ANEXOS

Diseño y desarrollo de una aplicación para la detección de plantas de maíz.

Design and development of a program to detect corn plants.

Autor/es

Guillermo García Otín

Director/es

Ramón Miralbés Buil

Universidad de Zaragoza

2018

ANEXO I: Características de la aeronave *Phantom 3.*

Peso (Batería y Hélices Incluidas)	1280 g
Tamaño Diagonal (Hélices Excluidas)	350 mm
Velocidad Máx. en Ascenso	5 m/s
Velocidad Máx. en Descenso	3 m/s
Precisión en Vuelo Estacionario	Vertical: +/- 0.1 m (si el Posicionamiento Visual está activado) o +/- 0.5 m Horizontal: +/- 1.5 m
Velocidad Máx.	16 m/s (modo ATTI, sin viento)
Altura Max. de Servicio sobre el Nivel del Mar	6000 m (Límite de altura por defecto: 120 m sobre el punto de despegue)
Temperatura de Funcionamiento	de 0°C a 40°C
Modo GPS	GPS/GLONASS
Máx. Duración de Vuelo	Aproximadamente 23 minutos

ANEXO II: Características de la cámara

Sensor	1/2.3” CMOS Píxeles efectivos: 12.4 M (píxeles totales: 12.76 M)
Lente	FOV 94° 20 mm (35 mm formato equivalente) f/2.8, enfoque a ∞
Rango ISO	100-3200 (vídeo) 100-1600 (foto)
Velocidad Del Obturador	8s -1/8000s
Tamaño Máx. de Imagen	4000×3000
Modos de Fotografía	Disparo único Disparo en ráfaga: 3/5/7 disparos Exposición Automática en Horquillado (AEB): 3/5 Horquilla de Exposición a 0.7EV Bias Time-lapse
Modos de Vídeo	UHD: 4096x2160p 24/25, 3840x2160p 24/25/30 FHD: 1920x1080p 24/25/30/48/50/60 HD: 1280x720p 24/25/30/48/50/60 2.7K: 2704 x1520p 24/25/30 (29.97)
Tipos de Tarjetas SD Compatibles	Micro SD Capacidad Máx.: 64 GB. Clase 10 ó UHS-1 valoración requerida
Tasa de Bits Máx. de Almacenamiento de Vídeo	60 Mbps
Formatos de Archivo	FAT32 (≤ 32 GB); exFAT (> 32 GB)

ANEXO III: Características del mando de control remoto

Frecuencia	2.400 GHz-2.483 GHz
Distancia Máxima	Hasta 5 km ó 3.1 millas (sin obstáculos ni interferencias) Según normas FCC. Hasta 3.5 km ó 2.1 millas (sin obstáculos ni interferencias) Según normas CE
Temperatura de Funcionamiento	de 0°C a 40°C
Batería	6000 mAh LiPo 2S
Soporte para Dispositivo Móvil	Para tableta o teléfono móvil
Transmisor de Potencia (PIRE)	FCC: 20 dBm CE: 16 dBm
Voltaje de Funcionamiento	1.2 A @7.4 V

ANEXO IV: Características de la batería de vuelo inteligente

Capacidad	4480 mAh
Voltaje	15.2 V
Tipo de Batería	LiPo 4S
Energía	68 Wh
Peso Neto	365 g
Temperatura de Funcionamiento	de -10°C a 40°C
Potencia de Carga Máx.	100 W

ANEXO V: Plantas de maíz y sus características.

La planta del maíz puede llegar a medir de unos 270 a 290 centímetros como promedio, aunque en algunas especies y cultivadas en lugares muy específicos puede llegar a medir más de 3 metros. Entre las características del maíz no se puede dejar de mencionar su típica estructura y las partes que forman a la planta del maíz: tallo, hojas y flores.

El tallo del maíz es cilíndrico y hueco lo que hace que sea resistente a fuertes vientos, sequías o suelos que no tienen los suficientes nutrientes para su cultivo y evolución. El tallo está principalmente estructurado en:

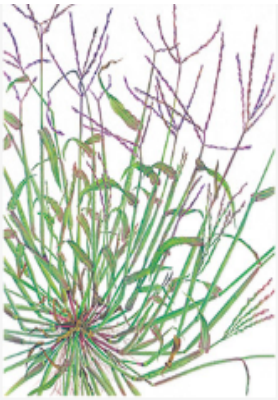
- Epidermis: es una capa transparente que protege a la planta contra posibles amenazas de insectos u otras enfermedades.
- Pared: es la capa dura por donde transitan las sustancias que lo alimentan.
- Médula: es un tejido flácido y esponjoso que se encuentra en la parte central del tallo y es donde almacena las sustancias alimenticias.

