PROYECTO FINAL

Visión Robótica

DESCRIPCIÓN BREVE

Utilizando la cámara del drone Bebop2, se guarda una imagen para procesarla y, al detectar un círculo de color amarillo, el drone aterrizará en el centro.

Gerardo Daniel Naranjo Gallegos, A01209499
Elvis Enrique Jiménez Reyes, A01204803

15/05/2019

Tabla de contenido

Introducción

La visión artificial es un sector de la inteligencia artificial que hace uso de las técnicas adecuadas para la obtención, procesamiento y análisis de cualquier tipo de información deseada, obtenida a través de imágenes digitales. Se compone por un conjunto de procesos destinados a realizar el análisis de imágenes, tales como: obtención de imágenes, memorización de la información, procesamiento e interpretación de los resultados.

Sin lugar a duda, la visión artificial es un campo de gran utilización y crecimiento en la actualidad. La visión artificial puede ser empleada de distintas formas y para distintos objetivos. En este curso en particular, se tratará la visión robótica y en esta práctica se elaborará un proyecto referente a drones.

Descripción de la práctica

Esta práctica consiste en desarrollar un proyecto usando nuestros conocimientos adquiridos de visión utilizando un dron, el cual es posible programar sus funciones mediante una librería de Python, permitiendo total manipulación del dispositivo.

El proyecto tenía como objetivo utilizar la cámara del dron, obteniendo la información de lo que el dron capta para poder procesar los datos e indicarle al mismo una acción a ejecutar. En este caso nosotros utilizamos la cámara del dron para que fuera capaz de reconocer objetos circulares, así poder calcular el centro mediante el radio y ejecutar un comando de aterrizaje.

Para lograr la ejecución, fue necesario tener conocimientos en el lenguaje Python, el cual cuenta con las librerías que permiten tener control y recibir información proveniente del dron, una vez que se reciben los datos, se mandan a la computadora la cual es capaz de procesarla, en con otra librería de Python, llamada OpenCV especializada para el procesamiento de imágenes, poder mandar instrucciones, según los datos que se reciban.

El desarrollo se realizó entre dos personas y con base a toda la documentación encontrada en internet, sobre las librerías para el procesamiento de imagen y para el control del dron, se pudo lograr combinando ambas tecnologías, el objetivo de la práctica.

Requisitos y material utilizado

Los materiales necesarios y utilizados a lo largo del proyecto son los siguientes:

- Aro de color.
- Conexión a internet para instalación de software.
- PC con el software requerido (ver más adelante en esta misma sección).
- Drone Bebop 2, con batería completamente cargada, hélices y protecciones.
- Arena de drones (puede ser cualquier espacio reservado para el vuelo y práctica con drones).

En cuanto a los requisitos, es imperativo la instalación del software que se detalla a continuación, para la elaboración e implementación del proyecto:

VLC

VLC es un reproductor y *framework* multimedia, disponible para diversas plataformas y cuenta con la característica de ser de código abierto. Este programa será utilizado para el *streaming* de video desde el drone a la PC. (VideoLAN, 2019)

Para instalarlo, se puede descargar desde la página oficial: https://www.videolan.org/vlc/. Cabe mencionar que al momento de la realización de este proyecto se utilizó la versión 3.0.6 y ésta en particular puede ser descargada desde: http://download.videolan.org/pub/videolan/vlc/3.0.6/.

Durante el proceso de instalación, se dejaron todas las opciones por defecto, incluida la ruta de instalación y el tipo de instalación (completa). **Es importante instalar la versión de 64 bits**.

Anaconda Navigator

Como se puede leer en su propia página web, Anaconda Navigator es una interfaz gráfica de usuario (GUI) de escritorio, que le permite iniciar aplicaciones y administrar fácilmente los paquetes, entornos y canales de **Conda** sin usar los comandos de la línea de comandos. El navegador puede buscar paquetes en Anaconda Cloud o en un repositorio local de Anaconda. Está disponible para Windows, macOS y Linux. (Anaconda, Inc, 2019)

Se puede encontrar una completa guía de instalación en su página oficial: https://docs.anaconda.com/anaconda/install/windows/. Y se puede descargar en la siguiente liga oficial: https://www.anaconda.com/distribution/#windows.

Gracias a la instalación de este software, contaremos con Spyder: un software potente con un entorno científico escrito en Python, para Python y diseñado por y para científicos, ingenieros y analistas de datos. Ofrece una combinación única de la avanzada de edición, análisis, depuración y perfiles de funcionalidad de una herramienta de desarrollo integral con la exploración de datos, ejecución interactiva, inspección profunda y capacidades de visualización de un paquete científico. Más allá de sus muchas funciones incorporadas, su capacidad puede ampliarse aún más a través de su sistema de *plugins* y API.

OpenCV y paquetes, vía comandos en Prompt

OpenCV es una librería para visión artificial de código abierto y que está disponible para muchos lenguajes de programación, entre ellos se encuentra Python.

Una vez finalizada la instalación de Anaconda, será necesario acceder a la siguiente página web para descargar el archivo de instalación más reciente (con extensión WHL): https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv.

Se deberá seleccionar la opción adecuada a la PC; en la PC utilizada para este proyecto fue la versión llamada opencv_python-4.1.0-cp37-cp37m-win_amd64.whl. Una vez descargado el archivo, guardar en una carpeta de fácil acceso y copiar la ruta (*path*) al archivo.

Posteriormente, ejecutar Anaconda Prompt **como administrador** y teclear el siguiente comando, cambiando la ruta a la dirección correcta del archivo:

```
pip install "C:\OpenCV\opencv_python-3.2.0+contrib-cp36-cp36m-
win_amd64.whl"
```

De igual forma, ejecutar los siguientes comandos en Prompt:

```
pip install untangle
pip install zeroconf
pip install pyparrot
pip install keyboard
```

Con estos comandos se habrán instalado todos los paquetes necesarios. (McGovern, Installation, 2018)

Conexión al drone Bebop 2

Es necesario encender el drone y conectarse a él desde la computadora. Para ello, es cuestión de buscarlo entre las redes Wifi y conectarse a él:

