

|  |  |
| --- | --- |
| ROBÓTICA  **TPL-1**  Visión Artificial | 4 de Mayo  2021 |

Profesor: Ing. Salariato, Ignacio

Jefe de Laboratorio: Ing. Suenaga, Gustavo

Alumnos: Napolitano, Gabriel

Comisión: Martes, Noche

|  |  |
| --- | --- |
| **LABORATORIO DE ELECTRÓNICA** | **ASIGNATURA: ROBÓTICA** |

|  |  |  |
| --- | --- | --- |
| **T.P.Nº 1** | **TÍTULO DEL TRABAJO PRÁCTICO:**  **Utilización de Python y OpenCV para Visión  Artificial** | **Código de Asignatura:**  **I0510** |
| **Horas de formación práctica asignada: 4** |

**OBJETIVOS:** Poner en práctica los conocimientos vistos en la teoría de visión artificial mediante distintos programas con distintas funcionalidades.

**OBJETIVO GENERAL:** Realizar programas que permitan desarrollar una cierta lógica o inteligencia en base al análisis de imágenes capturadas.

**OBJETIVOS ESPECÍFICOS:** Utilizar la visión artificial para el reconomiento y diferenciamiento de objetos y patrones.

**MARCO TEÓRICO Y CONOCIMIENTOS PREVIOS**

OpenCV es una librería que integra múltiples funcionalidades y herramientas que permiten facilitar el desarrollo de aplicaciones de visión artificial, análisis y procesamiento de imágenes, dibujo, entre otras.

La misma integra desde operaciones simples como el recorte de una zona de la imagen u obtener la información de los colores que conforman de cada pixel y/o la modificación de los mismos hasta funcionalidades realmente complejas como la detección de patrones, contornos, entre otras.

Es común su uso en conjunción con la librería Numpy, la cual permite trabajar con facilidad con arrays de múltiples dimensiones, lo que simplifica el procesamiento de toda la información contenida en una imagen.

Dentro de las múltiples posibilidades de aplicación que facilita la librería OpenCV se encuentran:

La umbralización, la cual permite dividir en una imagen las regiones que cumplan la condición de tener una intensidad mayor al umbral definido y por otro lado las regiones de menor intensidad.

La búsqueda de contornos con lo cual se simplifica la delimitación de los objetos que se encuentran en la imagen.

El cálculo de histogramas, herramienta que da información muy detallada sobre los niveles de cada color que se encuentran presentes en la imagen.

**EQUIPOS Y DIAGRAMA**

|  |  |  |
| --- | --- | --- |
| **Equipo requerido** | **Versión (utilizada en desarrollo)** | **Observaciones** |
| Python | 3.9.4 | Intérprete del lenguaje. |
| OpenCV | 4.5.2 | Módulo externo (debe descargarse) |
| Numpy | 1.20.2 | Módulo externo (debe descargarse) |
| Matplotlib | 3.4.1 | Módulo externo (debe descargarse) |

El código fuente de toda la práctica, junto con los archivos necesarios para su ejecución pueden descargarse de:

<https://drive.google.com/drive/folders/11tYTVru4zOtFjzxblXjnBzRgCi5VRx94?usp=sharing>

(Si se solicita permiso para acceder escribir a: [gabriel.napolitano@alumnos.udemm.edu.ar](mailto:gabriel.napolitano@alumnos.udemm.edu.ar))

O mediante la plataforma GitHub (se encontrará disponible al público hasta el día 11/05/2021, y luego se convertirá en un repositorio privado):

git clone <https://github.com/Naposprograms/Robotics_Lab_1>

**PROCEDIMIENTO**

Se desarrolló un programa completo que permite ejecutar todos los puntos del procedimiento en una única instancia. A su vez, cada uno de estos pasos puede probarse individualmente. Para ello se hace uso de una función denominada run\_program() en cada uno de los módulos. Cuando se llama esta función desde un script externo que haya importado el módulo, se ejecuta el programa del módulo. Para probar cada módulo individualmente se utiliza la siguiente sintaxis, incluída dentro de cada módulo:

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

El programa que ejecuta el procedimiento completo (TP\_Full.py), resulta entonces, en un simple llamado a run\_program() de cada parte individual:

import Part\_1

import Part\_2

import Part\_3

import Part\_4

import Part\_5

import Part\_6

import Part\_7

Part\_1.run\_program()

Part\_2.run\_program()

Part\_3.run\_program()

Part\_4.run\_program()

Part\_5.run\_program()

Part\_6.run\_program("Select\_Template")

Part\_6.run\_program("Match\_Template")

Part\_7.run\_program()

**Aclaraciones previas a la ejecución de los programas en forma individual:**

Si el programa Part\_1.py no se ha ejecutado aún, el programa Part\_4.py fallará debido a que el mismo utiliza la imagen que se captura y guarda durante la ejecución de Part\_1.py

Para el correcto funcionamiento del programa Part\_5.py se debe localizar la cámara en un ambiente con muy buena iluminación, ya que si la iluminación es mala, muchos de los píxeles del objeto verde que se desee detectar caerán bajo el umbral de grises, y el programa no los considerará verdes.