Las hojas del maíz son alargadas y un poco onduladas, su aspecto en el borde de la hoja es áspero. Salen alternas y nacen muy pegadas al tallo, donde se desarrollan las mazorcas. Se dice que las hojas tienen una gran importancia en el desarrollo y evolución de los granos. Dependiendo de cómo se cultiva una planta de maíz puede tener desde 12 hasta 24 hojas.

Cada planta de maíz desarrolla dos flores. La de arriba es la flor masculina, se llama *'espiga'* y tiene estambres que producen polen. La de abajo es femenina, está envuelta en hojas de las que sobresalen los estigmas a los que conocemos como pelos de lotes. Cuando el polen de esa planta o de una cercana fecunda a las flores femeninas crecen los frutos, cada granito de lotes es una fruta individual y la mazorca completa es una fruta múltiple como las moras o las piñas.

<u>ANEXO VI:</u> Malas hierbas de color violeta y sus características.

Digitaria



(*Digitaria sanguinalis (L.) Scop.*)

Descripción:

- **Tipo:** Monocotiledónea.
- **Nombre común:** Digitaria.
- **Nombre latín:** *Digitaria sanguinalis (L.) Scop.*
- **Familia:** Gramineae.
- **Sinónimos:**

Descripción de la plántula:

- Prefoliación enrollada, vaina cilíndrica.
- Primera hoja ancha y aplicada al suelo.
- Limbo de 2 a 5 veces más largo que ancho, hoja más o menos peluda.
- Lígula membranosa dentada, ausencia de aurículas.

Descripción planta adulta:

- Altura: 10 a 60 cm. Pilosidad abundante en toda la planta. Tallos ramificados extendidos ascendentes, a menudo radicantes en los nudos inferiores.
- Hojas: de mucha pilosidad (pelo medio de 0,5 mm) en las dos caras. Anchura notable de los limbos (casi 6 mm en la segunda hoja), hojas bastante cortas. Vaina verde a rojo violeta (de ahí la denominación sanguínea).
- Inflorescencias: digitadas, de 4 a 10 falsas espigas muy largas, que parten de varios puntos cercanos entre sí (diferencia con las inflorescencias de la grama pata de gallina). Espiguillas en cortos pedículos unifloros, glumas inferiores muy pequeñas, glumas superiores del mismo largo que la espiguilla, recubiertas de pelos finos.
- Frutos: cariópsides encerradas en las 2 glumelas.

Particularidades:

- Preferencia por los suelos ligeros.

Clave para la identificación de las malas hierbas gramíneas:

- Ahijado de la planta abierta y las vainas cilíndricas con prefloración enrollada y de color morado en la base, presencia de pelos largos y suaves en hojas y vainas desde el inicio (A).
- Presencia de lígula (B).
- Inflorescencia en espigas umbeliformes (C).

Daño:

- Alto.

