Python Tracker

Manual de Usuario

Contenido

Python Tracker

Información General

Programador

José Emmanuel Salcido Gutiérrez

Repositorio

https://github.com/JESG-27/Python-Tracker.git

Objetivo

El programa se enfoca en realizar el seguimiento de un elemento en un archivo de video para poder trazar su trayectoria y hacer una estimación de la velocidad, utilizando plantillas y algoritmos de Opencv.

Este manual está dirigido a usuarios con conocimientos básicos de Python y procesamiento de imágenes.

Árbol de archivos

```
Python-Tracker:.
│   output.avi
│   README.md
│   templateMatch.py
│   tracker.py
│   videoTemplateMatch.py
│
└───source
        balls.mp4
        basketball.jpg
        basketball.mp4
        freeKick.mp4
        sample_video.mp4
        soccerBall.jpg
        template.jpg
        tenisBall.jpg
        tenisBall.mp4
```

Bibliotecas

- numpy
- opencv-python
- matplotlib
- os

Su instalación se puede realizar con el comando:

```
pip install <biblioteca>
```

Inicio rápido

templateMatch

El archivo de templateMatch es el comportamiento principal de la comparación de plantillas o template matching para pruebas. Se requieren cargar las imágenes desde la carpeta de source en escala de grises. Se recorre un arreglo con los distintos métodos de detección y se van realizando uno por uno y se muestran los resultados en pantalla.

videoTemplateMatch

Dentro de este apartado se ejecutan los mismos pasos que en el pasado, únicamente que se van realizando cuadro por cuadro de un archivo de video, se van detectando y guardando la trayectoria del objeto para posteriormente graficar esas coordenadas y realizar el cálculo estimado de la velocidad. Basándose en el primer y último punto de detección del objeto. Para al final generar un archivo de video con el seguimiento del objeto.

tracker

Este código es una combinación de ambos conceptos anteriores ya que se tiene una detección en 3 puntos distintos de un video para decidir cuales son los mejores resultados. Se ejecuta el análisis del video y se realizan los cálculos de velocidad estimada, basada en la distancia proporcionada por el usuario ya que la variable de tiempo se obtiene por la frecuencia de cuadros por segundo del video.

Funcionamiento a detalle

La coincidencia de plantillas es un método para buscar y encontrar la ubicación de una imagen de plantilla en una imagen más grande. OpenCV viene con una función cv.matchTemplate() para este propósito. Simplemente desliza la imagen de la plantilla sobre la imagen de entrada (como en la convolución 2D) y compara la plantilla y el parche de la imagen de entrada debajo de la imagen de plantilla. En OpenCV se implementan varios métodos de comparación. Devuelve una imagen en escala de grises, donde cada píxel denota cuánto coincide la vecindad de ese píxel con la plantilla.

templateMatch

La lógico principal del código para la coincidencia de plantillas consta en:

Carga de imágenes

- img se carga desde source/frame.jpg representa la imagen más grande donde se debe encontrar el objeto.
- template se carga desde source/template.jpg y representa el objeto que desea detectar. Ambas imágenes se cargan en escala de grises usando cv2.imread(..., 0).

```
# Load the images in grey scale
img = cv2.imread('source/frame.jpg', 0)
template = cv2.imread('source/template.jpg', 0)
```

Coincidencia de plantillas

- Un bucle itera a través de varios métodos de coincidencia de plantillas (lista de métodos).
- Para cada método:
    - img2 es una copia de la imagen original (img).
    - El resultado se obtiene ejecutando cv2.matchTemplate(img2, template, method).
    - Esto calcula la similitud entre la plantilla y las diferentes regiones de la imagen en función del método elegido. min_val, max_val, min_loc y max_loc se obtienen mediante cv2.min MaxLoc(result).
    - Estos valores representan las puntuaciones de similitud mínima y máxima (según el método) y sus ubicaciones correspondientes en la matriz de resultados.

