GKNB_INTM038 Gépi Látás beadandó feladat

Fejlesztői dokumentáció

Máthé Dávid György R4PBVN

A program közlekedési táblákat detektál képekről. A program C++ nyelven íródott (ISO C++ 14 Standard) és az OpenCV 4.5.4 verzióját használja. A program 6 különböző tábla felismerésére képes. A program eldönti, hogy van-e a képen vizsgált közlekedési tábla, és ha van, melyik a 6 előre meghatározott tábla közül. A program a /kepek mappából tölti be a képeket illetve a /kepek/template mappából a mintákat. A program az állj! elsőbbségadás kötelező, a behajtani tilos, az elsőbbségadás kötelező, az autópálya, a parkoló és a főútvonal táblákat ismeri fel.

A program részei

A forráskód 3 .cpp fájlból és 2 .h fájlból áll. Ezek: main, preprocess, smatch. A vizsgált adatkészlet 100 darab .png képből áll a /kepek mappában. A program futás során létrehoz egy eredmeny.txt fáljt, melybe az eredményeket írja bele.

preprocess.cpp

A preprocess.cpp fájlban 4 függvény van definiálva.

void trackbar(Mat input) {}

Ez a függvény egy trackbar ablakot hoz létre, ahol beállítható a képre alkalmazott inRange hue, saturation és value értékei. A képet először HSV színtérbe konvertálja és létrehozza a trackbar ablakot. egy while ciklusban a képre folyamatosan alkalmazza az inRange függvényt, melynek paramétereit így lehet állítani a trackbar ablakban. Beállítható, hogy a képre alkalmazott inRange milyen határértékek közötti képpontokat keressen, a hmin, smin, vmin az alsó határértékeket, míg a hmax, smax és vmax értkékek a felső határt határozzák meg. A függvény void visszatérésű és a szoftver futása közben nem kerül meghívásra, viszont a program írásakor szükség volt arra, hogy gyorsan és egyszerűen megtalálhatóak legyenek bizonyos H,S,V értékek a színre szűrő függvények megfelelő beállítása érdekében.

Mat red(Mat input) {}

Ez a függvény a kapott input képből egy bináris képet állít elő inRange függvénnyel. A képet HSV színtérbe konvertálja majd inRange függvénnyel előállít egy bináris képet ahol a megadott tartományon belüli képpontok fehér, az azon kívül esők fekete színűek lesznek. Jelen esetben ez egy élénk piros színtartomány. Ezután megfordítja a képpontokat oly módon, hogy a fekete képpontok lesznek fehérek, a fehérek pedig feketék. Az így kapott képet adja vissza a függvény.

Mat blue(Mat input) {}

Mat yellow (Mat input) {}

A red függvényhez hasonlóan működnek, de az inRange függvény más színtartományra szűr, az első függvényben kékre, a másodikban sárgára.

smatch.cpp

A smatch.cpp fájlban 7 függvény van definiálva.

bool compareContourAreas(vector<Point> contour1, vector<Point> contour2) {}

Ez a függvény két kontúr területét vizsgálja. Igazzal tér vissza, ha az első kontúr nagyobb mint a második és hamissal ha kisebb vagy méretük egyenlő. A méreteket a contourArea függvénnyel számolja ki. Erre a kódrészre azért van szükség, mert az smatch függvény végrehajtása során sorba kell rendezni a képen felismert kontúrokat azok területe alapján.

double smatch(Mat input, Mat inputtempl) {}

A program e függvénye végzi a hasonlítást. Az input kép a korábban színszelektált kép, az inputtempl pedig a minta, amihez hasonlítani fogjuk a képet. A minta képet szürkeárnyalatossá konvertálja majd megkeresi benne a kontúrokat melyeket eltárol a contours változóba. Egy kontúr egy pontokból álló vektor, és ezeket a kontúrokat egy vektorban tároljuk. Miután a findContours algoritmus megtalálta és eltárolta a kontúrokat, ezeket sort segítségével csökkenő sorrendbe rendezzük (itt kerül felhasználásra a compareContourAreas) és kiválasztjuk a legnagyobb kontúrt. A legtöbb mintakép esetében nem is talál egynél több kontúrt. Ezután lefuttatjuk a findContours algorimust a vizsgált képre is, a contours változóban pedig eltároljuk a megtalált körvonalakat. Ezután az összes eltárolt kontúrt összehasonlítjuk a mintában talált kontúrral és a bestmatch változóban eltároljuk a legjobb eredményt. Az eredmény minél kisebb, annál pontosabb az egyezés a két kontúr között. A matchShapes algoritmus a két körvonalat vagy alakzatot mérettől, orientációtól, forgatástól függetlenül képes hasonlítani egymáshoz (Hu moments). Végül visszatérésre kerül a bestmatch érték.