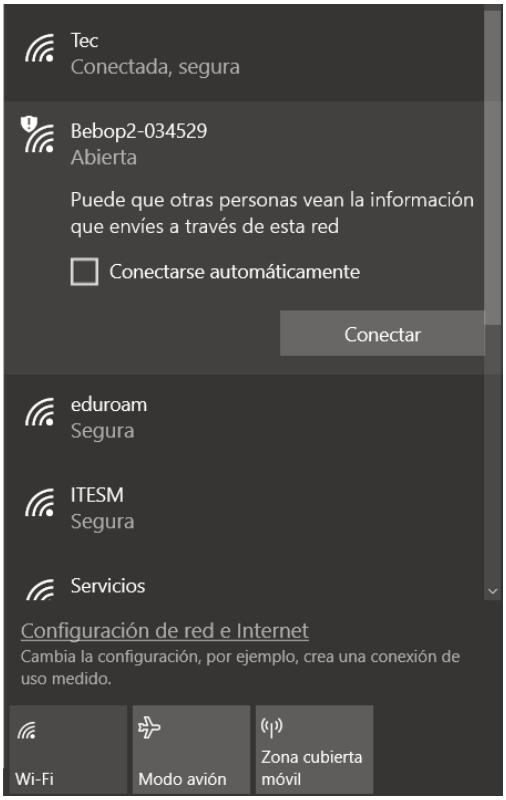

Imagen 1. Selección del drone entre redes Wifi.

Permisos de acceso a archivos

Se deberán otorgar permisos de acceso a la carpeta de Anaconda3. Para ello, es necesario ir a la carpeta Anaconda 3, generalmente ubicada en la ruta C:\ProgramData\Anaconda3 y dar clic derecho en la carpeta misma; posteriormente, dar clic en propiedades y pasar a la pestaña llamada Seguridad (ver Imagen 2). Dar clic en Editar y marcar todas las casillas para el usuario correcto. Finalmente, dar clic en aplicar y cerrar todas las ventanas del Explorador de Archivos.

Estos permisos son otorgados con el objetivo de evitar algunos posibles problemas presentados al momento de leer la imagen guardada para el *streaming* de la cámara del drone.

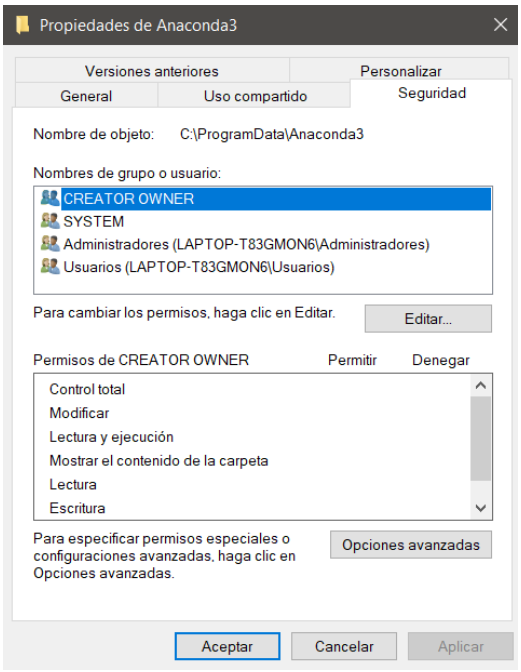

Imagen 2. Propiedades de la carpeta Anaconda3

Permisos de Firewall

Estos permisos son otorgados con el fin de evitar problemas con la comunicación, vía Wifi, con el drone. Para ello, es necesario ir a Firewall y protección de red (buscar así en Cortana y manda a configuración). Subsiguientemente, dar clic en la opción Permitir una aplicación a través del firewall y en ella dar clic en Cambiar configuración. Buscar en la lista a Python y marcar las casillas. Finalmente, dar clic en Aceptar.

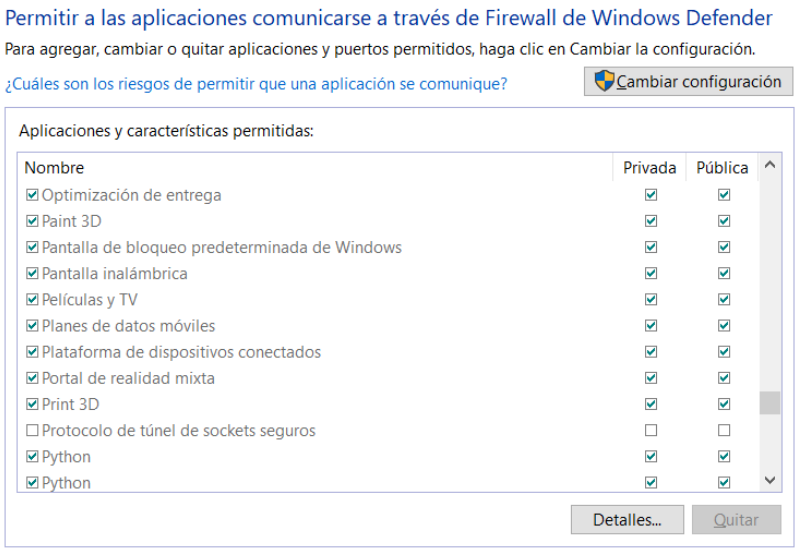

Imagen 3. Permitir Python a través del Firewall de Windows.

Desarrollo de la práctica

Antes de comenzar, cabe aclarar que el código completo (en su versión final) se encuentra en la sección de Anexos de este reporte como Código Final y en esta sección se explicarán diferentes secciones y versiones del código.

Streaming de video desde la cámara del drone

Se comenzó por la implementación de *streaming* de video del drone a la PC. Para lograrlo, se realizó primeramente una búsqueda en internet, en donde se logró encontrar una explicación y el código adjunto en la sección de Anexos, como Código de Streaming, que permite visualizar la cámara del drone en la PC, mediante VLC. (McGovern, libVLC demo code for the Bebop, 2018)

Este código sirvió de base para continuar el proyecto. Se le realizaron las modificaciones necesarias como se detalla en los siguientes pasos.

Prueba de vuelo del drone:

Para verificar la conexión, calibración y vuelo correcto del drone, se utilizó el código llamado test1. Con él, se pueden realizar los procesos básicos del vuelo del drone. Este código, establece la altitud y la velocidad máximas de los movimientos del drone, despega el drone, verifica los sensores, mueve el drone y lo aterriza. Dicho código, puede ser encontrado en la sección de Anexos bajo el nombre test1.

Para agregar funciones deseadas, se pueden consultar todas las instrucciones para el drone en el Código de instrucciones, en la sección de Anexos. También puede consultarse en la siguiente liga: https://pyparrot.readthedocs.io/en/latest/bebopcommands.html. (McGovern, Bebop Commands and Sensors, 2018)

Integración del vuelo del drone con el código de streaming

Una vez que se tiene el *streaming* funcionando correctamente y se comprobó el vuelo del drone, pasamos a realizar un solo código y con él se pudo obtener el vuelo del drone y una imagen en vivo de la cámara.

