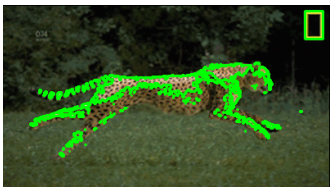
```
# mount drive to work with google drive
# Mount Google Drive
from google.colab import drive
import os
drive.mount('/content/drive/', force_remount=True)
%cd /content
os.chdir('/content/drive/MyDrive/AAAI/') #path was adjusted manually
```

```
Mounted at /content/drive/
/content
```

1. **Background Subtraction**:
   - If you're working with a video stream, first apply a background subtraction algorithm. This will give you the foreground objects in each frame. Tools like OpenCV have functions like `createBackgroundSubtractorMOG2()` for this.

outlined_objects.gif ×



Upload yor GIF

```
!pip install imageio opencv-python
```

```
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (2.31.5)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.8.0.76)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from imageio) (1.23.5)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.10/dist-packages (from imageio) (9.4.0)
```

```
import os
import cv2
import numpy as np
from PIL import Image, ImageSequence
import matplotlib.pyplot as plt
from IPython.display import display, Image as IPImage
from google.colab import files

# Prompt user to upload GIF file
uploaded = files.upload()

# Get the name of the uploaded file
input_gif_name = list(uploaded.keys())[0]
print(input_gif_name)

# Ensure the file is saved in the specified directory
save_directory = "/content/drive/MyDrive/AAAI/"
save_path = os.path.join(save_directory, input_gif_name)
with open(save_path, 'wb') as f:
    f.write(uploaded[input_gif_name])

# Display the uploaded GIF
with open(save_path, "rb") as f:
    display(IPImage(data=f.read(), format='png'))

# Extract frames from the GIF
img = Image.open(save_path)
frames = [frame.copy() for frame in ImageSequence.Iterator(img)]
print(f"Number of frames in the GIF: {len(frames)}")

# Derive a directory to save the extracted frames
frame_save_directory = os.path.join(save_directory, "original_" + os.path.splitext(input_gif_name)[0])
if not os.path.exists(frame_save_directory):
    os.makedirs(frame_save_directory)

# Save frames
saved_paths = []
for idx, frame in enumerate(frames):
    frame_save_path = os.path.join(frame_save_directory, f"frame_{idx + 1}.png")
    frame.save(frame_save_path)
    saved_paths.append(frame_save_path)

# Display the saved frames
fig, axes = plt.subplots(1, len(frames), figsize=(50, 30))
for i, ax in enumerate(axes):
    ax.imshow(plt.imread(saved_paths[i]))
    ax.set_title(f"Frame {i + 1}")
    ax.axis('off')
plt.tight_layout()
plt.show()
print(frame_save_directory)
print(save_path)
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving cheetah.gif to cheetah (6).gif
cheetah (6).gif
```

```
Number of frames in the GIF: 34
```



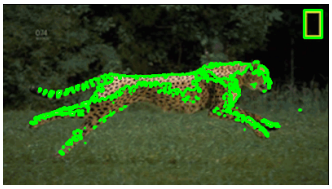
```
/content/drive/MyDrive/AAAI/original_cheetah (6)
/content/drive/MyDrive/AAAI/cheetah (6).gif
```

```
import imageio
import cv2
import numpy as np
import os

# Read the GIF file
gif_path = save_path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Determine the maximum width and height across all frames
max_width = max([frame.shape[1] for frame in frames])
max_height = max([frame.shape[0] for frame in frames])

# ... [previous code]

# Process each frame
outlines = []
for frame in frames:
    # Resize the frame with padding
    delta_w = max_width - frame.shape[1]
```

