Paradicsom

Mechatronika projekt

Révész Levente  
Szepessy Tamás  
Forrai Benedek

# Absztrakt

Üvegházi paradicsomtermesztés során használt, a növényt tartó alkatrészek automatikus áthelyezéséhez terveztünk képfelismerő szoftvert. Az OpenCV programkönyvtár segítségével készítettünk objektumfelismerő algoritmust, amely az alkatrész pontos pozícióját képes meghatározni. A pozíció ismeretében az alkatrész robotkarral áthelyezhető.

# Motiváció

Az agrárszektorban napjainkban egyre csökken a munkaerő. A szántóföldi növények termesztésében az automatizálás mára igen magas fokot ért el, azonban a zöldségtermesztés területén még nagy potenciál rejlik az automatizálás számára. Precíziós, önálló gépek bevonásával a zöldségtermesztés munkaigénye jelentősen csökkenthető, miközben a termelékenység növekedhet.

A négyzetméterenként betakarítható termés növelésének egyik új eszköze a magas üvegház. Bizonyos növények, köztük a paradicsom a szokásos termesztési méreténél magasabbra is képes nőni. Ezt lehetővé téve több fény éri a növény leveleit, a termésátlagok a szokásos üvegházi átlagot jóval meghaladják [SZÁMOK].

A magas üvegházakban a növények emberi erővel nehezebben hozzáférhetők, azonban robotokkal ideálisan művelhető az ültetvény.

# A feladat ismertetése

…

# Eszközök kiválasztása

A képfeldolgozásra az OpenCV függvénykönyvtárat (3.4.5) választottuk, programkörnyezetnek pedig a Python fejlesztőkörnyezet legfrissebb verzióját (3.7). Ábrázolásra a Matplotlib könyvtárat használtuk.

Mindhárom szoftvercsomag nyílt forráskódú, és széles körben használt, ezáltal könnyedén találtunk útmutatókat a részfeladatokhoz.

# Csipesz pozíciójának meghatározása

A csipesz türkizkék szivacsa színében elüt minden mástól a paradicsomok közt, ezért szín alapján szegmentáltuk a képet.

Erre a feladatra a cv2.inRange() függvény használható. A függvény egy maszkot hoz létre, mely türkizkék színű pixelek helyén 255, a többi pixel helyén 0 értékű. Ezen a maszkon megkereshetjük a fehér régiók alapján a csipeszek koordinátáit. Ezen kívül egy cv2.bitwise\_and() művelettel összekombinálhatjuk az eredeti képpel, majd megjelenítve ellenőrizhetjük, hogy tényleg a csipeszeket találta meg.

### HSV színtér

(Hue, Saturation, Value) Hengerkoordinátás színtér. A szín 1 koordinátában szerepel, ezért szegmentálásra kényelmesen használható.

Értékkészlete OpenCV-ben: H[0-179], S[0-255], V[0-255]  
Értékkészlete Matplotlib-ben: H[0-1], S[0-1], V[0-1]

A H értékre kiválasztjuk a türkizkék (kb. 100°) egy környezetét. A Saturation és a Value értékeknek tág tarományt választottunk, hogy változó fényviszonyok közt is képes legyen megtalálni a türkiz színt.

lower\_bound = (90, 100, 100) # (H, S, V)   
upper\_bound = (110, 255, 255) # (H, S, V)

### Gauss-elmosás

A módszer egy bizonyos sugárban konvolúciót hajt végre a kép pixelein, így homályossá teszi a teljes képet, hatásosan kiszűrve az apróbb pixelhibákat.

A szivacsok szűrés során fellépő árnyékhibáinak, illetve a kisebb, kék pixelcsoportok eltüntetése miatt egy Gauss-elmosást alkalmaztunk a beszkennelt képre. Így a nagyobb szivacsfelületeknél még nem lép fel akkora torzulás, hogy az befolyásolja a pozícióbecslést, azonban a kisebb kék pixelcsoportok eltűnnek, melyek a későbbi tárgykeresésnél zavaróak lehetnek. A feladatra a cv2.GaussianBlur() függvény használható, melynek (7,7) elmosási sugárértéket állítottunk be.

### Blob Detection

Az OpenCV már rendelkezik egy tárgykereső alapalgoritmussal, mellyel a szegmentált, világos háttér előtt sötét tárgyat képes felismerni, pozícióját listázni.

# OpenCV 3-as verziójától az alábbi konstruktor szükséges  
detector = cv2.SimpleBlobDetector\_create(params)

Ahol a params objektumban megadhatóak a tárgykeresés paraméterei. A feladatunk esetében például a következőkre szűrtünk: terület, csoportok közti távolság, a körkörösség, konvexitás és tehetetlenségi szűrőket pedig eltávolítottuk. A csoportok közti távolságot a csipesz különleges kialakítása miatt növeltük meg, hogy a két szivacsoldalt egy csoportba tartozónak kezelje a program.