El programa Part\_6.py debe recibir un argumento. Las opciones posibles son:

python Part\_6.py Select\_Template

python Part\_6.py Match\_Template

Para su ejecución en forma de módulo dicho argumento se recibe como el parámetro mode de la función run\_program(mode).

Para ejecutar el programa en modo Match\_Template, se debe haber ejecutado previamente en modo Select\_Template, ya que en caso contrario el modo Match no tendrá una imagen patrón que buscar.

Los programas Part\_2.py, Part\_3.py y Part\_7.py fueron desarrollados para buscar archivos dentro del directorio “Images”, situado en el mismo directorio que el programa. Para su correcto funcionamiento se debe respetar este directorio, así como los nombres de los archivos que se encuentren dentro de “Images”.

A continuación, se detallan algunas funcionalidades utilizadas en varios de los programas:

Módulo os y variable path (y variantes):

Se consulta el directorio desde el cual se está ejecutando el programa y se guarda en la variable path. Luego se genera una nueva variable con el path en el cual se guardará(n) la(s) imagen(es) que se capturará(n) o el path en el cual se encuentran la(s) imagen(es) a leer para utilizar en el programa.

Esta funcionalidad es utilizada en los programas Part\_1.py y Part\_6.py para guardar imágenes capturadas y en los programas Part\_2.py, Part\_3.py, Part\_4.py para leer imágenes guardardas, y en el programa Part\_7.py para leer un video guardado.

Variable check\_image\_type:

Se utiliza en los programas que leen una imagen guardada de un directorio, para verificar que el objeto leído sea de tipo imagen, y no un NoneType, lo cual indicaría que hubo un problema con el path que se utilizó para leer el archivo.

Se verifica con un if que la lectura no devuelva NoneType, y si se da dicho caso, se indica que se debe verificar el directorio provisto y se concluye la ejecución del programa.

Esta funcionalidad es utilizada en los programas Part\_2.py, Part\_3.py, Part\_4.py y Part\_6.py

Variable capture\_ok:

Se utiliza en los programas que utilizan la cámara, para verificar que la captura sea válida.

El programa crea un objeto con el cual se referencia a la cámara detectada en el índice 0 con la función VideoCapture(). Luego se toma una fotografía con la cámara y se la almacena en la variable image. El booleano capture\_ok nos permite saber si la captura fue correcta. Si la función read() devuelve un False para esta variable, significa que hay algún problema con la cámara. Esta situación se evalúa en un bucle if que termina la ejecución del programa si la cámara falla por algún motivo y no puede tomar fotogramas válidos.

Se utiliza en los programas Part\_1.py, Part\_5.py, Part\_6.py.

Se utiliza una variante en el programa Part\_7.py, pero en lugar de verificar que sea una captura válida por parte de la cámara, se verifica que se lea un fotograma válido del archivo de video. Cuando no se reciben más fotogramas válidos es porque el video terminó, y aquí en lugar de finalizar la ejecución del programa, se utiliza ‘break’ para romper el bucle while.

**Parte 1:**

Se desarrolló el programa Part\_1.py. El mismo utiliza los módulos: **cv2** y **os.**

**Al ejecutar la función run\_program():**

El programa toma una fotografía con la cámara y guarda el archivo. Luego mediante el método shape del objeto image se obtiene el tamaño de la imagen y la cantidad de canales y se almacena cada uno en una variable. El programa luego divide el tamaño a la mitad tanto para el eje X como para el eje Y, para así hallar el pixel central de la imagen. Luego mediante el método itemset del objeto imagen se establece el valor de intensidad 0 para los canales B y G, y el valor 255 para el canal R, volviendo el pixel de color rojo puro.

Finalmente se establecen los límites de la región de interés (ROI) a una distancia de 50 píxeles hacia ambos costados y 25 píxeles hacia arriba y hacia abajo del pixel central. Se crea un objeto image\_ROI que será la parte de la imagen original contenida dentro de los límites establecidos, y se muestra al usuario con el método imshow.

**Parte 2:**

Se desarrolló el programa Part\_2.py. El mismo utiliza los módulos: **cv2**, **numpy** y **os.**

**Al ejecutar la función run\_program():**

El programa lee la imagen "/Images/cuadrado.jpg" y crea un objeto image\_gray a partir de la imagen original mediante el método cvtColor. Este objeto se convierte luego al tipo float32 definido por el módulo Numpy, y este objeto se utiliza como parámetro para el método cornerHarris. El método cornerHarris halla las esquinas en la imagen mediante un algoritmo que busca diferenciales grandes en la intensidad de los pixeles cercanos, y devuelve un objeto imagen en el cual se marcan las esquinas detectadas.

El programa luego crea variables que tendrán las dimensiones de dicha imagen y mediante un bucle doble que recorre pixel a pixel en forma horizontal y luego vertical cuando encuentra un pixel perteneciente a una esquina en el objeto dst dibuja en la imagen original en esa misma posición un círculo color verde con el método circle.

