunk melyek csak az úgynevezett „erős” éleket tartalmazza.

A **Hough transzformáció** célja különböző alakzatok, mint például egy egyenes vonal, megtalálása és pozicionálása egy képen. Jelen projekt során egyenesek

detektálására és azok az origóhoz viszonyított dőlésszögének megtalálására használtam, ezért az algoritmus működését is ilyen alakzatokon mutatom be.

Minden egyenes leírható parametrikus módon a következő képlettel $p = x\cos\theta + y\sin\theta$, ahol p az origótól mért távolság és θ pedig az adott egyenesre állított merőleges és az origó által bezárt szög. Ezért az egyeneseket tárolhatjuk egy olyan tömbben mely dimenziói [p, θ] lehetséges értékei, ezt Hough térnek is szokták nevezni. A detektálás elvégzéséhez szükségünk lesz egy gyűjtő tömbre melynek sorai a p értékeket és oszlopai a θ értékeket reprezentáljak, így minden egyes cella egy lehetséges egyenest ír le. A gyűjtő tömb méreteit szabadon választhatjuk meg a kívánt pontosság alapján.

A funkció futása során, a kapott bináris képen, végig halad az összes nem 0 ponton majd az összes lehetséges θ értékre egyenest állítva megkeresi az ezekhez tartozó p értékeket, és a Hough térben a [p, θ] helyen tárol értéket 1-el növeli. Egy pont vizsgálata során tehát több helyen is inkrementálásra kerül sor a Hough térben, melyek elhelyezkedése általában szinuszos alakot vesz fel.

Miután ez az összes pontra megtörtént a lefutás, a gyűjtő tömbben bizonyos cellákban az értékek kimagaslóan nagyobbak lesznek mint a többiben, az ilyen [p, θ] párok által leírt egyenesek ezért nagy valószínűséggel az eredeti képen is megtalálhatók.

A „Probabilistic Hough transzformáció” ezt a lefutást annyiban egészíti ki, hogy a lefutás gyorsasága és erőforrás igény csökkentése miatt nem az összes ponton végzi el a pontok Hough térbe történő mappelését, csak azok egy szűk halmazán melyeket valamely véletlenszerű módszer alapján választ ki.

A megvalósítás terve és kivitelezése

A rendszámtábla leolvasásához be kell látnunk, hogy az első és egyben legfontosabb (mindamellett legnehezebb) lépésnek, annak elhelyezkedésének pontos meghatározása kell, hogy legyen. Az egész képen történő karakter keresés hozhat sikeres eredményeket ám nem nehéz olyan példát találni, ahol már az autón vagy annak környezetén olyan alakzatok, karakterek találhatók melyek az ilyen módú leolvasást megnehezítik vagy lehetetlenné teszik.

Éppen ezért a rendszám helyének meghatározását kell célba vennünk először. Itt számos megközelítés létezik, található olyan mely a rendszám hátterének színére

(fehér) hagyatkozik ám ez semmiképp sem konstans, kifejezetten a zöld rendszámmal rendelkező elektromos hajtású autók elterjedése miatt, de akár csak a tábla tisztasága is befolyásolhatja a detektálást. Mivel az autó és a kamera térbeli relációja ismeretlen, nem is beszélve a rendszámtáblának az autón történő elhelyezkedéséről így erre sem hagyatkozhatunk. Az általam legoptimálisabbnak és a mások által legelfogadottabbnak tűnő megoldások az élek detektálására majd az azok által leírt kontúrok szögeinek méretére, számára és oldalaik hosszának arányára hagyatkoznak.

A program kivitelezéséhez én a Python nyelvet választottam, részben a fejlesztési folyamat gyorsasága és egyszerűsége miatt, de nem utolsó sorban a kiterjedt és széleskörű kód könyvtár potenciálja miatt. A képek beolvasására, tárolására, elő és utó feldolgozására az openCV könyvtár csomagot választottam mely a hobbi gépi látásos körökben épp annyira elterjedt és sikeresen, mint az iparban. A rendszámtábla karaktereinek leolvasására pedig a Tesseract OCR nevű szoftver mellett döntöttem, melynek a Python nyelvű implementációja a pytesseract könyvtár.