Búsqueda de ubicación

- La ubicación de la mejor coincidencia depende del método de coincidencia:
  - Para métodos como TM_SQDIFF y TM_SQDIFF_NORMED (donde los valores más bajos indican mejores coincidencias), la ubicación se establece en min_loc.
  - Para otros métodos (donde los valores más altos indican mejores coincidencias), la ubicación se establece en max_loc.

```
methods = [cv2.TM_CCOEFF, cv2.TM_CCOEFF_NORMED, cv2.TM_CCORR,
cv2.TM_CCORR_NORMED, cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]

for method in methods:
    img2 = img.copy()
    result = cv2.matchTemplate(img2, template, method)
    min_val, max_val, min_loc, max_loc = cv2.min MaxLoc(result)

    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        location = min_loc
    else:
        location = max_loc
```

Dibujar rectángulo y visualización

- bottom_right calcula las coordenadas de la esquina inferior derecha del objeto detectado en función del tamaño y la ubicación de la plantilla.
- Se dibuja un rectángulo en img2 usando cv2.rectangle para visualizar la región detectada.
- Se cambia el tamaño de la imagen a dimensiones más pequeñas (down_width y down_height) usando cv2.resize.
- La imagen redimensionada se muestra usando cv2.imshow y espera a que se presione la tecla usando cv2.waitKey(0).
- Finalmente, todas las ventanas se cierran usando cv2.destroyAllWindows.

```
    bottom_right = (location[0]+width, location[1]+height)
    cv2.rectangle(img2, location, bottom_right, 255, 2)

    down_width = 600
    down_height = 450
    down_points = (down_width, down_height)
       resized_down  =  cv2.resize(img2,  down_points,  interpolation=
cv2.INTER_LINEAR)

    cv2.imshow('Match', resized_down)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

videoTemplateMatch

En el código que trabaja con el video se inicializa cargando los elementos

Inicialización

- Carga el vídeo (source/sample_video.mp4) y obtiene su velocidad de fotogramas.
- Carga una imagen de plantilla de la bola (source/ball_template.jpg) en escala de grises.
- Crea un grabador de vídeo (output.avi) para almacenar el vídeo de salida.
- Define las dimensiones de la imagen y almacena una lista de trayectorias vacía.
- Selecciona un método de coincidencia de plantillas del arreglo de métodos.

```
# Load the images in grey scale
vid = cv2.VideoCapture('source/sample_video.mp4')
frame_rate = int(vid.get(cv2.CAP_PROP_FPS))
template = cv2.imread('source/ball_template.jpg', 0)
output                =                cv2.VideoWriter("output.avi",
cv2.VideoWriter_fourcc(*'MJPG'),         30,         (int(vid.get(3)),
int(vid.get(4))))

height, width = template.shape
trajectory = []
```

```
methods = [cv2.TM_CCOEFF, cv2.TM_CCOEFF_NORMED, cv2.TM_CCORR,
cv2.TM_CCORR_NORMED, cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]
method = methods[3]
```

Bucle a través de fotogramas de vídeo

- Lee cada fotograma y comprueba si se ha realizado correctamente.
- Convierte el fotograma en escala de grises.
- Realiza la coincidencia de plantillas entre el marco y la plantilla.
- Encuentra la mejor ubicación de coincidencia en función del método elegido.
- Dibuja un rectángulo alrededor de la bola detectada.
- Calcula el centro de la bola detectada y la agrega a la lista de trayectorias.
- Dibuja círculos en el marco para cada punto de la trayectoria.
- Escribe el fotograma modificado en el vídeo de salida.

```
while (vid.isOpened()):
    ret, frame = vid.read()
    if (ret):
        grey_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        result = cv2.matchTemplate(grey_frame, template, method)
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

        if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
            location = min_loc
        else:
            location = max_loc

        bottom_right = (location[0]+width, location[1]+height)
        cv2.rectangle(frame, location, bottom_right, (0,0,255), 2)

        center = (location[0]+(width//2), location[1]+(height//2))
        trajectory.append(center)
        for circle in trajectory:
            cv2.circle(frame, circle, 5, (0, 0, 255), -1)

        output.write(frame)
```

Análisis de trayectoria y resultados

- Extrae las posiciones x e y de la trayectoria (para y se extraen las coordenadas en negativo ya que en la imagen las coordenadas se muestran invertidas).
- Traza la trayectoria como puntos de dispersión rojos.
- Encuentra las posiciones más altas y bajas (que representan los puntos superior e inferior de la trayectoria de la bola).
- Calcula la diferencia de fotogramas entre estos puntos.
- Calcula el tiempo de vuelo utilizando la diferencia de fotogramas y la velocidad de fotogramas.
- Solicita al usuario que introduzca la altura de caída estimada en metros.
- Calcula la velocidad media en función de la altura y el tiempo estimados.
- Imprime la velocidad y el tiempo estimados.