Finalmente, mediante el método imshow se muestran la imagen original con las esquinas resaltadas en verde, y la imagen que contiene la información de dónde se encontraron las esquinas.

**Parte 3:**

Se desarrolló el programa Part\_3.py. El mismo utiliza los módulos: **cv2** y **os.**

**Al ejecutar la función run\_program():**

El programa lee la imagen "/Images/herramientas.jpg" y crea un objeto image\_gray a partir de la imagen original. Se creó la variable user\_defined\_threshold con un valor intermedio. Luego mediante el método threshold se realizó la umbralización de la imagen utilizando user\_defined\_threshold como parámetro y se almacenó el resultado en el objeto dst. Visualizándolo mediante imshow.se ajustó manualmente el valor de la variable user\_defined\_threshold hasta llegar a un valor adecuado en el cual se distingue con claridad el contorno de las herramientas de la imagen.

Posteriormente se utilizó el método threshold nuevamente, pero esta vez con el parámetro cv2.THRESH\_BINARY | cv2.THRESH\_TRIANGLE para que el mismo calcule por sí mismo el valor óptimo de umbralización. Se guarda el resultado en la variable dst y se visualiza para poder compararlo con la imagen de la umbralización manual.

**Parte 4:**

Se desarrolló el programa Part\_4.py. El mismo utiliza los módulos: **cv2, matplotlib** y **os.**

**Al ejecutar la función run\_program():**

El programa recupera la fotografía tomada por el programa 1 y luego mediante un bucle que itera por cada canal/color (b, g, r) calcula el histograma de la imagen para el canal especificado, luego grafica los valores mediante el método plot en el color correspondiente al canal.

Finalmente, al salir del bucle, el programa muestra durante 5 segundos el gráfico completo con el histograma de cada color, cierra la ventana y finaliza la ejecución.

**Parte 5:**

Se desarrolló el programa Part\_5.py. El mismo utiliza los módulos: **cv2,** y **numpy.**

**Al ejecutar la función run\_program():**

El programa crea dos objetos de tipo array como los define el módulo numpy en los cuales almacena los rangos superiores e inferiores de los colores que se desean detectar. Los mismos se ajustaron manualmente para detectar objetos de matiz verde, con un brillo y saturación adecuados para evitar detectar blancos, grises y negros.

Luego se inicia un bucle while en el cual se toma un fotograma con la cámara, el mismo se convierte al tipo HSV mediante cvtColor y luego de este objeto se crea una máscara utilizando el método inRange donde se buscan los pixeles que se encuentren entre los rangos mínimos y máximos definidos en HSV. Luego mediante el método findContours se hallan los contornos de dichos grupos de píxeles, que conforman el objeto color verde. La función drawContours luego dibuja sobre el fotograma original los contornos del objeto, en color rojo y esta imagen es la que se muestra con el método imshow.

**Parte 6:**

Se desarrolló el programa Part\_6.py. El mismo utiliza los módulos: **cv2, os, sys** y **io.**

**Al ejecutar la función run\_program():**

El programa tiene dos modos de funcionamiento. El modo se determina ya sea pasándole el parámetro a la función run\_program() o como argumento (capturado mediante sys.argv[1]) al ejecutar el script como programa principal.

En el modo “Select\_Template" el programa captura una fotografía y la guarda, y luego se inicializa una función callback para detectar acciones del mouse, mientras se visualiza la imagen capturada con imshow.

La función callback del mouse detecta cuando el usuario realiza un click izquierdo y guarda la posición inicial del mouse al hacer dicho click en la tupla de valores (x, y) pointer\_start.

La función callback del mouse a su vez detecta también el evento cuando dicho click se soltó, y recupera nuevamente la posición (x, y) en la tupla pointer\_end. Con estos valores se construye una ROI rectangular delimitada por ambas tuplas de posiciones. El programa luego crea el objeto template\_image a partir de la ROI tomada de la imagen original, y la guarda en un archivo. Posteriormente dibuja con el método rectangle un rectángulo sobre la imagen original que delimita la parte de la imagen que se utilizará como patrón.

El programa utiliza la función save\_pattern\_location para almacenar las posiciones pointer\_start y pointer\_end en un archivo .txt. Este mismo archivo es luego leído con la función load\_pattern\_location en el modo “Match\_Template” para poder recuperar las coordinadas originales en las que se encontraba el patrón en la imagen original, y con ello contrastar si se ha desfasado.

En el modo “Match\_Template” el programa lee la imagen patrón y guarda sus dimensiones en las variables w y h. Luego mediante el método matchTemplate busca en la imagen provista la imagen patrón y esto se guarda en la variable result. La variable result se utiliza como patrón para el método minMaxLoc. De los valores que devuelve dicho método se toma el valor max\_loc, el cual devuelve los valores de posición (x, y) en la esquina superior izquierda en donde se encontró el patrón.

Luego a estos valores se suman las variables w y h (de width y height) para calcular el extremo inferior derecho. Con los extremos superior izquierdo e inferior derecho se dibuja un rectángulo en color rojo sobre la imagen original, para que el usuario pueda localizar en dónde se encontró el patrón.