Az algoritmus 5 főbb lépései:

- A kép beolvasása és szürkeárnyalatossá konvertálása
- Az élek és kontúrok detektálása
- Az ígéretesnek tűnő alakzatok egymás utáni feldolgozása amíg a helyes megoldást meg nem találtuk vagy az alakzatok el nem fogytak
- Az éppen tesztelt kontúr alapján az eredti kép egy szeletének kivágása és felkészítése leolvasásra
- Az előfeldolgozott képeken a karakterek felismerése

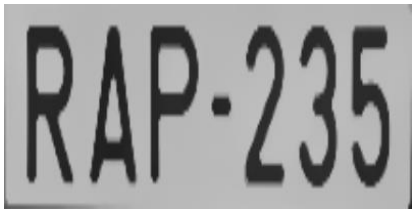
Ennek egy potenciális lefutását a következő képen mutatnám be mely egy optimális esetnek tekinthető. Első lépésként ezt át alakítjuk szürkeárnyalatossá.

1. ábra: a bemeneti és szürkeárnyalatos kép.

Majd végre hajtjuk az éldetektálást. Ez Canny módszerrel történik 75-ös alsó és 250-es felső határértékekkel. A kontúrok felismeréséhez az openCV beépített RETR_TREE és CHAIN_APPROX_SIMPLE konstansait használtam.

2. ábra: a Canny éldetektálás által adott eredmény.

Majd a detektált kontúrokat területük szerint csökkenő sorrendbe rendezzük és így haladunk rajtuk végig egészen addig amíg a helyes választ meg nem találtuk vagy a vizsgált alakzatok területe a le nem csökkent a legnagyobb terület 90%-a alá, ez a paraméter viszont szabadon módosítható, szimplán a program lefutásának gyorsításáért felel. Az éppen feldolgozott kontúr a képen pirossal látható, az őt bekerítő négyzet pedig kékkel került megrajzolásra. Mellette látható a kontúr alapján kivágott képmetszet mely a rendszámtáblát tartalmazza.



3. ábra: a detektált rendszámtábla és annak metszete.

Ezt követően futhat le a karakter felismerő algoritmus, a Tesseract. Ennek paraméterezése jelen esetben „--oem 1 --psm 11”, ahol az oem tag az LSTM neurális háló alapú szövegfelismerés kiválasztásáért felel, míg a psm-el a keresett szöveg szegmentálásának módját lehet leírni, jelen estben a 11-es egy

véletlenszerűnek tekinthető elhelyezkedést határoz meg és így minél több karaktert próbál megtalálni. Ezután a leolvasott szöveget az indításkor megadott paramétereknek megfélően lehetőség van szűrni és amennyiben a kapott szöveg egyezik az elvárttal úgy vége a lefutásnak, ám ha nem akkor a következő alakzat tesztje következik.