```
positions = [[],[]]
for position in trajectory:
    positions[0].append(position[0])
    positions[1].append(-position[1])

plt.scatter(positions[0], positions[1], color='red', s=10)
max = np.argmax(positions[1])
min = np.argmin(positions[1])
frame_dif = np.absolute(np.absolute(max)-np.absolute(min))
time = (frame_dif*1)/frame_rate
distance = float(input("Estimated drop height (meters): "))
speed = distance/time # speed = distance/time, this is an estimate
distance
print(f"Average speed: {speed:.2f} meters per second")
print(f"Estimated time: {time:.2f} secs")
```

tracker

El código de tracker es una compilación de ambos códigos descritos anteriormente:

Inicialización

- Carga una imagen de plantilla (origen/template.jpg) en escala de grises.
- Carga el vídeo (source/sample_video.mp4) y recupera su velocidad de fotogramas y el total de fotogramas.
- Extrae algunos fotogramas de ejemplo del vídeo para probar diferentes métodos de coincidencia.
- Crea un grabador de vídeo para almacenar el vídeo de salida con los resultados de seguimiento.

```
# Load template from frame
template = cv.imread('source/template.jpg', 0)

# Load video for tracking
cap = cv.VideoCapture('source/sample_video.mp4')
frame_rate = int(cap.get(cv.CAP_PROP_FPS))
total_frames = int(cap.get(cv.CAP_PROP_FRAME_COUNT))

# Extract example frames for matching testing
example_frames = 3
matching_frames = []
for i in range(total_frames//example_frames, total_frames,
total_frames//example_frames):
    cap.set(cv.CAP_PROP_POS_FRAMES, i)
    res, frame = cap.read()
    frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    matching_frames.append(frame)

# Output video
output = cv.VideoWriter("output.avi",
cv.VideoWriter_fourcc(*'MJPG'), 30, (int(cap.get(3)),
int(cap.get(4))))
```

Selección del método de coincidencia de plantillas

- Itera a través de diferentes métodos de coincidencia de plantillas (por ejemplo, TM_CCOEFF, TM_CCOEFF_NORMED, etc.).
- Para cada método, aplícalo a los fotogramas de ejemplo y dibuja rectángulos en ellos para visualizar las coincidencias.
- Muestra los resultados en un formato de cuadrícula utilizando Matplotlib.
- Solicita al usuario que seleccione el método de mejor rendimiento en función de la inspección visual.

```
# Template matching method selection
methods = [cv.TM_CCOEFF, cv.TM_CCOEFF_NORMED, cv.TM_CCORR,
cv.TM_CCORR_NORMED, cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]
methods_results = []
height, width = template.shape
for method in methods:
    for frame in matching_frames:
        frame_copy = frame.copy()
        result = cv.matchTemplate(frame_copy, template, method)
        min_val, max_val, min_loc, max_loc = cv.minMaxLoc(result)

        if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
            location = min_loc
        else:
            location = max_loc

        bottom_right = (location[0]+width, location[1]+height)
        cv.rectangle(frame_copy, location, bottom_right, 255, 2)