# Paraméterek objektuma  
params = cv2.SimpleBlobDetector\_Params()

# Vágási küszöb  
params.minThreshold = 0  
params.maxThreshold = 256

# Területszűrés  
params.filterByArea = True  
params.minArea = 30

# Körkörösség szűrése  
params.filterByCircularity = False

# Konvexitás szűrése  
params.filterByConvexity = False

# Tehetetlenség szűrése  
params.filterByInertia = False

# Csoportok közti távolság  
params.minDistBetweenBlobs = 90

Mivel a maszk a keresett csipeszek helyén fehér (255), a többi helyen pedig fekete (0) színű, ezt invertálva kaphatjuk meg a SimpleBlobDetector működéséhez szükséges fordított maszkot.

reversemask=255-mask

A detektornak ezt az invertált képet megadva, már használható a személyre szabott alakzatkeresőnk, mely a megtalált objektumokra kiadja kulcspontjaink listáját. Ebben szerepel a talált csoport középpontjának X és Y koordinátái pixelben, illetve a befoglaló kör átmérője, szintén pixelben. Nekünk ezek közül a feladat során csak a talált kulcspontok helyzete szükséges.

keypoints = detector.detect(reversemask)

A megtalált pixelbeli helyzetből a kamerarendszert és a csipesz méreteit ismerve már kiszámolhatók a csipeszek helyzetei.

### Befoglaló téglalapok

A BlobDetection

# Csipesz felhelyezésére alkalmas helyek keresése

A csipeszt a paradicsom szárán csak olyan részekre lehet felhelyezni, ahol nincs levéltő. A levéltövek megkereséséhez Hough-egyeneskeresést illetve élkeresést próbáltunk alkalmazni.

Az élkeresés és az egyeneskeresés csak akkor vezetett használható eredményre, amikor az előtérben lévő szár élesen jelent meg és a háttér el volt mosva. Ennek a helyzetnek az előállítására rövid fókuszú és szűk mélységélességi tartománnyal rendelkező optikát kerestünk. Antal Ákos tanár úrral konzultáltunk ebben a kérdésben. Az ő véleménye szerint ez a feladat a sorok sűrűsége (40-50 cm) miatt nem megoldható kamerával, ugyanis ilyen rövid tárgytávolságnál nem érhető el a kívánt mélységélesség. További ötletünk a Microsoft Kinect eszközének alkalmazása volt, azon ez sem alkalmazható, mivel mérési tartománya legjobb esetben is csak 70 centimétertől indul.

**Kérdés: hogyhogy nem lehet olyan kamerát találni, amin a háttér nem homályos? A rövid fókusz nem elég ehhez?**

Eközben Forrai Benedek az üvegházban készített képeket a növényekről. Ezeken feltűnő volt, hogy a levelek sok helyen eltakarják a rálátást a szárra, ezzel ellehetetlenítve mind a csipeszek felismerését, mind a szár struktúrájának elemzését.

# Áttérés csévélhető kampókra

Forrai Benedek javaslatára az iparban elterjedt másik megoldás, a csévélhető kampók alkalmazásának lehetőségét vizsgáltuk.

Az elrendezés az eredetileg ismertetetthez hasonló, de a paradicsom szárát nem csipeszekkel rögzítik statikus dróthoz, hanem a szárat egy ponton megfogva csévélhető huzallal az üvegház tetején elhelyezett sínre függesztik. A növény növekedése során a huzalt egyre lejjebb engedik, a szár új szakaszát pedig terhet nem viselő kapcsokkal a huzalhoz rögzítik.

Az automatizálási feladat a huzalt a sínhez kapcsoló, huzalcsévét tartó kampó alakú alkatrész robotkaros megfogása, horizontális áthelyezése, valamint megforgatása során a huzal lecsévélése.

# Markerválasztás

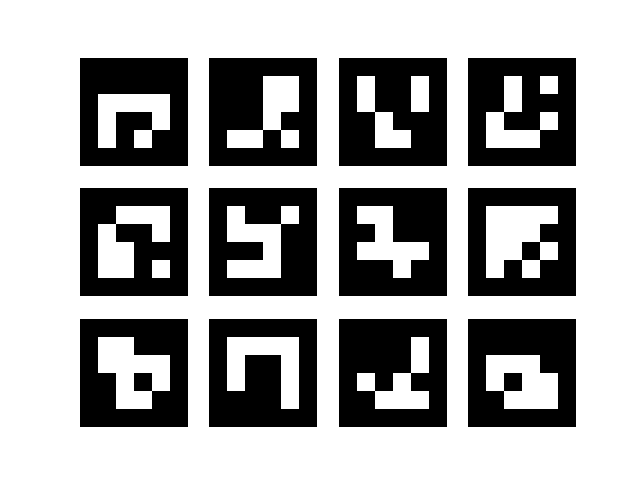
A kampók a csipeszekkel ellentétben nem tartalmaznak szín alapján felismerhető felületet, ezért markert kell rá helyezni. A markernek meg kell mutatnia az eszköz pozívióját és szöghelyzetét is.

Az egyszerűbb megoldás két színes matrica felhelyezése a megfogható tartomány két végére.