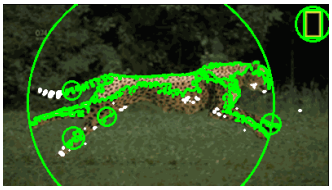
```
    delta_h = max_height - frame.shape[0]
    top, bottom = delta_h // 2, delta_h - (delta_h // 2)
    left, right = delta_w // 2, delta_w - (delta_w // 2)
    frame = cv2.copyMakeBorder(frame, top, bottom, left, right, cv2.BORDER_CONSTANT, value=[0, 0, 0])

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours
    outline = cv2.drawContours(frame.copy(), contours, -1, (0, 255, 0), 2)

    # Ensure the outline has only 3 channels (RGB)
    outline = outline[:, :, :3]

    outlines.append(outline)
    print(len(outlines))
# Save the frames with outlines as a new GIF
outline_save_directory = os.path.join(save_directory, 'outlined_objects.gif')
imageio.mimsave(outline_save_directory, outlines, duration=0.1)
print(outline_save_directory)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
/content/drive/MyDrive/AAAI/outlined_objects.gif
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename=outline_save_directory)
```



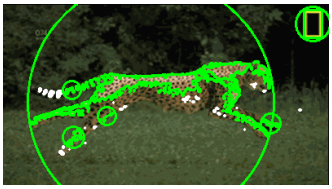
```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/modified_outlined_objects.gif')
```



```
import imageio
import cv2
import os

# Load the GIF using imageio
gif = imageio.mimread(outline_save_directory)

# Define the minimum radius threshold
min_radius_threshold = 10  # Adjust this value as needed

processed_frames = []

# Process each frame in the GIF
for frame in gif:
    # Convert the frame from imageio format to OpenCV format
    image = cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR)

    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Process each contour
    for contour in contours:
```

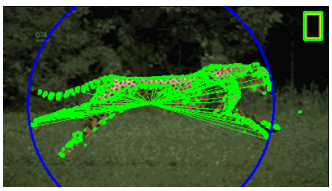
```
        # Calculate the enclosing circle
        (x, y), radius = cv2.minEnclosingCircle(contour)

        # Check if the radius is above the threshold
        if radius > min_radius_threshold:
            # Draw the circle around the contour
            cv2.circle(image, (int(x), int(y)), int(radius), (0, 255, 0), 2)
        else:
            # Remove the contour from the image by filling it with the background color (white in this case)
            cv2.drawContours(image, [contour], 0, (255, 255, 255), -1)

    # Convert the processed frame back to imageio format
    processed_frame = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
    processed_frames.append(processed_frame)

# Save the processed frames as a new animated GIF using imageio
imageio.mimsave(os.path.join(save_directory, 'modified_outlined_objects.gif'), processed_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/modified_outlined_objects.gif')
```



```
import imageio
import cv2
import os

# Load the GIF using imageio
gif = imageio.mimread(outline_save_directory)

processed_frames = []

# Process each frame in the GIF
for frame in gif:
    # Convert the frame from imageio format to OpenCV format
    image = cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR)

    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Identify the largest contour based on the radius of the enclosing circle
    largest_radius = 0
    largest_contour = None
    for contour in contours:
        (_, _), radius = cv2.minEnclosingCircle(contour)
        if radius > largest_radius:
            largest_radius = radius
            largest_contour = contour

    # Draw the enclosing circle of the largest contour in blue
    (x, y), _ = cv2.minEnclosingCircle(largest_contour)
    cv2.circle(image, (int(x), int(y)), int(largest_radius), (255, 0, 0), 2)

    # Draw lines from the center to every 10th pixel of the largest contour
    for i in range(0, len(largest_contour), 10):
        cv2.line(image, (int(x), int(y)), tuple(largest_contour[i][0]), (0, 255, 0), 1)

    # Convert the processed frame back to imageio format
    processed_frame = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
    processed_frames.append(processed_frame)


# Save the modified image
imageio.mimsave(os.path.join(save_directory, 'modified_outlined_objects_blue.gif'), processed_frames, duration=0.1)
```