ANEXO VII: Aplicación creada.

```
import numpy as np
import cv2
from cv2 import inRange
from operator import div


def Si_estan_cerca(cnt1,cnt2): #Si hay algunos pixeles de los dos contornos lo suficientemente cerca devuelve true sino false
    row1,row2 = cnt1.shape[0],cnt2.shape[0] # El numero de filas
    for i in xrange(row1):
        for j in xrange(row2):
            dist = np.linalg.norm(cnt1[i]-cnt2[j])
            if abs(dist) < 50 :
                return True
            elif i==row1-1 and j==row2-1:
                return False

def distanciaPixeles(xf,yf,xv,yv):
    a=(abs(((xv-xf)^2)-((yv-yf)^2))**0.5)

    return a;


def Max_distx(p1,p2,p3):
    if (abs(p2[0]-p1[0]) > abs(p1[0]-p3[0])) and (abs(p2[0]-p1[0]) > abs(p2[0]-p3[0])):
        return abs(p2[0]-p1[0])
    elif  (abs(p2[0]-p3[0])  >  abs(p1[0]-p2[0]))  and  (abs(p2[0]-p3[0])  > abs(p1[0]-p3[0])):
        return abs(p2[0]-p3[0])
    elif  (abs(p1[0]-p3[0])  >  abs(p1[0]-p2[0]))  and  (abs(p1[0]-p3[0])  > abs(p2[0]-p3[0])):
        return abs(p1[0]-p3[0])


def Esta_en_linea(p1,p2,p3):
    tol = 10000
    Area_triangulo = 0.5*abs((p1[0]*(p2[1] - p3[1])) + (p2[0]*(p3[1]-p1[1])) + (p3[0]*(p1[1] - p2[1])))
    if (Area_triangulo < tol):
        return True
    else:
        return False
print('Introduzca los datos referentes a los colores:')
R_maiz = int(raw_input('Rojo_maiz:168'))
V_maiz = int(raw_input('Verde_maiz:215'))
A_maiz = int(raw_input('Azul_maiz:112'))

green = np.uint8([[[A_maiz,V_maiz,R_maiz]]])
hsv_ma = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
hsv_maiz = hsv_ma.reshape(3,1)


R_malas = int(raw_input('Rojo_malas:102'))
```

```
V_malas = int(raw_input('Verde_malas:0'))
A_malas = int(raw_input('Azul_malas:205'))

violet = np.uint8([[[A_malas,V_malas,R_malas]]])
hsv_mal = cv2.cvtColor(violet,cv2.COLOR_BGR2HSV)
hsv_malas = hsv_mal.reshape(3,1)

print('Introduzca los datos referentes a la camara')
Sw = raw_input('Introduce el parametro Sw de la camara en metros:(0.00677)')
Sh = raw_input('Introduce el parametro Sh de la camara en metros:(0.00455)')
f = raw_input('Introduce distancia focal en metros:(0.035)')
H = raw_input('Introduce altua de vuelo en metros:')
print('Introduzca los datos referentes a la plantacion')
separacion_entre_hileras = raw_input('Introduce distancia de separacion entre
hileras en metros:(0.166)')
separacion_entre_plantas = raw_input('Introduce distancia de separacion entre
plantas en metros:(0.144)')

#Cargar Imagen
img= cv2.imread("image/DJI_0359.JPG")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

#Verdes y violetas
verde_bajos = np.array([(int(hsv_maiz[0]) - 10),(int(hsv_maiz[1]) - 60),90])
verde_altos = np.array([(int(hsv_maiz[0]) + 30), (int(hsv_maiz[1]) + 10), 255])

violetas_bajos = np.array([(int(hsv_malas[0]) - 20),60,60])
violetas_altos = np.array([(int(hsv_malas[0]) + 20),255,255])


#Crear la mascara
mask = cv2.inRange(hsv, verde_bajos, verde_altos)
mask= cv2.GaussianBlur(mask,(21,21), 0)

maskmalas = cv2.inRange(hsv, violetas_bajos, violetas_altos)
maskmalas= cv2.GaussianBlur(maskmalas,(21,21), 0)

cv2.imwrite('masmalas.jpg',maskmalas)
cv2.imwrite('mask.jpg',mask)
#Detectamos los bordes
canny= cv2.Canny(mask,30, 180)

cv2.imwrite('canny.jpg',canny)

cannymalas= cv2.Canny(maskmalas,30, 180)

#Buscamos los contornos
(_,    contours,_)    =    cv2.findContours(canny.copy(),    cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

(_, contoursmalas,_) = cv2.findContours(cannymalas.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)



#Calculo la correspondencia de pixeles a metros
pixeles_y, pixeles_x = img.shape[:2]
distancia_x_real= (float(Sw)*float(H))/float(f)
distancia_y_real= (float(Sh)*float(H))/float(f)
```

```
lado_pixel_x=distancia_x_real/pixeles_x
lado_pixel_y=distancia_y_real/pixeles_y

#Cierro los contornos
LENGTH = len(contours)
status = np.zeros((LENGTH,1))

for i,cnt1 in enumerate(contours): #Repite en contornos. La i llevara la cuenta (0,1,2,3...) a la vez que los contornos.
    x = i
    if (i != LENGTH-1):
        for j,cnt2 in enumerate(contours[i+1:]):
            x = x+1
            if (Si_estan_cerca(cnt1,cnt2) == True):
                val = min(status[i],status[x])
                status[x] = status[i] = val
            else:
                if (status[x]==status[i]):
                    status[x] = i+1

unified = []
maximum = int(status.max())+1
for i in xrange(maximum):
    pos = np.where(status==i)[0]
    if (pos.size != 0):
        cont = np.vstack(contours[i] for i in pos)
        hull = cv2.convexHull(cont)
        unified.append(hull)


#Elimino los contornos demasiado pequenos
lista=[c for c in unified if cv2.contourArea(c) > 10]

#Creo un array con los centros en vez de todos los pixeles que forman las plantas.
centros=[]
for i in lista:
    M = cv2.moments(i)
