Paradicsom

Mechatronika projekt

# Absztrakt

# 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. Ezt az alábbi számítás alapján tesszük meg, majd írjuk ki egy, a robot által később olvasható fájlba.

# 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/

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>