bool checkredtriangle(Mat redimg) {}

Ez a függvény piros háromszöget keres a képen, és ha talál, akkor igaz értékkel tér vissza. Bemenetként a piros színre szelektált képet kapja, így gyakorlatilag az algoritmus egy fehér hátterű fekete háromszöget keres. Csak azok a képpontok lehetnek feketék, melyek az eredeti képen pirosak voltak, így ha talál háromszöget akkor az biztosan piros. Első lépésként előállítja a kép éldetektált változatát Canny éldetektálással. Ezt követően az éldetektált képre végrehajt egy dilatációt. Az így kapott képből megkeresi és eltárolja a körvonalakat. Az összes kontúrra végrehajtja az approxPolyDP függvényt (Douglas-Peucker algoritmus), ami előállítja az alakzatok egyszerűsített körvonalát. Ha az így kapott körvonalnak három oldala van, akkor az eredeti képen van piros háromszög, így igazzal tér vissza a függvény. Az approxPolyDP algoritmust csak egy megadott méretűnél nagyobb területet körbezáró kontúrokra futtatja le, így elkerülhető, hogy apró, zajos elemeket is vizsgáljon a képen.

bool checkredcircle(Mat redimg) {}

Ez a függvény piros köröket keres a képen. Bemenetként a piros színre szelektált képet kapja, így gyakorlatilag az algoritmus egy fehér hátterű fekete kört keres. Az előző függvényhez hasonlóan Canny éldetektorral előállítja a kép éldetektált változatát és dilatációt hajt végre. HoughCircles függvény egy circles változóban eltárolja az összes megtalált körvonalat. Ha legalább egy kör eltárolásra kerül, akkor az azt jelenti, hogy van az eredeti képen piros kör, így a függvény igaz értékkel tér vissza.

bool checkredoctagon(Mat redimg) {}

Ugyanúgy működik, mint a checkredtriangle függvény, viszont akkor tér vissza igaz értékkel, ha az approxPolyDP által előállított alakzat körvonala 8 oldalú (if (approx.size() == 8)).

bool checkblue(Mat blueimg) {}

Ugyanúgy működik, mint a checkredtriangle függvény, viszont akkor tér vissza igaz értékkel, ha az approxPolyDP által előállított alakzat körvonala 4 oldalú (if (approx.size() == 4)). Az input kép a kék színre szelektált kép.

bool checkyellow(Mat yellowimg) {}

Ugyanúgy működik, mint a checkredtriangle függvény, viszont akkor tér vissza igaz értékkel, ha az approxPolyDP által előállított alakzat körvonala 4 oldalú (if (approx.size() == 4)). Az input kép a sárga színre szelektált kép.

main.cpp

A mainben történik a függvények megfelelő sorrendű meghívása.

struct matchresult {}

Ebben az adatstruktúrában tároljuk a smatch függvény által visszaadott értékeket. Attól függően, hogy a checkredtriangle, checkredcircle, checkredoctagon, checkblue, checkyellow függvények igaz vagy hamis értéket adnak-e vissza, nem biztos, hogy minden képnél minden hasonlítás minden mintával megtörténik.

string findbestmatch(matchresult m) {}

Megkeresi a legkisebb értéket a matchresult adatstruktúrában és a legkisebb érték alapján előállít egy sztringet, melyet visszaad. Később ezt a sztringet kiírjuk a fájlba (ha talált a képen táblát).