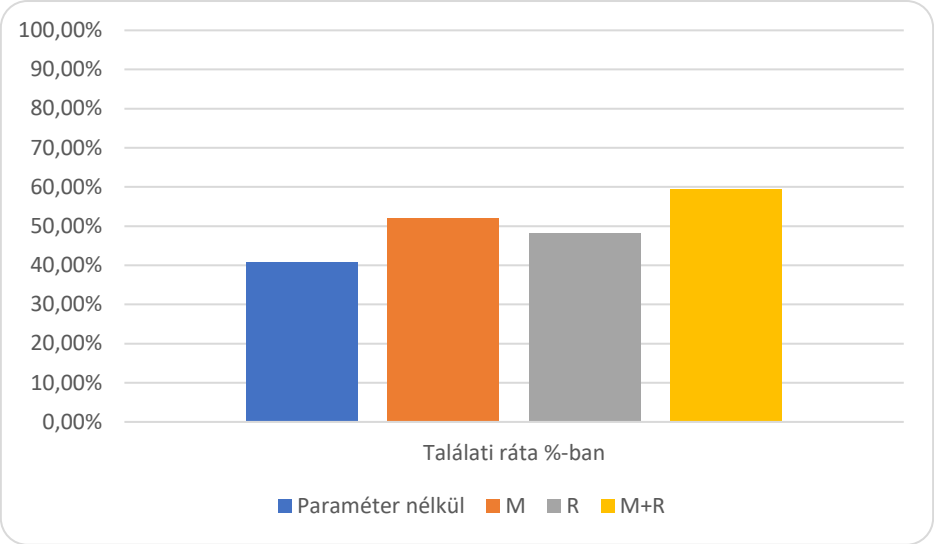
Itt kitérnék a fenti leírás során futólag említett behatároló négyzet szerepére. Annak érdekében, hogy a feldolgozott alakzatokat lehetőség legyen valamely módon automatikusan megszűrni úgy, hogy csak azokat teszteljük melyek méretarányai hasonlóak egy potenciális rendszámtábláéhoz bevezetésre került egy ratioCheck nevű funkció melynek feladata ennek megsaccolása a szélesség és magasság adatok alapján. Ennek implementálása azért is volt fontos mivel, a program könnyen vakvágányra futhatott, ha elsőként egy olyan nagy területű zárt alakzatot vizsgált mely egyáltalán nem, vagy csak részben tartalmazza a keresett rendszámot. Itt, ha a hibásan detektált terület mérete eléggé nagy akkor lehetséges, hogy később hiába tesztelnénk a helyesen detektált rendszámtábla kontúrját a program a területi eltérés mértéke miatt nem venné figyelembe. Ezért, ha a vizsgált kontúr megfelelt a ratioCheck elvárásainak és területe nem szembetűnően kisebb az eddigi maximum értéktől (ez alap esetben 70%-ra van beállítva ám ez módosítható), úgy a megengedett maximumot módosítjuk az adott alakzatéra.

Illetve a program egy másik „különlegessége”, hogy amennyiben végigfutottunk már az összes potenciális alakzaton és már kifogytunk az opciókból, mint egy utolsó próba a detektálási algoritmus újra indul más kép előfeldolgozási lépésekkel, melyek bizonyos esetekben a megoldás helyes meghatározását eredményezik ám nem applikálhatok minden kép esetén ezért csak, mint egy utolsó opció kerülnek próbára.

Tesztelés

A program működésének teszteléséhez egy 27 képből álló képsorozatot használtam, amelyek különböző autókat, különböző szögből, távolságból és viszonyok között ábrázolnak, ezeket az internen elérhető képek közű válogattam össze oly módon, hogy a lehető legjobban le teszteljem a szoftver határait. Ezek a képek a GitHub oldal „.\images\” mappájában találhatok.

A képsor tesztelése során a program paraméter megadása nélkül 11 rendszámot volt képes sikeresen megtalálni, ez a -M paraméterrel 14-re, -R paraméterrel 13-ra, -M és -R paraméterrel 27 képből 16-ot ismert fel sikeresen. Ezek az eredmények a következő grafikonon került összegzésre.



4. ábra: a különböző paraméterekkel történő futtatás eredményei.

A fenti eredményeknek számos oka van, melyek a különböző forrású képek közötti eltéréseknek, variációknak tudhatók fel. Ilyen volt például: a beviteli kép minősége, a kamera térbeli viszonya a rendszámtáblához képest, a változó fényviszonyok, ám bizonyos esetekben az autó környezete vagy típusa is problémát okozott a detektálás során.

Mivel a rendszámtábla detektálása zárt, téglalap alakú kontúrok észlelésére alapul ezért a következő képek feldolgozása során a rendszámtábla részleges takarása problémát okoz, de egy kifejezetten szögletes forma vagy világítás okozta fényviszonyok is könnyen tévútra vihetik a detektálást.

5. ábra: a nem detektálható, szélsőséges esetek.

A program teljesítményének további kivizsgálása érdekében készítettem egy saját képsort is melynek célja a kód határainak megtalálása a maximális betekintési szög közelítőleges megtalálása érdekében. Az itt felhasznált képek a „.\sajat\” mappában találhatók.

Egy számomra elérhető gépjármú rendszámtáblájáról készítettem képeket megközelítőleg 0°, ~45°, ~50° és ~60°-os szögekből. A teszt eredményei azt mutatták, hogy a program a 0 és 45 fokos szögből készült képeket még képes volt felismerni, ám az ennél nagyobb betekintési szögből készült képek már problémát okoztak, részben a kód hiányosságai miatt, részben a képek minősége miatt. Bizonyos módosításokkal a program képes lehetne ezen extrémnek tekinthető esetek feldolgozására is ám a téglalap kontúrjának torzulása miatt egy másik megoldás célra vezetőbb lehetne.