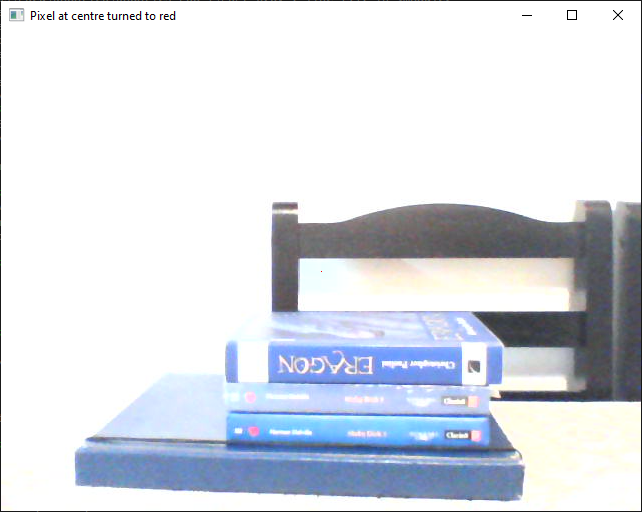
**Parte 7:**

Se desarrolló el programa Part\_7.py. El mismo utiliza los módulos: **cv2, os** y **numpy.**

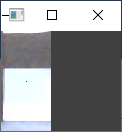
**Al ejecutar la función run\_program():**

El programa lee el archivo de video guardado en "/Images/video\_lineas.avi". Luego entra en un bucle while que se ejecutará hasta que haya leído todos los fotogramas del video. Para cada fotograma lo convierte a escala de gris con cvtColor. Luego utiliza el método Canny para detectar los bordes en la imagen. El método HoughLines se utiliza para detectar las líneas en la imagen. El mismo recibe como parámetros una longitud mínima en píxeles que deben ser coincidentes antes de ser considerados una línea, y el espacio máximo entre los mismos (interrupción en la traza continua de la línea). Este método devuelve una lista de líneas con las coordenadas (x, y) de inicio y fin de cada línea. Para cada línea detectada, luego con el método line y las coordenadas correspondientes, se las dibuja sobre el fotograma original. Esta imagen luego se visualiza hasta que por repetición del bucle se llega a la detección de las líneas en el siguiente fotograma del video y se visualiza este; y así continúa hasta concluir el video.

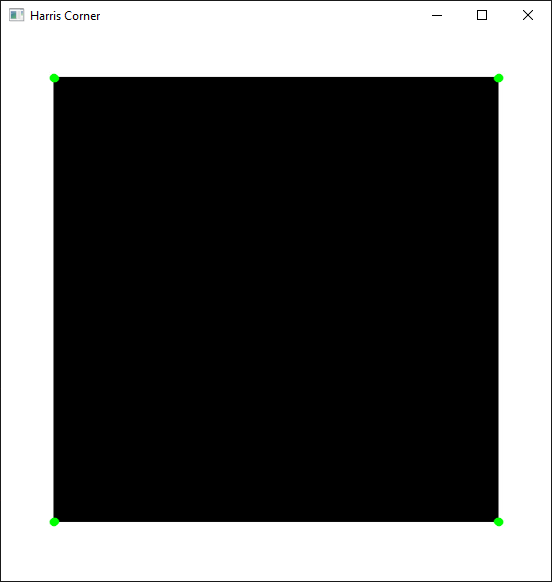
**TABLAS Y GRAFICOS**



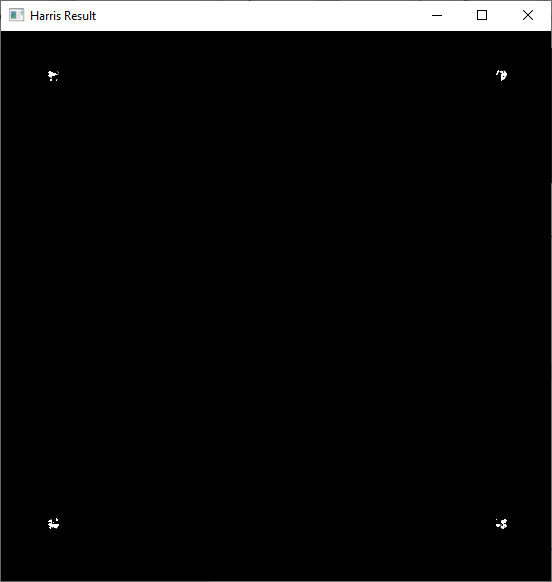
*Part\_1.py: Imagen capturada, con pixel central en rojo.*