Rotación de la cámara del drone

Para nuestra práctica, fue necesario rotar la cámara para que vea directamente hacia abajo (al suelo). Esto se logró utilizando el comando, encontrado en la lista de comandos presentada en la liga de la sección Prueba de vuelo del drone.

```
bebop.pan_tilt_camera(-90,0)
```

En donde los -90 representa un movimiento vertical para bajar la cámara (90°), para que vea el piso, y el cero representa un movimiento horizontal (de 0°), centrando la imagen.

Imagen 4. Descripción de ángulos de pan y tilt en el drone.

Convertir la imagen de RGB a HSV

Se convierte la imagen original obtenida del streaming del drone de formato RGB a formato HSV, con el objetivo de manejarla con más facilidad. Esto se logra mediante el comando:

```
HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

En donde: `HSV`, corresponde al nombre de la nueva imagen (ya en formato HSV) y `frame`, es la imagen original capturada por el drone.

Modelo de color HSV

Es un formato o modelo de color que define el color en términos de sus componentes. También, este modelo es conocido como HSB. (Fundación Wikimedia, Inc, 2019)

- H: del inglés *Hue*, es la matriz.
- S: del inglés *Saturation*, es la saturación del color.
- V: del inflés *Value*, es el valor del color dado.

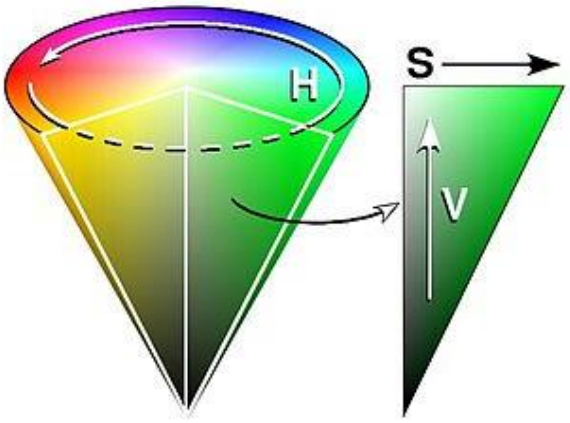

Imagen 5. Modelo de color HSV.

Definir el rango de color deseado para identificar el círculo

Se define un rango inferior y un rango superior de valores HSV. Estos, pueden ser fácilmente identificados con la ayuda del [Código para identificación de colores HSV], en la sección de Anexos. Gracias a este programa, se obtiene una imagen en vivo, a través de la cámara web de la PC, y una pestaña con *sliders* para seleccionar el valor de HSV máximo y mínimo y filtrarlo en la imagen de la cámara web. A continuación, se muestra una captura de pantalla del programa ejecutándose.

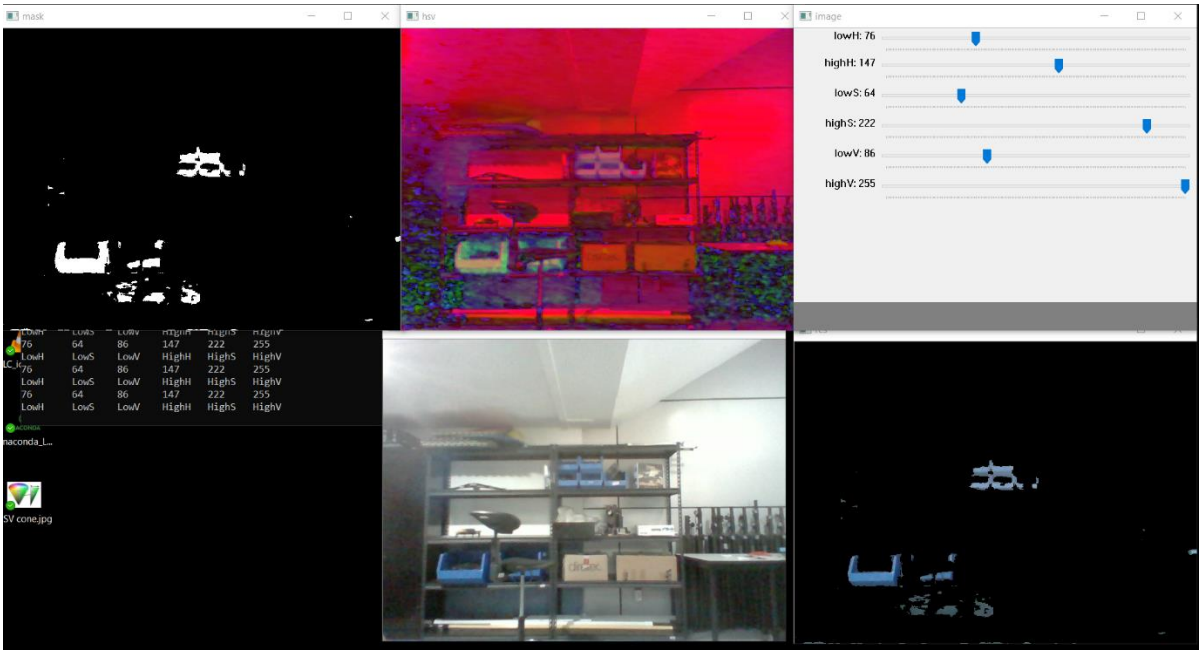

Imagen 6. Uso del programa SelectColor.py.

Finalmente, se añadieron las siguientes dos líneas de código al programa principal, declarando dos variables, para definir los límites del color deseado a reconocer por el drone.

```
lowerLimit = np.array([13, 40 , 120], dtype = np.uint8)
upperLimit = np.array([40, 150, 255], dtype = np.uint8)
```

En donde lowerLimit y upperLimit son las variables *unsigned int* de 8 bits que almacenarán un array con los valores dados.

Aplicación de filtros a la imagen original

Además de cambiar la imagen original al modelo HSV, se necesitan crear nuevas imágenes en donde se le aplican distintos filtros. Algunos, con el objetivo de una mejor visualización, mejor identificación, eliminación de ruido y transformación a escala de grises.

Threshold

Se aplica este filtro con la intención de eliminar el resto de los colores no deseados, dejando solamente visible el color escogido previamente. El comando utilizado es el siguiente:

```
Threshold = cv2.inRange(HSV, lowerLimit, upperLimit)
```

En donde: Threshold, es la nueva imagen generada; HSV es la imagen a modificar y después se le dan los límites deseados.

Bitwise AND

En este paso, se agrega este filtro para mezclar la imagen original con la imagen filtrada, presentando solamente el objeto deseado y resaltándolo. El comando utilizado es el siguiente:

```
filtered = cv2.bitwise_and(frame, frame, mask = Threshold)
```

Blur Filter

Este filtro nos ayuda a aplicar un efecto *blur*, con la intención de suavizar la imagen (y los bordes del circulo), permitiendo una detección con mayor facilidad. Se logra con el comando:

```
Blur = cv2.medianBlur(filtered,5)
```

Escala de grises

Finalmente, el último filtro aplicado es para convertir la imagen a escala de grises. Esto se realiza porque el proceso de detección de círculos con el método de Hough requiere trabajar con una imagen en escala de grises. El comando utilizado es:

```
grayImage = cv2.cvtColor(Blur, cv2.COLOR_BGR2GRAY)
```

Detección de círculos

Para esta parte del proyecto, se investigó la transformación de Hough. Se encontró diversa información, pero principalmente el proyecto se basó en dos páginas:

- Información: del comando: https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/.
- Ejemplo de detección de círculos: https://docs.opencv.org/3.1.0/da/d53/tutorial_py_houghcircles.html.

De donde podemos obtener la siguiente información del comando:

```
cv2.HoughCircles(image, method, dp, minDist)
```

En donde: image es la imagen para leer, el segundo parámetro representa el método de Hough a utilizar (solamente se puede utilizar `cv2.HOUGH_GRADIENT`), dp es un inverso de la ratio (a mayor dp menor acumulación en el arreglo), minDist es la distancia mínima a la que se detectará el círculo, a mayor distancia, círculos más grandes (entre más grande, menos ruido). (Rosebrock, 2014)

Este método realiza el procesamiento necesario mediante la transformada de Hough para detectar círculos, pero es necesario enfatizar que solamente trabaja con imágenes en escala de grises (por ello, fue necesario transformar la imagen a tonos de gris, como se explica en la sección Escala de grises).

La función guarda en una variable (matriz) la información de la imagen de entrada en donde se encuentra un círculo. De esta forma, se puede hacer la siguiente asignación en el código:

```
Circle = cv2.HoughCircles(grayImage,cv2.HOUGH_GRADIENT, 1, 150, param1 = 50,
param2=30, minRadius=0, maxRadius=0);
```

Para más información, favor de consultar las ligas escritas anteriormente.

Comprobación de la detección de círculos

Para comprobar que la función de detección de círculos los encuentra correctamente, implementamos una sencilla verificación. Con ello, podemos saber cuándo se detecta un círculo de las características dadas para, dentro de la condición, pasar la comprobación a una manera gráfica en la imagen, dibujado un círculo (como se explica en la siguiente sección).

```
if Circle is not None:
```

Dibujo del círculo encontrado

Este proceso es meramente una comprobación gráfica que será aplicada en la imagen en la escala de grises, para ser vista en tiempo real cuando un círculo es detectado.

Se utiliza la función básica:

```
cv.Circle(img, center, radius, color, thickness=1, lineType=8, shift=0)
```

Esto permite dibujar un circulo en la imagen, con las características que le demos:

- Img: imagen en la cuál se va a dibujar.
- Center: posición del centro del círculo a dibujar.
- Radius: radio del círculo a dibujar.
- Color: color círculo a dibujar. Se da en formato BGR (RGB). Por ejemplo, para obtener un círculo de color verde: `(0,255,0)`.
- Thickness: espesor del contorno del círculo, si es positivo. El grosor negativo significa que se debe dibujar un círculo relleno.
- lineType: tipo de contorno del círculo.
- Shift: número de bits fraccionarios en las coordenadas del centro y en el valor del radio.

En este proyecto se dibujan dos círculos, uno que resalte el contorno del círculo detectado (y que funciona a manera de comprobación) y otro que estará siempre en el centro de la imagen. El motivo es comparar el centro de ambos círculos y hacer una comparación de la posición (en pixeles) del centro de ambos círculos.

Para dibujar el contorno del círculo detectado se utiliza el siguiente comando. Los valores que se le asignan como centro son los obtenidos de la función Hough:

```
grayImage = cv2.circle(filtered,(i[0],i[1]),i[2],(0,0,255),3)
```

Para dibujar el círculo en el centro de la imagen, la posición será siempre la misma. Se le asigna 410 y 240 ya que es la mitad de los pixeles de la imagen (el drone tiene una calidad de 820x480).

```
grayImage = cv2.circle(filtered,(410,240),2,(0,0,255),3)
```

En ambos casos, se dibujan en dos imágenes al mismo tiempo, tanto en la que está en escala de grises como en la que se llama filtered.

Obtener la posición del círculo

Para determinar la posición del círculo, respecto al drone, se realiza la siguiente asignación:

```
Circle_Width  = i[0]
Circle_Height = i[1]
```

Estas dos variables contendrán el valor del primer píxel del círculo detectado, el cual podrá ser comparado con el valor de la posición del drone, que siempre será el centro de la imagen:

```
Drone_Height = 240
Drone_Width = 420
```

De esta forma, se logra determinar si el drone está más a la izquierda, derecha, adelante o atrás del círculo y hacer los movimientos para ajustar su posición.

Centrar el drone en el círculo

Como se explicó anteriormente, se cuenta con una comparación (en pixeles) entre la posición del drone y la posición del círculo detectado. Inmediatamente, se implementa lo siguiente:

```
if Circle_Width > (Drone_width+15):
bebop.fly_direct(6,0,0,0,2)
       bebop.smart_sleep(2)
if Circle_Width < (Drone_width-15):
bebop.fly_direct(-6,0,0,0,2)
       bebop.smart_sleep(2)
if Circle_Height > (Drone_height+15):
       bebop.fly_direct(0,6,0,0,2)
       bebop.smart_sleep(2)
if Circle_Height < (Drone_height-15):
       bebop.fly_direct(0,-6,0,0,2)
       bebop.smart_sleep(2)
elif (Circle_Height < (Drone_height+50)) or (Circle_Height > (Drone_height-50))
or (Circle_Width < (Drone_width+50)) or (Circle_Width > (Drone_width-50)):
       #bebop.fly_direct(-5,13,0,0,2)
       bebop.safe_land(8)
```

Estas validaciones comprobarán y moverán el drone para corregir su posición, respecto a la del círculo. Por ejemplo, si el drone está a la izquierda del círculo, el drone se moverá a la derecha.

Movimiento del drone

Para mover el drone se utiliza el siguiente comando:

```
fly_direct(roll, pitch, yaw, vertical_movement, duration)
```

Este comando hace que el drone rote sobre los ejes roll, pitch y yaw, dependiendo de los grados que le demos en el comando. Al mismo tiempo, `vertical_movement` corresponde a un movimiento vertical, en donde al darle un valor positivo aumentará la altura y viceversa. Finalmente, `duration` corresponde al tiempo en el que el comando estará en ejecución; es decir, durante cuánto tiempo el drone estará moviéndose en esas direcciones. (V, Kadamatt, 2017)

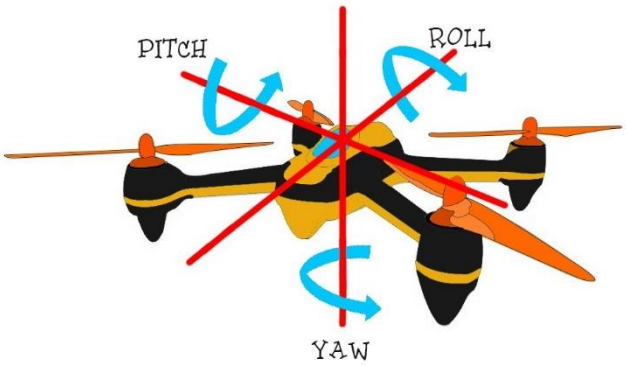

Imagen 7. Ejes de movimiento del drone.

Mostrar las imágenes obtenidas

En la sección de Aplicación de filtros se puede ver que, además de la imagen original, se trabaja con varias imágenes elaboradas. Con el siguiente comando, será posible desplegarlas y hacerlas visibles al usuario:

```
cv2.imshow('Name of the frame', Image_to_show)
```

Basta con ingresar el nombre con el que llamaremos y será mostrada la imagen, así como seleccionar la imagen deseada. En el caso de este proyecto, únicamente se despliegan dos imágenes: la imagen original y la imagen final a escala de grises (sobre la cual también se dibuja el círculo encontrado); se utilizan los siguientes comandos:

```
cv2.imshow('Original', frame)
cv2.imshow('Gray Scale', grayImage)
```

Tecla de seguridad

Finalmente, pero no menos importante, se asignó una tecla de seguridad. Esta función permite que, al presionar una tecla específica del teclado, se detengan todas las funciones del drone, aterrice y se cancele el programa.

Existen dos formas de implementarla, la primera es utilizando la librería del teclado y escribir la siguiente condición:

```
if keyboard.is_pressed('Esc'):
break
```

El segundo método, que logra lo mismo, es comparando el número del carácter ASCII que se recibe del teclado. Por ejemplo, para la misma tecla de escape (‘Esc’) se deberá de recibir el valor 27 (acorde al ASCII). Para más información del valor en ASCII, consultar la Tabla ASCII, en la sección de Anexos. A continuación, el código para implementar la tecla de seguridad con este método:

```
k = cv2.waitKey(5) & 0xFF
if k == 27:
break
```

Finalmente, se decidió combinar ambos métodos, a manera de doble verificación. No importando que se tenga una redundancia, por tratarse de un tema de seguridad. Por lo tanto, la implementación queda de la siguiente forma:

```
k = cv2.waitKey(5) & 0xFF
if k == 27 or k == 20 or keyboard.is_pressed('Esc'):
break
```

Resultados

Como resultados del proyecto, se obtuvo el código final (presente en la sección de Anexos) y un video demostrativo del funcionamiento de este, que se encuentra en la siguiente liga: http://youtu.be/ueUS8aOvBVs.

Cómo implementar el proyecto

Primeramente, asegurarse de conectarse al drone vía wifi. Posteriormente, abrir Anaconda Prompt (de preferencia, ejecutar como administrador). Ya en el Prompt, cambiar de directorio con el comando `cd` a la ruta (o *path*) en donde se encuentra el archivo del código. Finalmente, ejecutándolo con el comando: `python test1.py`; en donde se debe de dar el nombre del archivo y su extensión “`.py`”.

Principales obstáculos

La primera dificultad encontrada fue que el código de *streaming* no permitía de forma correcta la conexión con el drone. Se mostraban códigos de error. La solución al problema fue dar los permisos del Firewall de Windows. Si el problema persiste, desactivar la protección en tiempo real.

La segunda dificultad se presentó al momento de leer la imagen tomada desde el drone. La solución fue otorgar los permisos de acceso a la carpeta de Anaconda3, en donde se guarda la imagen con la ruta seleccionada en el código. Si los problemas persisten, cambiar la ruta y **ejecutar Anaconda Prompt como administrador**.

La tercera dificultad presentada, también al leer la imagen, pero por parte de VLC se presentaba únicamente en circunstancias especiales y puede ser resuelto yendo a la configuración de VLC, luego al menú Entrada/Códecs y desactivar la decodificación acelerada por hardware. Recalcamos que se debe de instalar la **versión de 64 bits**.

La cuarta dificultad se presentó al momento de la detección de círculos, ya que al aplicar la función se detectaban cientos de círculos en la imagen. La solución a esto es modificar los valores de la distancia mínima y el radio mínimo y máximo (ver sección Detección de círculos).

La quinta dificultad se presento al momento de mover el drone. Ya que al drone se le dan indicaciones de inclinación en los ejes *roll*, *pitch* y *yaw* durante un determinado tiempo (ver Movimiento del drone) y, por lo tanto, este tiempo a veces se veía modificado, dependiendo de factores externos como el viento.

Finalmente, otra dificultad que depende de factores externos, de iluminación en este caso, es la asignación del valor HSV del color que se desea filtrar, como se describe en la sección *Threshold*. La solución a este problema fue cambiar el valor, de acuerdo con las condiciones del día; este proceso se puede apoyar con el Código para identificación de colores HSV, de la sección de Anexos.

Anexos

Código de Instrucciones para el drone

```
# -*- coding: utf-8 -*-
"""
SET DE INSTRUCCIONES BEBOP Y BEBOP2

CREAR UN OBJETO:

Bebop(drone_type="Bebop") / Bebop(drone_type="Bebop2")

CONECTAR CON EL DRONE VIA WIFI:

connect(num_retries) / ESPECIFICA EL NUMERO DE INTENTOS DE ESTABLECER
COMUNUICACIÓN CON EL DRONE ANTES DE MANDAR UN ERROR

disconnect() / DESCONECTA LA COMUNICACIÓN CON EL DRONE

DESPEGUE Y ATERRIZAJE:

safe_takeoff(timeout) / DESPEGA EL DRONE Y VERIFICA QUE SE HAYA REALIZADO.
ESTABLECE UN TIMEOUT QUE EN CASO DE SOBREPASAR CANCELA LA OPERACIÓN

safe_land(timeout) / ATERRIZA EL DRONE Y VERIFICA QUE SE HAYA REALIZADO.
ESTABLECE UN TIMEOUT QUE EN CASO DE SOBREPASAR CANCELA LA OPERACIÓN