5. ábra: a program által legnagyobb feldolgozható betekintési szögből készült kép, illetve annak kivágott metszete

Felhasználói leírás

A program fútatásához mindenképpen szükségünk lesz a Python 3 telepítésére (fejlesztés során én 3.10-et használtam). Továbbá szükség lesz az openCV és pytesseract könyvtárak telepitésére. Illetve szükség van a Tesseract OCR telepítésére is, fontos, hogy a telepítés után a kiválasztót mappa elérési útvonalát be kell másolnunk a kódba futtatás előtt (kivéve, ha az alapból felajánlott elérési útvonalat válasszuk, ami alap esetben „C:\Program files\Tessarect OCR\”).

A program futtatásához nélkülözhetetlen a -i parancssori paraméter megadása mely után egy elérési útvonalat kell megadnunk. Ez lehet egy fájl vagy egy mappa is (ez egy speciális eset amire a később térek ki).

A program rendelkezik továbbá 6 opcionális paraméterrel is:

- -B: ez az az eset mikor egy mappát adhatunk meg az i paraméter számára. Ha a program ezt az argumentumot észleli, bekapcsol egy úgynevezett „batch” módot, ahol a mappában szereplő összes képfájlra elvégzi a detektálási algoritmus futtatását. Majd ennek végeztével az utolsó sorban ki írja a sikeres találatok arányát.

- -V és -v: ez két különálló paraméternek is tekinthető ám mivel a feladatuk szinte megegyezik ezért egyben kezelem őket. A céljuk a „verbose” kimenet bekapcsolása, az-az az adott lépések közötti képek megjelenítése a felhasználó számára. A különbség a két megoldás között abban rejlik, hogy míg a -v minden lépés után egyesével jeleníti meg a képeket, addig a -V pedig egyben teszi.
- -O: a „verbose” mód által megjelenített képeket el is menti (abba a mappába, ahonnan a program fútatásra került).
- -S: a „silent” mód bekapcsolására szolgál. Ez a gyakorlatban annyit tesz, hogy a program futása során semmilyen képet nem jelent meg és csak a szöveges kimenetet használja a felhasználó tájékoztatására.
- -h: a help üzenet megjelenítésére szolgál. Ez az egy eset, ahol a -i paraméter elhagyható. Továbbá amennyiben valamely másik paraméter hibásan lett megadva vagy a program más hibába ütközött úgy a program szinten ezt az üzenetet jeleníti meg.
- -M: megadásának segítségével a szoftver a képről leolvasott szövegre egy extra szűrési lépést is elvégez mely a kapott szövegben a magyar rendszám formátumnak megfelelő részletet keres. Ez hasznos lehet olyan esetekben, ahol a szövegleolvasás a rendszámon kívül más karaktereket is beolvasott (vagy valamely más objektumot detektált félre, mint karakter) vagy olyan esetekben, ahol a rendszámtáblán más karakter is szerepel (mint például az országot jelölő „H”). Fontos, hogy ez a szövegen nem módosít, karaktereket nem szúr be vagy töröl, csupán a kiolvasott karakterhalmaz egy részletét emeli ki ezzel javítva a detektálási arányon.
- -R: ezen opció megadásával, a program, amennyiben egy kontúr detektálása sikertelen volt, úgy kikalkulálja a rendszámtábla vízszinteshez viszonyított dőlési szögét és ha ez meghalad egy értéket elforgatja a kivágott kép szeletet és újra megkísérli a rendszám le olvasását ezzel javítva az eredményeket.

Irodalomjegyzék

— Ballard, D., 1979. Generalizing the Hough transform to detect arbitrary shapes. pp.714-725.

— Canny, J., 1986. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6), pp.679-698.

— Docs.opencv.org. 2021. OpenCV: Canny Edge Detection. Available at: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html [Accessed 10 November 2021].

— Docs.opencv.org. 2021. OpenCV: Hough Line Transform. Available at: https://docs.opencv.org/3.4/d3/de6/tutorial_js_houghlines.html [Accessed 4 December 2021].