pip3 install supervision == 0.2.0

—----------------------------------------------------------

**Open supervision lib folder, supervision > detection > Line\_counter.py > comment the trigger function and copy below function there.**

def trigger(self, detections: Detections):

"""

Update the in\_count and out\_count for the detections that cross the line.

Attributes:

detections (Detections): The detections for which to update the counts.

"""

for xyxy, confidence, class\_id, tracker\_id in detections:

# handle detections with no tracker\_id

if tracker\_id is None:

continue

# we check if all four anchors of bbox are on the same side of vector

x1, y1, x2, y2 = xyxy

anchors = [

Point(x=x1, y=y1),

Point(x=x1, y=y2),

Point(x=x2, y=y1),

Point(x=x2, y=y2),

]

triggers = [self.vector.is\_in(point=anchor) for anchor in anchors]

# detection is partially in and partially out

if len(set(triggers)) == 2:

continue

tracker\_state = triggers[0]

# handle new detection

if tracker\_id not in self.tracker\_state:

self.tracker\_state[tracker\_id] = tracker\_state

if tracker\_state:

self.in\_count += 1

else:

self.out\_count += 1

continue

# handle detection on the same side of the line

if self.tracker\_state.get(tracker\_id) == tracker\_state:

continue

self.tracker\_state[tracker\_id] = tracker\_state

if tracker\_state:

self.in\_count += 1

self.out\_count -= 1

else:

self.in\_count -= 1

self.out\_count += 1

—----------------------------------------------------------------------------------------------------------------------------

**In supervision>detection folder replace polygon\_zone.py with bellow code:**  
  
from typing import Optional, Tuple

import cv2

import numpy as np

from shapely.geometry import Polygon, box

from supervision import Detections

from supervision.detection.utils import generate\_2d\_mask

from supervision.draw.color import Color

from supervision.draw.utils import draw\_polygon, draw\_text

from supervision.geometry.core import Position

from supervision.geometry.utils import get\_polygon\_center

class PolygonZone:

def \_\_init\_\_(

self,

polygon: np.ndarray,

frame\_resolution\_wh: Tuple[int, int],

triggering\_position: Position = Position.BOTTOM\_CENTER,

):

self.polygon = polygon

self.frame\_resolution\_wh = frame\_resolution\_wh

self.triggering\_position = triggering\_position

self.mask = generate\_2d\_mask(polygon=polygon, resolution\_wh=frame\_resolution\_wh)

self.current\_count = 0

def is\_bbox\_more\_than\_50\_percent\_inside(self, bbox: np.ndarray) -> bool:

# Convert bbox to shapely box

bbox\_polygon = box(bbox[0], bbox[1], bbox[2], bbox[3])

zone\_polygon = Polygon(self.polygon)

intersection\_area = bbox\_polygon.intersection(zone\_polygon).area

bbox\_area = bbox\_polygon.area

return (intersection\_area / bbox\_area) > 0.5

def trigger(self, detections: Detections) -> np.ndarray:

anchors = (

np.ceil(

detections.get\_anchor\_coordinates(anchor=self.triggering\_position)

).astype(int)

- 1

)

is\_in\_zone = np.array([

self.is\_bbox\_more\_than\_50\_percent\_inside(detection) for detection in detections.xyxy

])

self.current\_count = np.sum(is\_in\_zone)

return is\_in\_zone.astype(bool)

class PolygonZoneAnnotator:

def \_\_init\_\_(

self,

zone: PolygonZone,

color: Color,

thickness: int = 2,

text\_color: Color = Color.black(),

text\_scale: float = 0.5,

text\_thickness: int = 1,

text\_padding: int = 10,

):

self.zone = zone

self.color = color

self.thickness = thickness

self.text\_color = text\_color

self.text\_scale = text\_scale

self.text\_thickness = text\_thickness

self.text\_padding = text\_padding

self.font = cv2.FONT\_HERSHEY\_SIMPLEX

self.center = get\_polygon\_center(polygon=zone.polygon)

def annotate(self, scene: np.ndarray, label: Optional[str] = None) -> np.ndarray:

annotated\_frame = draw\_polygon(

scene=scene,

polygon=self.zone.polygon,

color=self.color,

thickness=self.thickness,

)

annotated\_frame = draw\_text(

scene=annotated\_frame,

text=str(self.zone.current\_count) if label is None else label,

text\_anchor=self.center,

background\_color=self.color,

text\_color=self.text\_color,

text\_scale=self.text\_scale,

text\_thickness=self.text\_thickness,

text\_padding=self.text\_padding,

text\_font=self.font,

)

return annotated\_frame

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**In same folder replace one class in the line\_counter.py:**

class LineZoneAnnotator:

def \_\_init\_\_(

self,

thickness: float = 2,

color: Color = Color.white(),

text\_thickness: float = 2,

text\_color: Color = Color.black(),

text\_scale: float = 0.5,

text\_offset: float = 1.5,

text\_padding: int = 10,

):

"""

Initialize the LineCounterAnnotator object with default values.