*Part\_1.py: ROI, con pixel central en rojo.*



*Part\_2.py: Imagen de cuadrado con sus esquinas marcadas.*



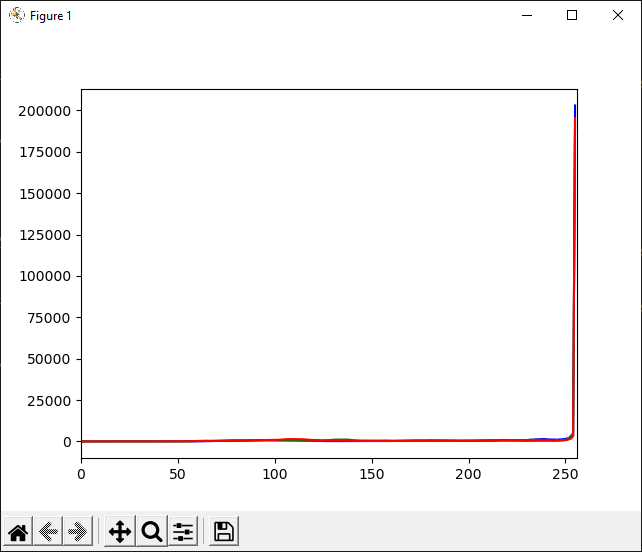
*Part\_2.py: Imagen del resultado del método HarrisCorner.*



*Part\_3.py: Imagen de umbralización ajustada manualmente.*

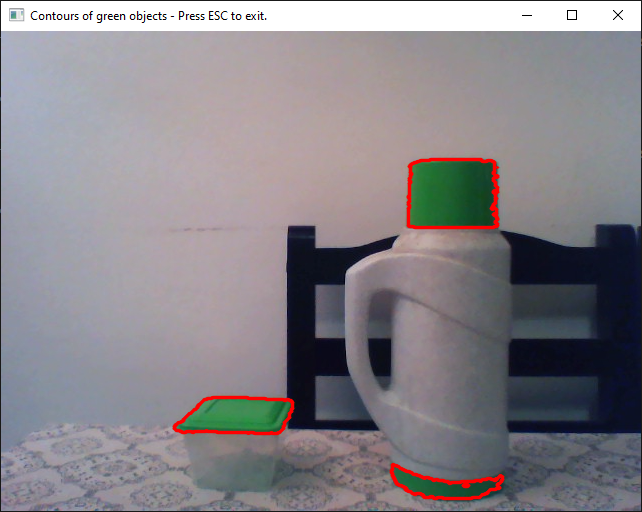


*Part\_3.py: Imagen de umbralización triangulada por el método threshold.*

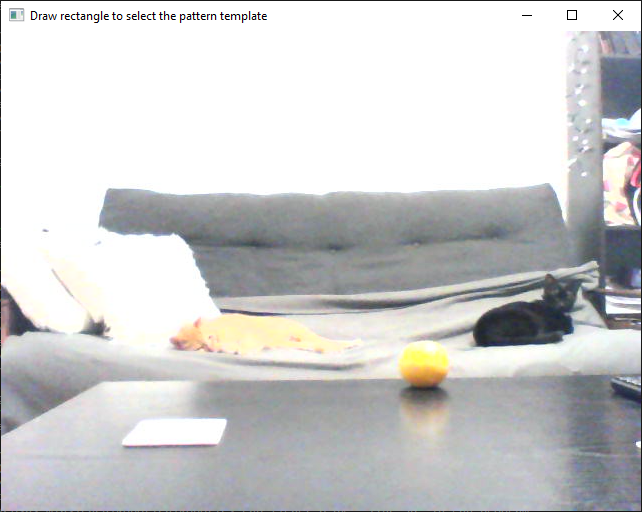


*Part\_4.py: Histograma de la imagen capturada en Part\_1.py.*

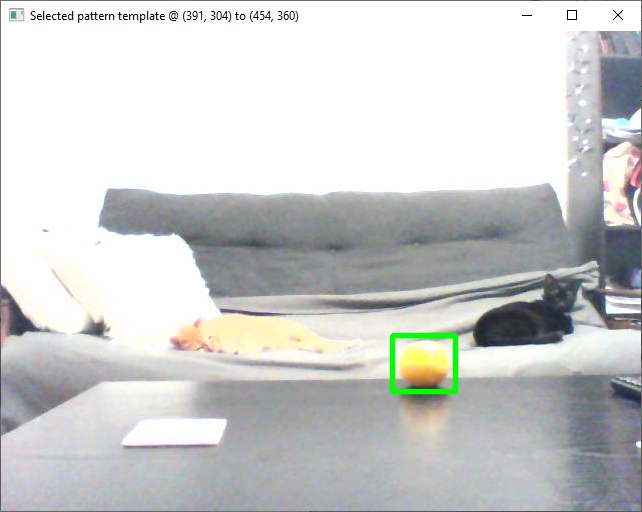
*Nótese la gran concentración de blancos debido a la pared y mantel blancos en la imagen original, con una leve preponderancia de los azules debido a los libros.*



