

|  |  |
| --- | --- |
| 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 de computación visual creada por Intel, esta librería está disponible para múltiples plataformas como: Windows, Linux, Mac, Android, además cuenta con soporte para diferentes lenguajes como: Python, Java, C/C++, Net.

La biblioteca OpenCV puede ser usado bajo licencia BSD para proyectos escolares o comerciales, las aplicaciones de esta librería incluyen, la robótica, análisis y procesamiento de imágenes o vídeos, seguimiento y detección de objetos, detección y reconocimiento de rostros, reconocimiento de placas de vehículos, análisis de formas, reconstrucción 3D, realidad aumentada

Dentro de las funciones que se puede realizar se puede modificar a nivel de pixel, para una imagen a color obtendremos los canales azul, verde y rojo, si la imagen es a escala de grises solo tendremos un canal, el cual indica la intensidad de gris de dicho pixel, se puede seleccionar una región de interés y trabajar solo sobre ella.

Podemos dibujar diversas figuras de una menara rápida y sencilla, como círculos, rectángulos.

Uno de los métodos más conocidos para la detección de esquinas es el llamado Harris Corner Detector el cual busca los cambios de intensidad producidos en cada pixel para una región de un tamaño definido, en OpenCV este método esta implementado por la fución cv2.cornerHarris().

La umbralización y umbralización adaptativa son unos de los métodos más simples y eficientes de segmentación, segmentar quiere decir, separar o extraer las regiones de una imagen que nos interese estudiar o analizar, para lograr separar la región deseada se establece un valor que define el umbral, los pixeles cuya intensidad superen el umbral serán rechazado o aceptados, según sea el caso.

Un histograma es una gráfica en donde se muestra la frecuencia con las que aparecen los distintos niveles de intensidad de una imagen a escala de grises, normalmente el nivel de intensidad está en el rango de 0 a 255, en donde el valor 0 representa los color negro y 255 el color blanco, utilizando el histograma de una imagen podemos modificar sus características, por ejemplo, el brillo y contraste.

Un contorno es un conjunto de puntos que conectados unos con otros de manera consecutiva que forman una figura que rodea un objeto determinado, la detección de contornos en OpenCV se aplica sobre imágenes binarias, la figura obtenida puede ser analizada posteriormente para determinar cuál es el objeto que hemos detectado, para obtener los contornos utilizaremos la función findContours(), mientras que drawContours() nos ayudará a dibujarlos.

**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:

**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()

Modificar un Pixel:

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.

Detección de esquinas:

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.

Umbralización y Umbralización adaptativa:

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”.

Histograma:

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

Seguimientos de Objetos:

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

Busqueda patrones Match Template:

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.

Seguimiento de líneas (line detection)

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.

**TABLAS Y GRAFICOS**

**ECUACIONES Y CALCULOS**

**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**

Se indican las conclusiones de la práctica agregando referencias sobre el tema.

**Desarrollo**

1. **Introducción**
2. **Objeto**
3. **Alcance**
4. **Desarrollo (Redactar lo realizado indicando el porqué de las mediciones realizadas)**
5. **Resultados obtenidos (para cada caso estudiado resumir los resultados obtenidos en tablas, gráficos y diagramas)**
6. **Conclusiones y recomendaciones.**