int main() {}

Első lépésként deklaráljuk a matchresult adatstruktúrát, az eredményfájlt, a képek helyét, majd beolvassuk a mintákat a kepek/template/ mappából. Deklarálásra kerül 4 Mat típusú mátrix is a kép illetve annak színre szelektált változatainak tárolására. Ezután for ciklus indul, mind a 100 vizsgált képet egyénileg beolvassuk és eltároljuk. Fontos, hogy a képek nevei számok legyenek 0-tól 99-ig, mert a program a képek nevét az i változóból állítja elő a beolvasáshoz. A képeket átméretezzük, ha túl kicsik lennének. Erre ezért van szükség, mert a checkredtriangle, checkredcircle, checkredoctagon, checkblue, checkyellow függvények csak olyan alakzatokat vizsgálnak, melyek területe meghalad egy alsó határt, így kicsi képek esetén lehetséges volna, hogy nem kerülnek vizsgálatra a táblák. A mátrix sorainak és oszlopainak a számát addig szorozzuk 1.5-el, míg mindkettő el nem éri legalább az 1000 képpontot. Ezután meghívjuk a képre a színszelektáló függvényeket (red, blue, yellow), és eltároljuk az így létrejött képeket. A továbbiakban a program ezekkel dolgozik. Meghívásra kerül a checkredtriangle, checkredcircle, checkredoctagon, checkblue, checkyellow függvény. Ha a program nem talál például piros háromszöget, abban az esetben nem fog lefutni a hasonlítás az elsőbbség adás kötelező tábla mintájával a képre. Ha semmilyen vizsgált alakzatot nem talál a program, azaz mindegyik függvény hamis értékkel tér vissza, akkor a program úgy ítéli meg, hogy nincsen vizsgált tábla a képen. Ha viszont az adott függvény igaz értékkel tér vissza, akkor megtörténik a hasonlítás az smatch algoritmussal. Az smatch algoritmus a kép táblához megfelelő színszelektált képével fog lefutni (tehát ha talál például piros háromszöget, abban az esetben az smatch algoritmus az elsőbbségadás kötelező mintával és a piros színre szelektált képpel fog lefutni). A hasonlítás eredményét eltároljuk a matchresult adatstruktúra megfelelő változójában és az eredmény kiírásra kerül a fájlba. A két szélső eset egyike az, amikor a program mind az 5 alakzatot detektálja a képen, így mind a 6 mintára megtörténik a hasonlítás (a parkoló és az autópálya tábla is kék alapon négyszög alakú tábla, így ha talál kék négyszöget, akkor parkoló és autópálya táblára is vizsgál). A másik szélső eset az, amikor nem talál alakzatot, így nem történik egyetlen hasonlítás sem. Ilyenkor a „NINCS TÁBLA A KÉPEN” szöveg kerül kiírásra a fájlba. Ezután előkészítésre kerül a következő iteráció: a matchresult változóit visszaállítjuk 100 értékre mert a program futása során csak egyetlen egy matchresult struct-ot használ, azaz nem tárolja a memóriában mind a 100 kép összes hasonlítási értékét, hiszen felesleges lenne (az eredmények már benne vannak a fájlban). Ezután bezárásra kerül a fájl. A szoftver az állj! elsőbbségadás kötelező táblákra csak egyszerűen „stop” táblaként hivatkozik mindenhol, annak ellenére, hogy ilyen nem az a teljesen megfelelő kifejezés. Ez amiatt van, hogy ne lehessen még csak véletlenül se összekeverni az állj! elsőbbségadás kötelező táblát az elsőbbségadás kötelező táblával, ha módosítjuk a kódot vagy használjuk a programot.

Hogyan javítható a program/eredmény

A program által meghívott függvények paraméterinek megváltoztatásával javítható az eredmény. A checkredtriangle, checkredoctagon, checkblue, checkyellow függvényekben (smatch.cpp) minél kisebbre választjuk az „area” változót, annál kisebb alakzatokra is le fog futni az approxPolyDP, így „érzékenyebb” a függvény a kisebb alakzatokra is. Ennek hátránya, hogy könnyen felismerhet a program zajos elemeket a képen alakzatként. A checkredcircle függvényben a HoughCircles param2 paramétere minél kisebb, annál valószínűbb, hogy nem teljesen tökéletes köröket is körként fog felismerni a program. Ez ezért lehet rossz, mert a piros nyolcszög a stop táblák esetében majdnem kör alakú, így ha ezt a param2 paramétert mondjuk 100-ra állítjuk, úgy nagyon sok nyolcszög alakú stop táblát is fel fog ismerni körként. A megfelelő érték 100 és 200 között van (200-nál túl kevés kört ismert fel a program). Ezekben a függvényekben a Canny és a dilatáció paramétereinek változtatásával is potenciálisan jobb eredmény érhető el.

A preprocess.cpp-ben a színszelektáló algoritmusok inRange paramétereinek állításával jelentős változásokat vagyunk képesek elérni. Az alsó és felső határok Scalarjainak az S értékét ha szűkebb tartományra állítjuk, akkor sokkal pontosabban tudunk az adott színárnyalatokra szűrni, viszont előfordulhat, hogy különböző látási és fényviszonyok, illetve kifakult táblák esetén nem leszünk képesek felismerni a táblát. Bizonyos képeknél szükség van a nagyobb érzékenységre, míg másoknál pont hogy szigorúbb tartományok megadása javíthatna az eredményen. A program a nagyon élénk színekre szűr (minimum S érték 170) de még így is előfordulnak input képtől függően zajok, olyan tartományok, amiket nem lenne megfelelő vizsgálni (ezért is van az approxpolyDP algoritmus elött egy if ág ami csak nagy területű kontúrokat vizsgál). Az adatkészletben a 28.png, a 40.png, 68.png, a 79.png olyan táblákat ábrázol, amik megfakultak. Ezekben az esetekben a program nem képes jól felismerni ezeket a táblákat, a szín szelekció szűk tartományai miatt. Más esetekben, például a 43.png esetén a lámpára kifüggesztett sárga hirdetőtáblának az árnyalata pont beleesik az vizsgált tartományba, ilyen esetekben a tartomány szűkítésével érhető el lehetségesen jobb eredmény. Ha a színszelekciót követően erodációt hajtunk végre, azzal kiszűrhetjük a zajok egy részét, de potenciálisan elveszhetnek olyan részek, melyekre szükség van a vizsgálat során.