Felmerült még a gépi látásban igen elterjedt ArUco markerek használata. Ezek segítségével pozíciót, szöghelyzetet és távolságot is meg lehet határozni egyetlen markerrel, valamint a rögzített kód információtovábbításra is alkalmas, egy sorszámot tartalmaz.

### ArUco

Az ArUco markerek kis négyzet alakú jelölők, hasonlóak a datamatrix és a QR kódokhoz. Az OpenCV elő van készítve a használatukra, a contrib (kiegészítő) csomagok közt elérhető cv2.aruco modul tágas eszköztárat kínál. A contrib csomagok beszerzéséhez az opencv-contrib-python csomagot kell telepíteni a pip csomagkezelőn keresztül.



. ábra: 4X4 bites ArUco markerek

Az egy feladat során használt markereket egy aruco.Dictionary tárolja. Létrehozhatunk egyedi Dictionary-t a Dictionary\_create(…) függvénnyel, de használhatunk előre definiáltakat is. Ezeket a Dictionary\_get( dict ) vagy a getPredifinedDictionary( dict ) függvényekkel érhetjük el (a két függvény ekvivalens). A dict paraméter helyére az aruco.PREDEFINED\_DICTIONARY\_NAME enum elemei kerülhetnek. A kívánt méret és a szükséges markerszám alapján választhatunk, néhány példa az elérhetőek közül:

DICT\_4X4\_50 DICT\_4X4\_100 DICT\_4X4\_250 DICT\_4X4\_1000

DICT\_5X5\_250 DICT\_6X6\_250 DICT\_7X7\_250 DICT\_7X7\_1000

A második szám határozza meg az elérhető markerek maximális számát. Kisebb elemszám esetén nagyobb Hamming-távolság lesz az egyes kódok közt, robosztusabb lehet a felismerés. Nagyobb elemszám esetén értelemszerűen több markert készíthetünk.

A markerek bármilyen orientációban beolvashatók, a program képes meghatározni az orientációjukat, amit a „gazdatárgy” helyzetének bemérésére is használhatunk. Esetünkben egy marker alkalmas lehet az alkatrész pozíciójának és dőlésszögének meghatározására.

A felismerő algoritmus igen gyors. Egy thresholding után lehetséges markereket keres, konvex négyszög alakzatokat világosabb belső résszel. Ezután ezeket a lehetséges markereket dolgozza fel, perspektívatranszformációt hajt végre, Otsu thresholding segítségével különíti el a fekete és fehér színeket, majd szükség esetén hibajavítást végez és eldönti, hogy a marker a Dictionary része-e.

### Kamera kalibráció

A kamerák leképezése torzítással rendelkezik. A képmezőelhajlás (disztorzió) jelenségével kell ezen feladat során foglalkoznunk. A kamera tengelyére merőleges, nagy kiterjedésű síkon (a tárgysíkon) elhelyezkedő egyenesek képei a kép szélein görbék lesznek. Egy négyzethálós minta képén a szélső vonalak egyre jobban görbülnek, hordó alakúvá válik a minta. Emiatt a képen meghatározott távolságok a valóságnak nem felelnek meg; a képen látható egyenesek a valóságban nem egyenesek.

Ez a hiba szoftveresen korrigálható. A disztorzó inverz transzformációját képezhetjük, amit a képre alkalmazva eltüntethetjük a disztorzió hatását. Az OpenCV tartalmaz egy kamerakalibrációs eszközt, ezt használjuk.

# Linkek

Leírás a paradicsomnevelő módszerről  
<https://www.commercial-hydroponic-farming.com/trellising-tomato-plants/>

Object segmentation szín alapján, tutorial  
<https://realpython.com/python-opencv-color-spaces/>

Object tracking színekkel, HSV határ kiválasztása  
<https://docs.opencv.org/3.4.2/df/d9d/tutorial_py_colorspaces.html>

Simple blob detector paraméterei  
<https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html>

Simple blob detector használata  
<https://makehardware.com/2016/05/19/blob-detection-with-python-and-opencv/>

Befoglaló téglalapok a megtalált kontúrokra  
<https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html>

Hough-transzformáció egyenesillesztés  
<https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html>

ArUco Marker  
<https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html>  
Predefined Dictionaries  
<https://docs.opencv.org/3.4/d9/d6a/group__aruco.html#gac84398a9ed9dd01306592dd616c2c975>  
Python tutorial  
<https://mecaruco2.readthedocs.io/en/latest/notebooks_rst/Aruco/aruco_basics.html>

Kamera kalibráció  
<https://docs.opencv.org/2.4.13.7/doc/tutorials/calib3d/camera_calibration/camera_calibration.html>  
<https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>

Szárdetektálási tanulmányok  
<https://www.researchgate.net/publication/286576522_Automatic_plant_branch_segmentation_and_classification_using_Vesselness_measure>

<https://www.researchgate.net/publication/241619223_AUTOMATED_MACHINE_VISION_SENSING_OF_PLANT_STRUCTURAL_PARAMETERS>