MOVIMIENTO:

fly_direct(roll, pitch, yaw, vertical_movement, duration) / MUEVE EL DRONE SEGUN
LO INDICADO EN ROLL, PITCH, YAW, VERTICAL_MOVEMENT Y DURATION.
                                                          LOS PRIMEROS 4
PARAMETROS SE INTRODUCEN EN PORCENTAJE DE SU MAXIMA VELOCIDAD desde -100 hasta
100
                                                          Y LA DURACIÓN DEL
MOVIMIENTO EN SEGUNDOS

CONFIGURACIÓN:

set_max_altitude(altitude)  / ESTABLECE LA ALTITUD MÁXIMA PERMITIDA PARA EL DRONE
EN VALORES DESDE 0.5 Y HASTA 150 METROS

set_max_tilt(tilt)   / ESTABLECE LA INCLINACIÓN MÁXIMA PARA LOS MOVIMIENTOS DE
ROLL, PITCH, YAW. SE MIDE EN GRADOS Y VA DESDE
                        5(MOVIMIENTO LENTO) HASTA 30 (MUY RÁPIDO)

set_max_vertical_speed(speed)  / ESTABLECE LA VELOCIDAD MÁXIMA VERTICAL EN
METROS/SEGUNDO VA DESDE 0.5 HASTA 2.5

REVISAR LOS SENSORES DEL DRONE:

ask_for_state_update()  / ENVÍA LA SOLICITUD AL DRONE PARA QUE DEVUELVA LA
INFORMACIÓN ACTUALIZADA DE LOS SENSORES

bebop.sensors.SENSOR   / DEVUELVE LA INFORMACIÓN DEL SENSOR INDICADO
POR EJEMPLO:
bebop.sensors.flying_state
bebop.sensors.battery


"""
from pyparrot.Bebop import Bebop #Carga la libreria del drone
```

```
bebop = Bebop(drone_type="Bebop") #Se crea el objeto bebop

print("connecting")
success = bebop.connect(10) #Verifica la conexión
print(success)

# No cambiar los valores de las siguientes 3 configuraciones !!!!!!
bebop.set_max_altitude(2)           #Establece la altitud máxima en 3 metros
bebop.set_max_tilt(5)               #Establece la velocidad máxima de movimientos
angulares en 5°
bebop.set_max_vertical_speed(0.5)   #Establece la velocidad máxima vertical en
0.5 m/s

if (success):
    print("sleeping")
    bebop.smart_sleep(4)            #Realiza una pausa en el programa
    bebop.safe_takeoff(10)          #Despega el drone
    bebop.ask_for_state_update()    #Actualiza la información de los sensores
    print("flying state is %s" % bebop.sensors.flying_state) #imprime el estado
del drone
    bebop.smart_sleep(4)            #El drone se mantiene 4 segundos en el aire
    bebop.safe_land(10)             #Aterriza el drone
    bebop.ask_for_state_update()    #Actualiza la información de los sensores
    print("flying state is %s" % bebop.sensors.flying_state) #imprime el estado
del drone
    print("Battery is %s" % bebop.sensors.battery)  #imprime el estado de la
bateria
    print("DONE - disconnecting")
    bebop.disconnect()              #Termina la conexión con el drone
```

Código de Streaming

```
"""
Demo of the Bebop vision using DroneVisionGUI (relies on libVLC).  It is a
different
multi-threaded approach than DroneVision

Author: Amy McGovern
"""
from pyparrot.Bebop import Bebop
from pyparrot.DroneVisionGUI import DroneVisionGUI
import threading
import cv2
import time

isAlive = False

class UserVision:
    def __init__(self, vision):
        self.index = 0
        self.vision = vision

    def save_pictures(self, args):
        #print("saving picture")
        img = self.vision.get_latest_valid_picture()

        if (img is not None):
            filename = "test_image_%06d.png" % self.index
            #cv2.imwrite(filename, img)
            self.index +=1


def demo_user_code_after_vision_opened(bebopVision, args):
    bebop = args[0]

    print("Vision successfully started!")
    #removed the user call to this function (it now happens in open_video())
    #bebopVision.start_video_buffering()

    # takeoff
    bebop.safe_takeoff(5)

    # skipping actually flying for safety purposes indoors - if you want
    # different pictures, move the bebop around by hand
    print("Fly me around by hand!")
    bebop.smart_sleep(5)

    if (bebopVision.vision_running):
        print("Moving the camera using velocity")
        bebop.pan_tilt_camera_velocity(pan_velocity=0, tilt_velocity=-2,
duration=4)
        bebop.smart_sleep(5)

        # land
        bebop.safe_land(5)

        print("Finishing demo and stopping vision")
        bebopVision.close_video()

    # disconnect nicely so we don't need a reboot
    print("disconnecting")
    bebop.disconnect()
```

```
if __name__ == "__main__":
    # make my bebop object
    bebop = Bebop()

    # connect to the bebop
    success = bebop.connect(5)

    if (success):
        # start up the video
        bebopVision = DroneVisionGUI(bebop, is_bebop=True,
user_code_to_run=demo_user_code_after_vision_opened,
                                     user_args=(bebop, ))

        userVision = UserVision(bebopVision)
        bebopVision.set_user_callback_function(userVision.save_pictures,
user_callback_args=None)
        bebopVision.open_video()

    else:
        print("Error connecting to bebop.  Retry")
```

Código test1: volar el drone