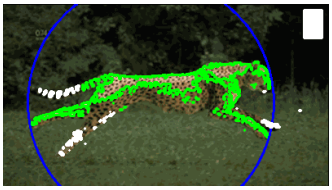
```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/modified_outlined_objects_blue.gif')
```



```
import imageio
import cv2
import os

# Load the GIF using imageio
gif = imageio.mimread(outline_save_directory)

processed_frames = []

# Process each frame in the GIF
for frame in gif:
    # Convert the frame from imageio format to OpenCV format
    image = cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR)

    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Find contours
```

```
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Identify the largest contour based on the radius of the enclosing circle
    largest_radius = 0
    largest_contour = None
    for contour in contours:
        (_, _), radius = cv2.minEnclosingCircle(contour)
        if radius > largest_radius:
            largest_radius = radius
            largest_contour = contour

    # Draw the enclosing circle of the largest contour in blue
    (x, y), _ = cv2.minEnclosingCircle(largest_contour)
    cv2.circle(image, (int(x), int(y)), int(largest_radius), (255, 0, 0), 2)

    # Create a mask for the lines drawn from the center to the outline
    line_mask = np.zeros_like(gray)
    for i in range(0, len(largest_contour), 10):
        cv2.line(line_mask, (int(x), int(y)), tuple(largest_contour[i][0]), 255, 1)

    # Check each smaller contour for overlap with the lines
    for contour in contours:
        if cv2.contourArea(contour) < cv2.contourArea(largest_contour):
            mask = np.zeros_like(gray)
            cv2.drawContours(mask, [contour], -1, 255, -1)
            overlap = cv2.bitwise_and(mask, line_mask)
            if not np.any(overlap):
                cv2.drawContours(image, [contour], -1, (255, 255, 255), -1)

    # Convert the processed frame back to imageio format
    processed_frame = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
    processed_frames.append(processed_frame)

# Save the processed frames as a new animated GIF using imageio
imageio.mimsave(os.path.join(save_directory, 'modified_outlined_objects2.gif'), processed_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/modified_outlined_objects2.gif')
```



```
!pip install scikit-image
```

```
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-packages (0.19.3)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (1.23.5)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (1.11.3)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (3.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (9.4.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (2.31.5)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (2023.9.26)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (1.4.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (23.2)
```

Skeleton Draft import imageio import cv2 import os from skimage.morphology import skeletonize from skimage.color import rgb2gray

Load the GIF using imageio

gif_path = "path_to_your_gif.gif" # Replace with your GIF path gif = imageio.mimread(gif_path)

skeletonized_frames = []

Process each frame in the GIF

for frame in gif:

```
# Convert the frame from imageio format to OpenCV format
image = cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR)

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Thresholding
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Normalize the thresholded image to [0, 1] and skeletonize
skeleton = skeletonize(thresh / 255)

# Convert the skeleton back to [0, 255] range
skeleton_255 = (skeleton * 255).astype(np.uint8)

# Convert the processed frame back to imageio format
processed_frame = cv2.cvtColor(skeleton_255, cv2.COLOR_GRAY2RGBA)
skeletonized_frames.append(processed_frame)
```

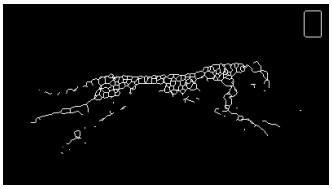
Save the skeletonized frames as a new animated GIF using imageio

save_directory = "path_to_save_directory" # Replace with your desired save directory imageio.mimsave(os.path.join(save_directory, 'skeletonized_objects.gif'), skeletonized_frames, duration=0.1)

```
import imageio
import cv2
import os
from skimage.morphology import skeletonize
from skimage.color import rgb2gray

# Load the GIF using imageio
gif_path = outline_save_directory  # Replace with your GIF path
gif = imageio.mimread(gif_path)

skeletonized_frames = []
```

```
# Process each frame in the GIF
for frame in gif:
    # Convert the frame from imageio format to OpenCV format
    image = cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR)

    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Normalize the thresholded image to [0, 1] and skeletonize
    skeleton = skeletonize(thresh / 255)

    # Convert the skeleton back to [0, 255] range
    skeleton_255 = (skeleton * 255).astype(np.uint8)

    # Convert the processed frame back to imageio format
    processed_frame = cv2.cvtColor(skeleton_255, cv2.COLOR_GRAY2RGBA)
    skeletonized_frames.append(processed_frame)

save_directory = '/content/drive/MyDrive/AAAI/'

# Save the skeletonized frames as a new animated GIF using imageio
imageio.mimsave(os.path.join(save_directory, 'skeletonized_objects.gif'), skeletonized_frames, duration=0.1)
```