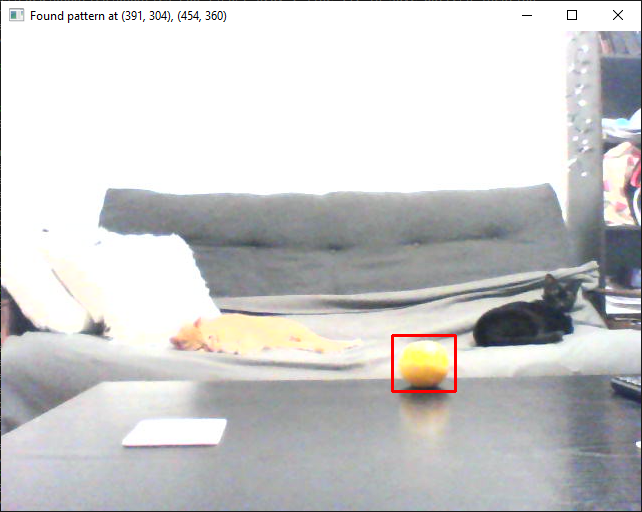
*Part\_5.py: Contornos de objetos verdes.*



*Part\_6.py: Imagen original de la que se tomará el patrón.*



*Part\_6.py: Imagen del patrón seleccionado.*



*Part\_6.py: Imagen del patrón encontrado.*



*Part\_7.py: Captura del video en que se marcan las líneas.*

**ECUACIONES Y CALCULOS NO FUERON NECESARIOS – SE ADJUNTAN CODIGOS FUENTE DE LOS PROGRAMAS DESARROLLADOS**

**Código fuente Part\_1.py:**

"""

Tomar una Foto con la cámara.

Obtener el valor de los canales BGR del Pixel Central.

Modificar dicho Pixel para que se visualice en color Rojo.

Seleccionar una Región de Interés ROI de 100 pixeles en el eje x por 50 pixeles en el eje y donde el pixel modificado quede centrado.

"""

import cv2

import os

path = os.getcwd()

image\_path = "/Images/Part\_1\_captured\_image.png"

path += image\_path

def run\_program():

print("\nRunning Part\_1.py")

capture = cv2.VideoCapture(0)

capture\_ok, image = capture.read()

if capture\_ok:

cv2.imwrite(path, image)

image\_x\_size, image\_y\_size, image\_channels = image.shape

print("Image size: ( {} , {} )".format(image\_x\_size, image\_y\_size) )

image\_half\_x\_size = int( image\_x\_size / 2 )

image\_half\_y\_size = int( image\_y\_size / 2 )

print("Pixel @ [{}, {}] turned to red.".format(image\_half\_y\_size, image\_half\_x\_size) )

image.itemset( (image\_half\_x\_size, image\_half\_y\_size, 0), 0)

image.itemset( (image\_half\_x\_size, image\_half\_y\_size, 1), 0)

image.itemset( (image\_half\_x\_size, image\_half\_y\_size, 2), 255)

cv2.imshow("Pixel at centre turned to red", image)

ROI\_limit\_left = image\_half\_x\_size - 50

ROI\_limit\_right = image\_half\_x\_size + 50

ROI\_limit\_inferior = image\_half\_y\_size - 25

ROI\_limit\_superior = image\_half\_y\_size + 25

image\_ROI = image[ ROI\_limit\_left : ROI\_limit\_right, ROI\_limit\_inferior : ROI\_limit\_superior ]

cv2.imshow('ROI', image\_ROI)

cv2.waitKey(0)

cv2.destroyAllWindows()

else:

print("Revise camera")

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

**Código fuente Part\_2.py:**

"""

Dada una imagen de una figura geométrica detectar los vértices de dicha figura mediante el método de Harris Corner y marcarlos en la imagen original.

"""

import cv2

import os

import numpy

path = os.getcwd()

image\_path = "/Images/cuadrado.jpg"

path += image\_path

image = cv2.imread(path)

check\_image\_type = str( type(image) )

if check\_image\_type == "<class 'NoneType'>":

print("Revise image's path")

print("Finished execution")

quit()

def run\_program():

print("\nRunning Part\_2.py")

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

image\_gray = numpy.float32(image\_gray)

dst = cv2.cornerHarris(image\_gray, 2, 3, 0.04)

height, width = dst.shape

color = (0, 255, 0)

for y in range(0, height):

for x in range(0, width):

if dst.item(y, x) >0.01\* dst.max():

cv2.circle(image, (x, y), 3, color, cv2.FILLED, cv2.LINE\_AA)

cv2.imshow('Harris Result', dst)

cv2.imshow('Harris Corner', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

**Código fuente Part\_3.py:**

"""

Realizar la umbralización de la imagen ¨Herramientas¨ cargando valores de umbral y máximo en forma manual para que las herramientas queden negras usando la función “cv2.THRESH\_BINARY”.

Realizar la misma operación cargando el valor aportado por “cv2.THRESH\_TRIANGLE”, que calcula el valor óptimo de umbralización.

Realizar la Umbralización adaptativa de la imagen “Herramientas”.