```
import keyboard #Using module keyboard
from pyparrot.Bebop import Bebop

bebop = Bebop(drone_type="Bebop2")                                  #Se crea el
objeto bebop

print("connecting")
success = bebop.connect(10)
print(success)
# No cambiar los valores de las siguientes 3 configuraciones !!!!!!
bebop.set_max_altitude(2)                                           #Establece la
altitud máxima en 2 metros
bebop.set_max_tilt(5)                                               #Establece la
velocidad máxima de movimientos angulares en 5°
bebop.set_max_vertical_speed(0.5)                                   #Establece la
velocidad máxima vertical en 0.5 m/s

if (success):
    print("sleeping")
    bebop.smart_sleep(4)                                            #Realiza una
pausa en el programa
    bebop.safe_takeoff(10)                                          #Despega el drone
    bebop.ask_for_state_update()                                    #Actualiza la
información de los sensores
    print("flying state is %s" % bebop.sensors.flying_state)        #imprime el
estado del drone
    bebop.smart_sleep(4)                                            #El drone se
mantiene 4 segundos en el aire
    bebop.fly_direct(0, 0, 100, 0, 5)                               # MUEVE EL DRONE
(%roll, %pitch, %yaw, vertical_movement, duration)
    bebop.smart_sleep(4)                                            #El drone se
mantiene 4 segundos en el aire
    bebop.safe_land(10)                                             #Aterriza el
drone
    bebop.ask_for_state_update()                                    #Actualiza la
información de los sensores
    print("flying state is %s" % bebop.sensors.flying_state)        #imprime el
estado del drone
    print("Battery is %s" % bebop.sensors.battery)                  #imprime el
estado de la bateria
    print("DONE - disconnecting")
    bebop.disconnect()                                              #Termina la
conexión con el drone
```

Código para identificación de colores HSV

```
# RGB to HSV conversion, using sliders to define
# the region of interest to select pixels
# RickWare
# May 3, 2019
import cv2
import numpy as np

def callback(x):
    pass

cap = cv2.VideoCapture(0)
cv2.namedWindow('image')

# Initial track bar (sliders) limits
ilowH = 0       # low Hue value
ihighH = 255    # High Hue value

ilowS = 0       # low Saturartion value
ihighS = 255    # High Saturation value

ilowV = 0       # low Value value
ihighV = 255    # High Value value

# Create trackbars for color change
cv2.createTrackbar('lowH','image',ilowH,255,callback)
cv2.createTrackbar('highH','image',ihighH,255,callback)

cv2.createTrackbar('lowS','image',ilowS,255,callback)
cv2.createTrackbar('highS','image',ihighS,255,callback)

cv2.createTrackbar('lowV','image',ilowV,255,callback)
cv2.createTrackbar('highV','image',ihighV,255,callback)



while(1):
    # Get trackbar positions
    ilowH = cv2.getTrackbarPos('lowH', 'image')
    ihighH = cv2.getTrackbarPos('highH', 'image')
    ilowS = cv2.getTrackbarPos('lowS', 'image')
    ihighS = cv2.getTrackbarPos('highS', 'image')
    ilowV = cv2.getTrackbarPos('lowV', 'image')
    ihighV = cv2.getTrackbarPos('highV', 'image')
    # Read camera frame
    ret, frame = cap.read()
    # Convert frame from RGB to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Display image in HSV color space
    cv2.imshow('hsv', hsv)
    # Define region that matches pixel color criteria
    lower_hsv = np.array([ilowH, ilowS, ilowV])
    higher_hsv = np.array([ihighH, ihighS, ihighV])
    # Define region to create mask
    mask = cv2.inRange(hsv, lower_hsv, higher_hsv)
    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)
    # Display mask, frame and resultant frame
    cv2.imshow('mask', mask)
    cv2.imshow('frame', frame)
    cv2.imshow('res',res)
    print ('LowH\t', 'LowS\t', 'LowV\t', 'HighH\t', 'HighS\t', 'HighV\t')
```

```
    print (ilowH, '\t', ilowS, '\t', ilowV, '\t', ihighH, '\t', ihighS, '\t',
ihighV)
    # Check if "q" key has been pressed to exir program
    if(cv2.waitKey(1) & 0xFF == ord('q')):
        break

# Program ends with releasing camera and closing all windows
cv2.destroyAllWindows()
cap.release()
```

Código Final

```
import cv2
import cv2
import time
import numpy as np
import threading
from pyparrot.Bebop import Bebop
from pyparrot.DroneVisionGUI import DroneVisionGUI

isAlive = False

class UserVision:
    def __init__(self, vision):
        self.index = 0
        self.vision = vision

    def save_pictures(self, args):
        # Saving picture:
        img = self.vision.get_latest_valid_picture()

        if (img is not None):
            filename = "test_image_%06d.png" % self.index           # The name of
the picture taked by the drone, to be saved
            #cv2.imwrite(filename, img)                             #
Uncommenting this line, will save all the image in the file's folder. CAUTION!!!
            self.index +=1


def demo_user_code_after_vision_opened(bebopVision, args):
    bebop = args[0]

    print("Vision successfully started!")
    print("Battery is %s" % bebop.sensors.battery)                  # Write the
porcentage of battery when starting the flight.
    while(True):
        try:
            # Variable name = Command to read the image saved before (on the
default folder):
            frame = cv2.imread(r'C:\ProgramData\Anaconda3\Lib\site-
packages\pyparrot\images\visionStream.jpg')

            # Convert BGR to HSV:
            HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

            # Define range of desired color in HSV:
            lowerLimit = np.array([0, 65 , 192], dtype = np.uint8)     # Defined
as unsigned int of 8 bits. Order: H, S, V.
            upperLimit = np.array([34, 255, 255], dtype = np.uint8)     # Yellow
color selected.


            # Threshold the HSV image to get only the selected color:
            Threshold = cv2.inRange(HSV, lowerLimit, upperLimit)

            # Bitwise-AND:
            filtered = cv2.bitwise_and(frame, frame, mask = Threshold)     # This
will combine the original image and the filter color selected in one image.

            # Blur Filter:
            Blur = cv2.medianBlur(filtered,5)                       # Making a
Blur in order to reducing noise.
```

```
            # Convert to Gray scale:
            grayImage = cv2.cvtColor(Blur, cv2.COLOR_BGR2GRAY)

            # Gonig forward:
            bebop.smart_sleep(3)                                        # Delay.
            bebop.fly_direct(0,30,0,0,2)                                # Going
fordward.

            # Circle detection, using Hough function:
            print("Looking for the correct Circle...")
            Circle = cv2.HoughCircles(grayImage,cv2.HOUGH_GRADIENT, 1, 150,
param1 = 50, param2=30, minRadius=0, maxRadius=0);

            # Verify if a Circle is detected and Draw it:
            if Circle is not None:
                print("Circle Detected!")
                Circle = np.uint16(np.around(Circle))
                for i in Circle[0,:]:
                    # Draw the circle:
                    grayImage = cv2.circle(filtered,(i[0],i[1]),i[2],(0,0,255),3)
# Draw the outer of the Circle.
                    grayImage = cv2.circle(filtered,(410,240),2,(0,0,255),3)
# Draw the center of the Image.

                    # Get the circle position:
                    Circle_Width  = i[0]
# Get the position of the Drone.
                    Circle_Height = i[1]
# Get the position of the Drone.