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    aux=[]
    aux.append(cX)
    aux.append(cY)
    centros.append(aux)




#Cierro los contornos de las malas
LENGTHMALAS = len(contoursmalas)
statusmalas = np.zeros((LENGTHMALAS,1))

for i,cnt1 in enumerate(contoursmalas): #Repite en contornos. La i llevara la cuenta (0,1,2,3...) a la vez que los contornos.
    x = i
    if (i != LENGTHMALAS-1):
        for j,cnt2 in enumerate(contoursmalas[i+1:]):
            x = x+1
```

```
            if (Si_estan_cerca(cnt1,cnt2) == True):
                val = min(statusmalas[i],statusmalas[x])
                statusmalas[x] = statusmalas[i] = val
            else:
                if (statusmalas[x]==statusmalas[i]):
                    statusmalas[x] = i+1

unifiedmalas = []
maximummalas = len(statusmalas)+1
for i in xrange(maximummalas):
    posmalas = np.where(statusmalas==i)[0]
    if (posmalas.size != 0):
        contmalas = np.vstack(contoursmalas[i] for i in posmalas)
        hullmalas = cv2.convexHull(contmalas)
        unifiedmalas.append(hullmalas)


#Elimino los contornos demasiado pequenos
listamalas=[c for c in unifiedmalas if cv2.contourArea(c) > 10]

#Creo un array con los centros en vez de todos los pixeles que forman las
plantas.
centrosmalas=[]
for i in listamalas:
    M = cv2.moments(i)
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    aux=[]
    aux.append(cX)
    aux.append(cY)
    centrosmalas.append(aux)



print(len(listamalas))

for i in centrosmalas:
    cv2.drawContours(img,listamalas,-1,(0,0,255),2)


#Compruebo si las plantas estan en linea recta
#Calculo el triangulo que forman tres plantas y si el area es suficientemente
pequena entonces estan en linea
maiz=[]

for p1 in centros:
    for p2 in centros:
        for p3 in centros:
            if ((p1 != p2) and (p3 != p2) and (p3 != p1)):
                if      ((Esta_en_linea(p1,p2,p3)      ==      True)      and
(Max_distx(p1,p2,p3)<500)):
                    if (p3 in maiz) == False:
                        maiz.append(p3)
                    if (p2 in maiz) == False:
                        maiz.append(p2)
                    if (p1 in maiz) == False:
                        maiz.append(p1)
```

```
#Creo un array (mahier) comprobando si los centros de las plantas estan en el
array maiz las cuales son las plantas buenas detectadas
mahier=[]
for x in centros:
    if (x in maiz) == False:
        mahier.append(x)



#pinto las aeras malas de diferente color
for i in mahier:
    prueba=[]
    prueba.append(lista[centros.index(i)])
    cv2.drawContours(img,prueba,-1,(0,0,255),2)

#pinto las areas buenas
for i in centros:
    if (i in mahier) == False:
        prueba=[]
        prueba.append(lista[centros.index(i)])
        cv2.drawContours(img,prueba,-1,(0,255,0),2)


#Mostramos el numero de plantas por consola
print("He encontrado {} plantas".format(len(maiz)))
malas=len(centros)-len(maiz)+len(centrosmalas)
print("He encontrado {} malas plantas".format(malas))

#Muestro el indice de nacencia
plantas_teoricas=
(pixeles_y*pixeles_x)/(float(separacion_entre_hileras)*float(separacion_entre
_plantas))
indice_nacencia=(len(maiz)*100) /plantas_teoricas

#Creo las listas de buenas plantas y malas con las areas
BuenImp=[]
MalaImp=[]
auxiliar=[]
for i in centros:
    if (i in mahier) == False:
        Area_en_pixeles= cv2.contourArea(lista[centros.index(i)])
        Area= Area_en_pixeles*(lado_pixel_x)*(lado_pixel_y)
        auxiliar.append(Area)
        auxiliar.append(i)
        BuenImp.append(auxiliar)
        auxiliar=[]
    else:
        Area_en_pixeles= cv2.contourArea(lista[centros.index(i)])
        Area=Area_en_pixeles*(lado_pixel_x)*(lado_pixel_y)
        auxiliar.append(Area)
        auxiliar.append(i)
        MalaImp.append(auxiliar)
        auxiliar=[]
Malasvio=[]
for i in centrosmalas:
    Area_en_pixeles= cv2.contourArea(listamalas[centrosmalas.index(i)])
    Area= Area_en_pixeles*(lado_pixel_x)*(lado_pixel_y)
    auxiliar.append(Area)
    auxiliar.append(i)
```

```
    Malasvio.append(auxiliar)
    auxiliar=[]


print('Las areas y la posicion en coordenadas x e y de las plantas buenas son:')
for c in BuenImp:
    print(c)

print('Las areas y la posicion en coordenadas x e y de las plantas malas de igual color qeu las de maiz son:')
for c in MalaImp:
    print(c)

print('Las areas y la posicion en coordenadas x e y de las plantas malas de diferente color que las de maiz son:')
for c in Malasvio:
    print(c)

#Las plantas que hay que regar
suma=0
for i in BuenImp:
    suma=suma+ i[0]

suma_media=suma/len(BuenImp)

print('las plantas quehay que regar son:')
for i in BuenImp:
    if (i[0] < (suma_media/2)):
        print(i)



#Guardo la imagen
cv2.imwrite('img.jpg',img)

#Salir con ESC
while(1):
    tecla = cv2.waitKey(5) & 0xFF
    if tecla == 27:
        break

cv2.destroyAllWindows()
```