        methods_results.append(frame_copy)
```

Seguimiento de video

- Restablece el video al principio.
- Itera por cada fotograma:
    - Convierte el fotograma a escala de grises.
    - Realiza la coincidencia de plantillas mediante el método seleccionado.

- Encuentra la mejor ubicación de coincidencia y dibuja un rectángulo a su alrededor.
- Calcula el centro del objeto detectado y lo agrega a una lista de trayectorias.
- Dibuja círculos en el marco para cada punto de la trayectoria.
- Dibuja líneas que conectan los puntos recientes de la trayectoria para visualizar el movimiento.
- Escribe el fotograma modificado en el vídeo de salida.

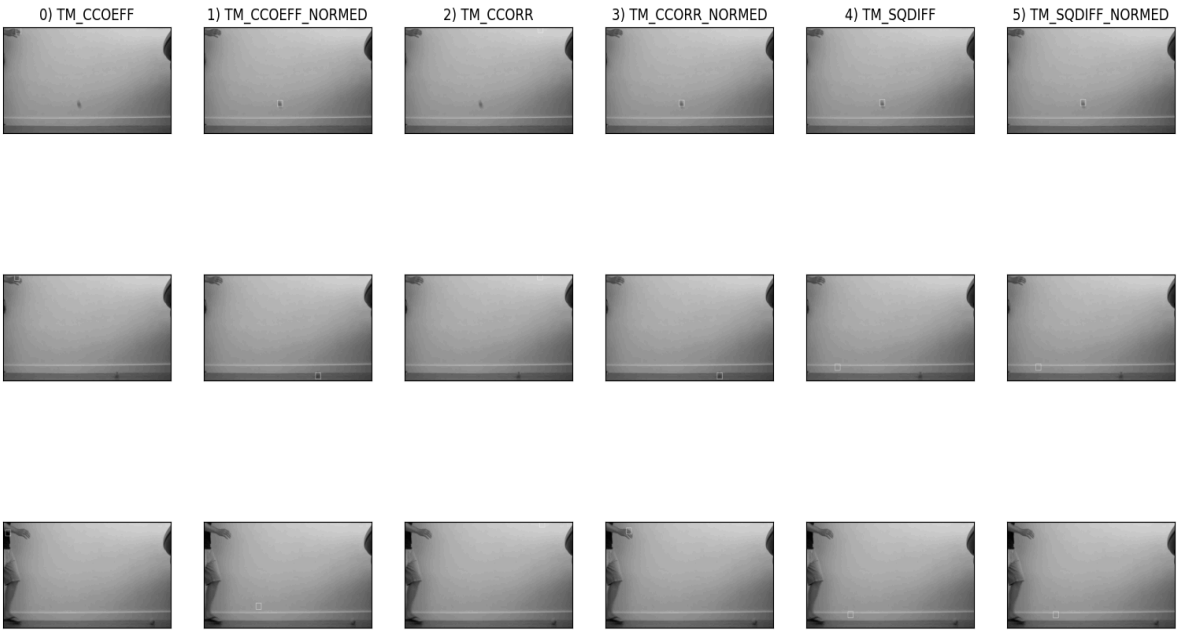
```
while (cap.isOpened()):
    ret, frame = cap.read()
    if (ret):
        grey_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
                  result  =  cv.matchTemplate(grey_frame,  template,
methods[option])
        min_val, max_val, min_loc, max_loc = cv.minMaxLoc(result)

        if methods[option] in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
            location = min_loc
        else:
            location = max_loc

        bottom_right = (location[0]+width, location[1]+height)
        cv.rectangle(frame, location, bottom_right, (0,0,255), 2)

        center = (location[0]+(width//2), location[1]+(height//2))
        trajectory.append(center)
        for circle in trajectory:
            cv.circle(frame, circle, 5, (0, 0, 255), -1)
        if (len(trajectory) > 3):
              frame = cv.line(frame, center, trajectory[-4], (0, 0,
255), 3)
             frame = cv.line(frame, trajectory[-4], trajectory[-3],
(0, 0, 255), 3)
             frame = cv.line(frame, trajectory[-3], trajectory[-2],
(0, 0, 255), 3)