"""

import cv2

import os

import numpy

path = os.getcwd()

image\_path = "/Images/herramientas.jpg"

path += image\_path

image = cv2.imread(path)

check\_image\_type = str( type(image) )

if check\_image\_type == "<class 'NoneType'>":

print("Revise image's path")

print("Finished execution")

quit()

def run\_program():

print("\nRunning Part\_3.py")

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

user\_defined\_threshold = 210

threshold\_value, dst = cv2.threshold(image\_gray, user\_defined\_threshold, 255, cv2.THRESH\_BINARY)

cv2.imshow("Manual threshold @ {}".format(threshold\_value), dst)

cv2.waitKey(0)

threshold\_value, dst = cv2.threshold(image\_gray, 0, 255, cv2.THRESH\_BINARY | cv2.THRESH\_TRIANGLE)

cv2.imshow("Triangled threshold @ {}".format(threshold\_value), dst)

print("user\_defined\_threshold: {}. Triangled threshold: {}".format(user\_defined\_threshold, threshold\_value) )

cv2.waitKey(0)

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

**Código fuente Part\_4.py:**

"""

Calcular la frecuencia con la que aparecen los niveles de intensidad Azul, Verde y Rojo de la imagen tomada en el punto 1.

"""

import cv2

import os

from matplotlib import pyplot

path = os.getcwd()

image\_path = "/Images/Part\_1\_captured\_image.png"

path += image\_path

image = cv2.imread(path)

check\_image\_type = str( type(image) )

if check\_image\_type == "<class 'NoneType'>":

print("Revise image's path: {}".format(path))

print("Be sure to run Part\_1.py before Part\_4.py")

quit()

def run\_program():

print("\nRunning Part\_4.py")

colours = ('b', 'g', 'r')

for i, c in enumerate(colours):

histogram = cv2.calcHist([image], [i], None, [256], [0, 256])

pyplot.plot(histogram, color = c)

pyplot.xlim( [0, 256] )

pyplot.show(block=False)

pyplot.pause(5)

pyplot.close("all")

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

**Código fuente Part\_5.py:**

"""

Seguir un objeto de color verde con la cámara con un enmarcado que delimite el objeto a medida que se está moviendo.

"""

import cv2

import numpy as np

def run\_program():

print("\nRunning Part\_5.py")

flag\_key\_pressed = False

upper\_hsv\_range = np.array([80, 255, 255])

lower\_hsv\_range = np.array([40, 70, 50])

capture = cv2.VideoCapture(0)

while (not flag\_key\_pressed):

capture\_ok, frame = capture.read()

if capture\_ok:

hsv = cv2.cvtColor(frame, cv2.COLOR\_RGB2HSV)

mask = cv2.inRange(hsv, lower\_hsv\_range, upper\_hsv\_range)

contours, hierarchy = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cv2.drawContours(frame, contours, -1, (0, 0, 255), 2, cv2.LINE\_AA)

cv2.imshow("Contours of green objects - Press ESC to exit.", frame)

if cv2.waitKey(1) & 0xFF == 27:

flag\_key\_pressed = True

capture.release()

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

**Código fuente Part\_6.py:**

"""

Tomar una foto con la cámara, seleccionar un objeto a identificar en la imagen, hacer un recorte y guardar como patrón.

Identificar ese patrón en la imagen original recuadrándola e indicando su posición.

El patrón debe ser encontrada en una posición definida por el desarrollador.

Si se carga una nueva imagen con el mismo patrón pero en otra posición debe avisarnos patrón desfasado.

"""

import cv2

import os

import sys

from io import open

path = os.getcwd()

image\_path = "/Images/Part\_6\_captured\_image.png"

template\_path = "/Images/Part\_6\_template\_image.png"

full\_image\_path = path + image\_path

full\_template\_path = path + template\_path

pointer\_start = None

pointer\_end = None

flag\_selected\_area = False

start = False

def on\_mouse(event, x, y, flags, source\_image):

global start, flag\_selected\_area, pointer\_start, pointer\_end

if event == cv2.EVENT\_LBUTTONDOWN:

pointer\_start = (x, y)

start = True

elif start and event == cv2.EVENT\_LBUTTONUP:

pointer\_end = (x, y)

ROI\_limit\_left = pointer\_start[0]

ROI\_limit\_right = pointer\_end[0]

ROI\_limit\_inferior = pointer\_start[1]

ROI\_limit\_superior = pointer\_end[1]

template\_image = source\_image[ ROI\_limit\_inferior : ROI\_limit\_superior, ROI\_limit\_left : ROI\_limit\_right ]

cv2.imwrite(full\_template\_path, template\_image)

source\_image = cv2.rectangle(source\_image, pointer\_start, (x, y), (0, 255, 0), 3, cv2.LINE\_AA)

start = False

flag\_selected\_area = True

def save\_pattern\_location():

global pointer\_start, pointer\_end

pattern\_location\_file = open(path + "\Pattern\_location.txt", 'w')

pattern\_location\_file.close()

pattern\_location\_file = open(path + "\Pattern\_location.txt", 'a')

pattern\_location = str(pointer\_start[0]) + '\n'

pattern\_location += str(pointer\_end[0]) + '\n'

pattern\_location += str(pointer\_start[1]) + '\n'

pattern\_location += str(pointer\_end[1])

pattern\_location\_file.write(pattern\_location)

pattern\_location\_file.close()

def load\_pattern\_location():

global pointer\_start, pointer\_end

pointer\_start, pointer\_end = (0, 0)