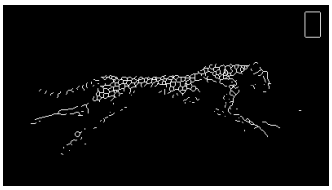
```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/skeletonized_objects.gif')
```



Skeleton draft

```
import imageio
import cv2
import os
from skimage.morphology import skeletonize
from skimage.color import rgb2gray

# Load the GIF using imageio
gif = imageio.mimread('/content/drive/MyDrive/AAAI/cheetah (5).gif')

skeletonized_frames = []

# Process each frame in the GIF
for frame in gif:
    # Convert the frame from imageio format to OpenCV format
    image = cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR)

    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Normalize the thresholded image to [0, 1] and skeletonize
    skeleton = skeletonize(thresh / 255)

    # Convert the skeleton back to [0, 255] range
    skeleton_255 = (skeleton * 255).astype(np.uint8)

    # Convert the processed frame back to imageio format
    processed_frame = cv2.cvtColor(skeleton_255, cv2.COLOR_GRAY2RGBA)
    skeletonized_frames.append(processed_frame)

save_directory = '/content/drive/MyDrive/AAAI/'

# Save the skeletonized frames as a new animated GIF using imageio
imageio.mimsave(os.path.join(save_directory, 'skeletonized_objects2.gif'), skeletonized_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/skeletonized_objects2.gif')
```



Motion Detection

```
import cv2
import imageio

# Load the GIF using imageio
gif_path = "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

# Initialize the background model with the first frame
background_model = cv2.cvtColor(frames[0], cv2.COLOR_BGR2GRAY)

motion_detected_frames = []
```

```
for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Compute the absolute difference between the current frame and background model
    diff = cv2.absdiff(gray_frame, background_model)

    # Threshold the difference to get a binary image
    _, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)

    # Apply morphological operations to clean up the binary image
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    thresh = cv2.dilate(thresh, kernel, iterations=2)
    thresh = cv2.erode(thresh, kernel, iterations=1)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on the original frame
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter out small contours
            (x, y, w, h) = cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    motion_detected_frames.append(frame)

# Save the frames with motion detection as a new GIF
imageio.mimsave(os.path.join(save_directory, 'motion_detected.gif'), motion_detected_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/motion_detected.gif')
```

```
import cv2
import imageio

# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

# Initialize the background model with the first frame
background_model = cv2.cvtColor(frames[0], cv2.COLOR_BGR2GRAY)

motion_detected_frames = []

for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Compute the absolute difference between the current frame and background model
    diff = cv2.absdiff(gray_frame, background_model)

    # Threshold the difference to get a binary image
    _, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)

    # Apply morphological operations to clean up the binary image
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    thresh = cv2.dilate(thresh, kernel, iterations=2)
    thresh = cv2.erode(thresh, kernel, iterations=1)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw circles on the original frame
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter out small contours
            # Calculate the centroid of the contour
            M = cv2.moments(contour)
            if M["m00"] != 0:
                cX = int(M["m10"] / M["m00"])
                cY = int(M["m01"] / M["m00"])
            else:
                cX, cY = 0, 0

            # Calculate the radius based on the contour's area
            radius = int((cv2.contourArea(contour) / 3.14) ** 0.5)

            # Draw the circle
            cv2.circle(frame, (cX, cY), radius, (0, 255, 0), 2)

    motion_detected_frames.append(frame)

# Save the frames with motion detection as a new GIF
imageio.mimsave(os.path.join(save_directory, 'motion_detected_circles.gif'), motion_detected_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/motion_detected_circles.gif')
```