        output.write(frame)
```

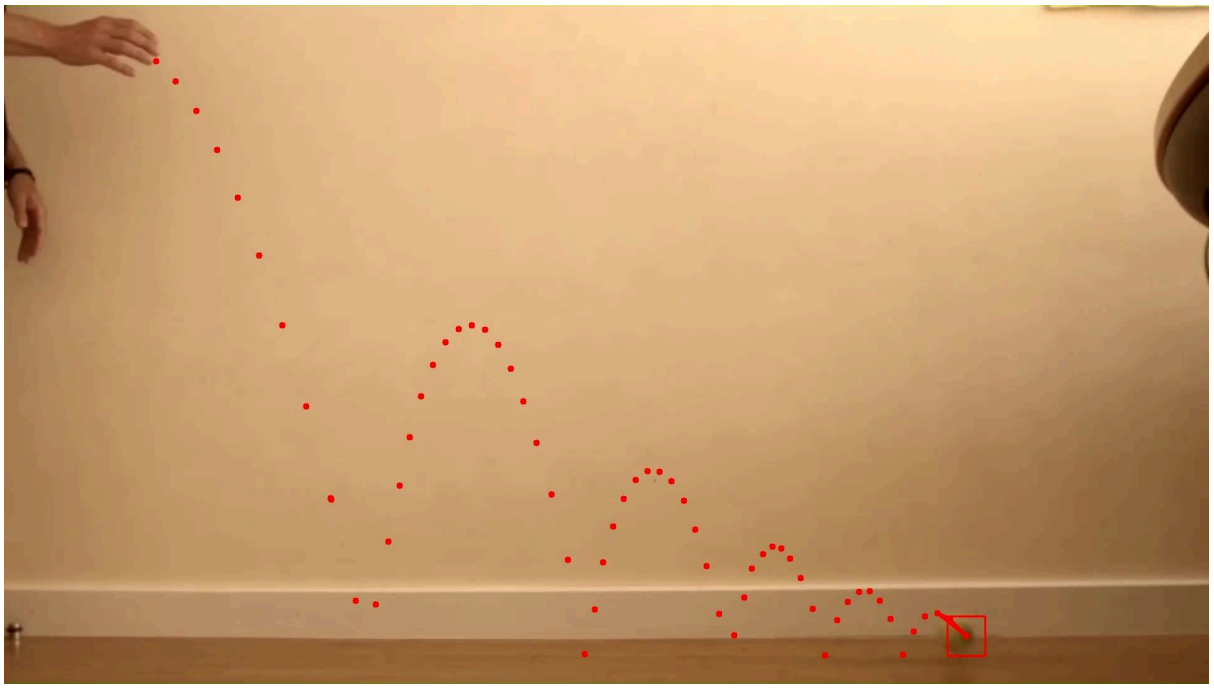
Análisis de trayectoria y estimación de velocidad

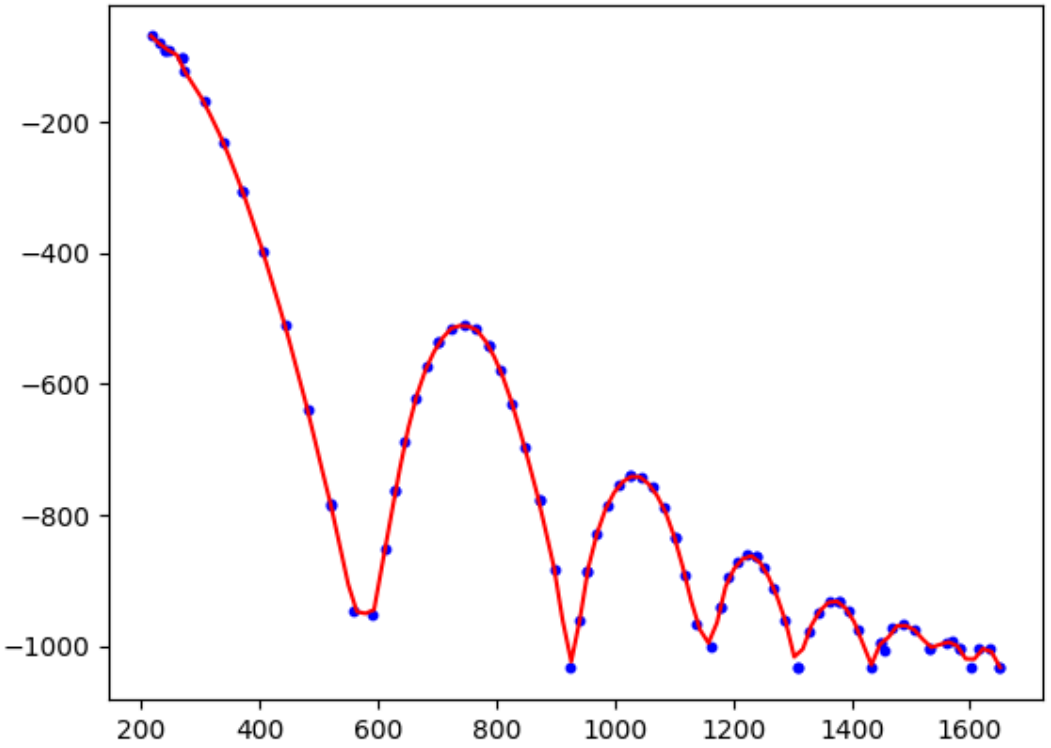
- Extrae las coordenadas x e y de la lista de trayectorias.
- Traza la trayectoria como puntos de dispersión azules utilizando Matplotlib.
- Interpola la trayectoria para crear una curva más suave para la visualización.
- Solicita al usuario que introduzca la distancia estimada recorrida por el objeto.
- Calcula el tiempo de movimiento mediante la diferencia de fotogramas y la velocidad de fotogramas.
- Estima la velocidad media utilizando la distancia y el tiempo.
- Imprime el tiempo, la distancia y la velocidad estimados.