                    # Define the drone position, always is the center of the
image:
                    Drone_height = 240
                    Drone_width = 420

                # Drone movement to correct the position and go to the center of
the circle:
                print("Going to the center...")
                # Check if is displaced:
                if Circle_Width > (Drone_width+15):                 # Check if
the drone is on the left of the circle.
                    bebop.fly_direct(6,0,0,0,2)                     # Move to the
right.
                    bebop.smart_sleep(2)                            # Delay.
                if Circle_Width < (Drone_width-15):
                    bebop.fly_direct(-6,0,0,0,2)
                    bebop.smart_sleep(2)
                if Circle_Height > (Drone_height+15):
                    bebop.fly_direct(0,6,0,0,2)
                    bebop.smart_sleep(2)
                if Circle_Height < (Drone_height-15):
                    bebop.fly_direct(0,-6,0,0,2)
                    bebop.smart_sleep(2)
                # Check if now is cetered:
                elif (Circle_Height < (Drone_height+50)) or (Circle_Height >
(Drone_height-50)) or (Circle_Width < (Drone_width+50)) or (Circle_Width >
(Drone_width-50)):
                    #bebop.fly_direct(-5,13,0,0,2)                  # Correction
for the air.
                    bebop.safe_land(8)                              # Land the
Drone.
                    print("Center finded. Landing...")
```

```
            # Display the images:
            cv2.imshow('Original', frame)                               # Original
image.
            cv2.imshow('Gray Scale', grayImage)                         # GrayScale
image.
            #cv2.imshow('Filtered', filtered)                            # Only the
selected color (filtered) image.

            # Safety key!
            k = cv2.waitKey(5) & 0xFF                                   # Press 'Esc'
to break and land the drone.
            if k == 27 or k == 20 or keyboard.is_pressed('Esc'):
                cv2.destroyAllWindows()
                break

        except:
            pass

    print("Battery is %s" % bebop.sensors.battery)                      # Write the
porcentage of battery at the end of the flight.

    # Land the drone:
    bebop.safe_land(10)
    print("Drone landed")

    #bebop.ask_for_state_update()                                        # Update
sensors information.
    #print("flying state is %s" % bebop.sensors.flying_state)            # Print the
drone status.

    # Finish the connection to the drone:
    bebop.disconnect()
    print("DONE - disconnected")

if __name__ == "__main__":
    # Make the bebop object:
    bebop = Bebop(drone_type="Bebop2")                                  # Write the
Bebop number, in this case is bebop2.

    # Connect to the bebop:
    print("Connecting to bebop...")
    success = bebop.connect(5)                                          # Verify the
drone connection.

    if (success):
        Done = 0
        print("Connection succesfull.")
        bebop.smart_sleep(4)                                            # Delay to
ensure that the drone receives the commands well.

        # Set the limits for the drone:
        bebop.set_max_altitude(2)                                       # Set the
maximum altitude in 2 meters.
        bebop.set_max_tilt(5)                                           # Sets the
maximum speed of angular movements in 5°.
        bebop.set_max_vertical_speed(0.5)                               # Sets the
maximum vertical speed at 0.5 m/s.

        bebop.safe_takeoff(10)                                          # Drone take
off.
        bebop.smart_sleep(3)                                            # Delay to
stabilize the drone in the air.
```

```
        bebop.fly_direct(0,0,0,25,3)
        bebop.smart_sleep(3)


        # Check:
        #print("Checking sensors...")
        #bebop.ask_for_state_update()                                    # Update
sensors information.
        #print("Flying state is %s" % bebop.sensors.flying_state)    # Print the
drone status.
        #bebop.smart_sleep(4)                                           # Delay to
stabilize the drone in the air.

        # Move the camera:
        bebop.pan_tilt_camera(-90,0)                                  # Rotate the
camera (-90° means downside and 0° means in front)
        bebop.smart_sleep(3)                                           # Delay

        # Start up the video:
        bebopVision = DroneVisionGUI(bebop, is_bebop=True,
user_code_to_run=demo_user_code_after_vision_opened,
                                     user_args=(bebop, ))

        userVision = UserVision(bebopVision)
        bebopVision.set_user_callback_function(userVision.save_pictures,
user_callback_args=None)
        bebopVision.open_video()

        bebop.smart_sleep(3)                                           # Delay

    else:
        print("Error connecting to bebop.  Retry")                  # Print only
if an error happends.
```

Tabla ASCII

A continuación, se adjunta una tabla ASCII, con el objetivo de apoyar a la sección Tecla de seguridad y poder escoger cualquier tecla. (El código ASCII, 2019)

El código ASCII

sigla en inglés de American Standard Code for Information Interchange
(Código Estadounidense Estándar para el Intercambio de Información)

www.elcodigoascii.com.ar

Caracteres de control ASCII

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de línea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

elCodigoASCII.com.ar

ASCII extendido

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Imagen 8. Código ASCII.

Para más información, consultar la siguiente página: https://elcodigoascii.com.ar/.

Referencias

Anaconda, Inc. (2019). *Installing on Windows*. Retrieved from Anaconda Documentation: https://docs.anaconda.com/anaconda/install/windows/

El código ASCII. (2019). Retrieved from Código Estadounidense Estándar para el Intercambio de Información: https://elcodigoascii.com.ar/

Fundación Wikimedia, Inc. (2019, mayo 14). *Modelo de color HSV*. Retrieved from Wikipedia: https://es.wikipedia.org/wiki/Modelo_de_color_HSV

McGovern, A. (2018). *Bebop Commands and Sensors*. Retrieved from Pyparrot - Read the Docs: https://pyparrot.readthedocs.io/en/latest/bebopcommands.html

McGovern, A. (2018). *Installation*. Retrieved from Pyparrot - Read the Docs: https://pyparrot.readthedocs.io/en/latest/installation.html

McGovern, A. (2018). *libVLC demo code for the Bebop*. Retrieved from Pyparrot - Read the Docs: https://pyparrot.readthedocs.io/en/latest/vision.html#libvlc-demo-code-for-the-bebop

Rosebrock, A. (2014, julio 21). *Detecting Circles in Images using OpenCV and Hough Circles*. Retrieved from Py Image Search: https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/

V, Kadamatt. (2017, febrero 28). *How to Fly a Quadcopter: Eightfold Missions to Mastery*. Retrieved from Droney Bee: http://www.droneybee.com/how-to-fly-a-quadcopter/

VideoLAN. (2019). *VLC Features*. Retrieved from VideoLAN: https://www.videolan.org/vlc/features.html