ANEXO VIII: Utilización de *cvtColor*.

cvtColor

Converts an image from one color space to another.

C++: void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

Python: cv2.**cvtColor**(src, code[, dst[, dstCn]]) → dst

C: void **cvCvtColor**(const CvArr* **src**, CvArr* **dst**, int **code**)

Python: cv.**CvtColor**(src, dst, code) → None¶

Parameters:
- **src** – input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.
- **dst** – output image of the same size and depth as src.
- **code** – color space conversion code (see the description below).
- **dstCn** – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code .

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

The conventional ranges for R, G, and B channel values are:

- 0 to 255 for CV_8U images
- 0 to 65535 for CV_16U images
- 0 to 1 for CV_32F images

ANEXO IX: Utilización de *GaussianBlur.*

GaussianBlur

Blurs an image using a Gaussian filter.

C++: void `GaussianBlur`(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)

Python: `cv2.GaussianBlur`(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst

Parameters:
- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from `sigma*` .
- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height` , respectively (see `getGaussianKernel()` for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of `ksize`, `sigmaX`, and `sigmaY`.
- **borderType** – pixel extrapolation method (see `borderInterpolate` for details).

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

ANEXO X: Utilización de *Canny*.

Canny

Finds edges in an image using the [Canny86] algorithm.

C++: void **Canny**(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)

Python: cv2.**Canny**(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]) → edges

C: void **cvCanny**(const CvArr* **image**, CvArr* **edges**, double **threshold1**, double **threshold2**, int **aperture_size**=3)

Python: cv.**Canny**(image, edges, threshold1, threshold2, aperture_size=3) → None

Parameters:
- **image** – single-channel 8-bit input image.
- **edges** – output edge map; it has the same size and type as `image` .
- **threshold1** – first threshold for the hysteresis procedure.
- **threshold2** – second threshold for the hysteresis procedure.
- **apertureSize** – aperture size for the `Sobel()` operator.
- **L2gradient** – a flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (`L2gradient=true`), or whether the default L_1 norm $= |dI/dx| + |dI/dy|$ is enough (`L2gradient=false`).

The function finds edges in the input image `image` and marks them in the output map `edges` using the Canny algorithm. The smallest value between `threshold1` and `threshold2` is used for edge linking. The largest value is used to find initial segments of strong edges. See http://en.wikipedia.org/wiki/Canny_edge_detector

ANEXO XI: Utilización de *ConvexHull*.

```
hull = cv2.convexHull(points[, hull[, clockwise[, returnPoints]]
```

Arguments details:

- **points** are the contours we pass into.
- **hull** is the output, normally we avoid it.
- **clockwise** : Orientation flag. If it is True, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise.
- **returnPoints** : By default, True. Then it returns the coordinates of the hull points. If False, it returns the indices of contour points corresponding to the hull points.