pattern\_location\_file = open(path + "\Pattern\_location.txt", 'r')

loc\_coordinates = pattern\_location\_file.readlines()

pattern\_location\_file.close()

x = int(loc\_coordinates[0])

y = int(loc\_coordinates[2])

pointer\_start = (x, y)

x = int(loc\_coordinates[1])

y = int(loc\_coordinates[3])

pointer\_end = (x, y)

def run\_program(mode):

print("\nRunning Part\_6.py, mode: {}".format(mode))

global flag\_selected\_area

if (mode == "Select\_Template"):

title = "Draw rectangle to select the pattern template"

capture = cv2.VideoCapture(0)

capture\_ok, image = capture.read()

if capture\_ok:

cv2.imwrite(full\_image\_path, image)

cv2.namedWindow(title)

cv2.setMouseCallback(title, on\_mouse, image)

while (not flag\_selected\_area):

cv2.imshow(title, image)

if cv2.waitKey(1) & 0xFF == 27:

flag\_selected\_area = True

title = "Selected pattern template @ {} to {}".format(pointer\_start, pointer\_end)

save\_pattern\_location()

cv2.imshow(title, image)

cv2.waitKey(0)

else:

print("Revise camera")

if (mode == "Match\_Template"):

template\_image = cv2.imread(full\_template\_path)

check\_image\_type = str( type(template\_image) )

if check\_image\_type == "<class 'NoneType'>":

print("Revise image's path: {}".format(full\_template\_path))

print("Be sure to select a template before trying to find a match")

quit()

original\_img = cv2.imread(full\_image\_path)

check\_image\_type = str( type(original\_img) )

if check\_image\_type == "<class 'NoneType'>":

print("Revise image's path: {}".format(full\_image\_path))

print("Be sure to select a template before trying to find a match")

quit()

try:

load\_pattern\_location()

except:

print("Be sure to select a template before trying to find a match")

quit()

temp\_original\_location = str( pointer\_start ) + str( pointer\_end )

print("Pattern original location: {}".format(temp\_original\_location) )

h, w = template\_image.shape[:-1]

result = cv2.matchTemplate(original\_img, template\_image, 3)

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

top\_left = max\_loc

bottom\_right = (top\_left[0] + w, top\_left[1] + h)

cv2.rectangle(original\_img, top\_left, bottom\_right, (0, 0, 255), 2)

if ( pointer\_start != top\_left and pointer\_end != bottom\_right ):

print("Pattern out of phase")

cv2.imshow("Found pattern at {}, {}".format(str(top\_left), str(bottom\_right) ), original\_img)

cv2.waitKey(0)

if \_\_name\_\_ == "\_\_main\_\_":

try:

mode = str(sys.argv[1])

if (mode != "Select\_Template" and mode != "Match\_Template"):

print("Script argument should be Select\_Template or Match\_Template")

quit()

except IndexError:

print("Script requires one argument (mode)")

quit()

run\_program(mode)

**Código fuente Part\_7.py:**

"""

En una foto o video mp4 en donde se vea una línea (para seguimiento de robots o de autos), realizar el seguimiento de líneas con la función cv2.HoughLines(), como lo haría un robot móvil.

"""

import cv2

import os

import numpy as np

path = os.getcwd()

video\_path = "/Images/video\_lineas.avi"

path += video\_path

def run\_program():

print("\nRunning Part\_7.py")

capture = cv2.VideoCapture(path)

flag\_key\_pressed = False

while ( not flag\_key\_pressed ):

capture\_ok, image = capture.read()

if capture\_ok:

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, 50, 150, apertureSize =3)

lines = cv2.HoughLinesP(edges, 1, np.pi/180, 100, minLineLength=100, maxLineGap=1)

for line in lines:

x1, y1, x2, y2 = line[0]

cv2.line(image, (x1,y1), (x2,y2), (0,255,0), 1, cv2.LINE\_AA)

cv2.imshow("Lines video", image)

cv2.waitKey(1)

else:

break

if cv2.waitKey(1) & 0xFF == 27:

flag\_key\_pressed = True

capture.release()

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

run\_program()

**INFORME SOBRE LAS CONCLUSIONES Y OBSERVACIONES DE LA PRÁCTICA**

El desarrollo de la práctica permitió conocer y experimentar con muchas de las herramientas disponibles para el procesamiento de imágenes y la visión artificial. Se tomó conocimiento de la facilidad con que permite la librería OpenCV la implementación de aplicaciones de visión artificial realmente complejas, y a la vez se destaca la eficiencia de los programas incluídos en la librería. Casi la totalidad de las instrucciones que ejecutan las distintas funciones de la librería se completan en tiempos imperceptibles por el ser humano, esto permite el procesamiento y la visualización de la información en tiempo real (tiempo relativo a las personas).

Cabe destacar el potencial de la herramienta para el desarrollo de proyectos complejos ya que se pudieron lograr implementaciones simples de tareas complejas en muy pocas líneas de código, en parte también, gracias a la gran versatilidad y flexibilidad que otorga el lenguaje Python.