```
# Plot trajectory
print("Trajectory Processing...")
fig = plt.figure()
x = []
y = []
for position in trajectory:
    x.append(position[0])
    y.append(-position[1])

new_x = np.linspace(min(x), max(x), 100)
y_inter = np.interp(new_x, x, y, period=10000)

plt.scatter(x, y, color='blue', s=10, label="match")
plt.plot(new_x, y_inter, color='red', label="Interpolation")
plt.show(block=False)

# Speed
frame_dif = np.absolute(np.absolute(max(y))-np.absolute(min(y)))
time = (frame_dif*1)/frame_rate
distance = float(input("Estimated distance (meters): "))
speed = distance/time # speed = distance/time, this is an estimate
distance
print(f"Time: {time:.2f} seconds")
print(f"Estimated distance: {distance:.2f} meters")
print(f"Average estimated speed: {speed:.2f} meters per second")
```

Ejemplos de Resultados







Dentro de la carpeta de source se encuentran algunos archivos para realizar algunas pruebas, aunque para obtener mejores resultados es necesario modificar el código según se requiera.
Puedes obtener más ejemplos de este link:
https://youtube.com/playlist?list=PL2182063FEDC31F7E&feature=shared

Solución de problemas

- Si se quieren realizar pruebas con un fotograma de un video es necesario extraerlo de este mismo y que sus dimensiones sean las mismas que del video.
- La plantilla para realizar la coincidencia debe de ser extraída directamente de la imagen, para que tenga las mismas dimensiones. De lo contrario ocurrirán errores al realizar el procesamiento o resultados de mala calidad.
- La precisión de la detección depende de factores como la calidad de la plantilla, el ruido de la imagen y las variaciones de los objetos.
- Este código proporciona un ejemplo básico y se puede personalizar o mejorar aún más para escenarios específicos.
- La precisión de la detección y la estimación de la velocidad depende de factores como la visibilidad de la pelota, la velocidad de fotogramas y el ángulo de la cámara.
- El código asume una altura de caída constante y calcula una velocidad media. Se puede mejorar aún más incorporando factores como la gravedad, la aceleración de objetos y la calibración de la cámara según se requiera.

Documentación

3.9.18 documentation. (s. f.). https://docs.python.org/3.9/

NumPY Documentation. (s. f.). https://numpy.org/doc/

Matplotlib documentation — Matplotlib 3.8.3 documentation. (s. f.).

https://matplotlib.org/stable/index.html

OpenCV: OpenCV modules. (s. f.). https://docs.opencv.org/4.x/index.html

OpenCV: Template Matching. (s. f.).

https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html