Attributes:

thickness (float): The thickness of the line that will be drawn.

color (Color): The color of the line that will be drawn.

text\_thickness (float): The thickness of the text that will be drawn.

text\_color (Color): The color of the text that will be drawn.

text\_scale (float): The scale of the text that will be drawn.

text\_offset (float): The offset of the text that will be drawn.

text\_padding (int): The padding of the text that will be drawn.

"""

self.thickness: float = thickness

self.color: Color = color

self.text\_thickness: float = text\_thickness

self.text\_color: Color = text\_color

self.text\_scale: float = text\_scale

self.text\_offset: float = text\_offset

self.text\_padding: int = text\_padding

def annotate(self, frame: np.ndarray, line\_counter: LineZone) -> np.ndarray:

"""

Draws the line on the frame using the line\_counter provided.

Attributes:

frame (np.ndarray): The image on which the line will be drawn.

line\_counter (LineCounter): The line counter that will be used to draw the line.

Returns:

np.ndarray: The image with the line drawn on it.

"""

cv2.line(

frame,

line\_counter.vector.start.as\_xy\_int\_tuple(),

line\_counter.vector.end.as\_xy\_int\_tuple(),

self.color.as\_bgr(),

self.thickness,

lineType=cv2.LINE\_AA,

shift=0,

)

cv2.circle(

frame,

line\_counter.vector.start.as\_xy\_int\_tuple(),

radius=5,

color=self.text\_color.as\_bgr(),

thickness=-1,

lineType=cv2.LINE\_AA,

)

cv2.circle(

frame,

line\_counter.vector.end.as\_xy\_int\_tuple(),

radius=5,

color=self.text\_color.as\_bgr(),

thickness=-1,

lineType=cv2.LINE\_AA,

)

in\_text = f"right: {line\_counter.in\_count}"

out\_text = f"left: {line\_counter.out\_count}"

(in\_text\_width, in\_text\_height), \_ = cv2.getTextSize(

in\_text, cv2.FONT\_HERSHEY\_SIMPLEX, self.text\_scale, self.text\_thickness

)

(out\_text\_width, out\_text\_height), \_ = cv2.getTextSize(

out\_text, cv2.FONT\_HERSHEY\_SIMPLEX, self.text\_scale, self.text\_thickness

)

# Calculate positions for the labels at the bottom right corner

frame\_height, frame\_width, \_ = frame.shape

in\_text\_x = frame\_width - in\_text\_width - self.text\_padding

in\_text\_y = frame\_height - out\_text\_height - in\_text\_height - self.text\_padding \* 2

out\_text\_x = frame\_width - out\_text\_width - self.text\_padding

out\_text\_y = frame\_height - out\_text\_height - self.text\_padding

in\_text\_background\_rect = Rect(

x=in\_text\_x,

y=in\_text\_y - in\_text\_height,

width=in\_text\_width,

height=in\_text\_height,

).pad(padding=self.text\_padding)

out\_text\_background\_rect = Rect(

x=out\_text\_x,

y=out\_text\_y - out\_text\_height,

width=out\_text\_width,

height=out\_text\_height,

).pad(padding=self.text\_padding)

cv2.rectangle(

frame,

in\_text\_background\_rect.top\_left.as\_xy\_int\_tuple(),

in\_text\_background\_rect.bottom\_right.as\_xy\_int\_tuple(),

self.color.as\_bgr(),

-1,

)

cv2.rectangle(

frame,

out\_text\_background\_rect.top\_left.as\_xy\_int\_tuple(),

out\_text\_background\_rect.bottom\_right.as\_xy\_int\_tuple(),

self.color.as\_bgr(),

-1,

)

cv2.putText(

frame,

in\_text,

(in\_text\_x, in\_text\_y),

cv2.FONT\_HERSHEY\_SIMPLEX,

self.text\_scale,

self.text\_color.as\_bgr(),

self.text\_thickness,

cv2.LINE\_AA,

)

cv2.putText(

frame,

out\_text,

(out\_text\_x, out\_text\_y),

cv2.FONT\_HERSHEY\_SIMPLEX,

self.text\_scale,

self.text\_color.as\_bgr(),

self.text\_thickness,

cv2.LINE\_AA,

)

return frame