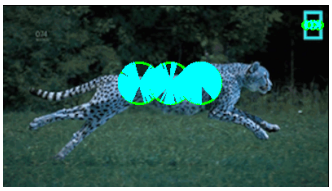
```
import cv2
import imageio
```

```
# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

# Initialize the background model with the first frame
background_model = cv2.cvtColor(frames[0], cv2.COLOR_BGR2GRAY)

motion_detected_frames = []

for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Compute the absolute difference between the current frame and background model
    diff = cv2.absdiff(gray_frame, background_model)

    # Threshold the difference to get a binary image
    _, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)

    # Apply morphological operations to clean up the binary image
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    thresh = cv2.dilate(thresh, kernel, iterations=2)
    thresh = cv2.erode(thresh, kernel, iterations=1)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw circles on the original frame
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter out small contours
            x, y, w, h = cv2.boundingRect(contour)

            # Calculate the radius based on the contour's area
            radius = int((cv2.contourArea(contour) / (3 * 3.14)) ** 0.5)

            # Split based on width and height
            if w > h:  # Split horizontally
                centroids = [(x + w//6, y + h//2), (x + w//2, y + h//2), (x + 5*w//6, y + h//2)]
            else:  # Split vertically
                centroids = [(x + w//2, y + h//6), (x + w//2, y + h//2), (x + w//2, y + 5*h//6)]

            for cX, cY in centroids:
                # Draw the circle
                cv2.circle(frame, (cX, cY), radius, (0, 255, 0), 2)

    motion_detected_frames.append(frame)

# Save the frames with motion detection as a new GIF
imageio.mimsave(os.path.join(save_directory, 'motion_detected_3circles.gif'), motion_detected_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/motion_detected_3circles.gif')
```

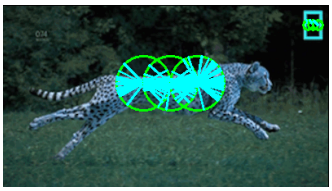
```
import cv2
import imageio
import numpy as np

# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

motion_detected_frames = []

for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Threshold the frame to get a binary image
    _, thresh = cv2.threshold(gray_frame, 127, 255, cv2.THRESH_BINARY)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw circles on the original frame
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter out small contours
            (x, y), main_radius = cv2.minEnclosingCircle(contour)

            # Calculate the radius for the three smaller circles
            small_radius = int(main_radius / 4)

            # Calculate the positions for the three smaller circles (always horizontal)
            offsets = [-small_radius * 1.5, 0, small_radius * 1.5]
            centroids = [(int(x + offset), int(y)) for offset in offsets]

            for cX, cY in centroids:
                # Draw the circle
                cv2.circle(frame, (cX, cY), small_radius, (0, 255, 0), 2)

                # Draw the line from the center of the small circle to the outline of the object
                for point in contour:
                    point = point[0]
                    angle = np.arctan2(point[1] - cY, point[0] - cX)
                    endX = int(cX + small_radius * np.cos(angle))
                    endY = int(cY + small_radius * np.sin(angle))
                    cv2.line(frame, (cX, cY), (endX, endY), (0, 255, 255), 2)  # Yellow line

    motion_detected_frames.append(frame)

# Save the frames with motion detection as a new GIF
imageio.mimsave(os.path.join(save_directory, 'motion_detected_circlesV4.gif'), motion_detected_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/motion_detected_circlesV4.gif')
```



```
import cv2
import imageio
import numpy as np

# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

motion_detected_frames = []

for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Threshold the frame to get a binary image
    _, thresh = cv2.threshold(gray_frame, 127, 255, cv2.THRESH_BINARY)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw circles on the original frame
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter out small contours
            (x, y), main_radius = cv2.minEnclosingCircle(contour)

            # Calculate the radius for the three smaller circles
            small_radius = int(main_radius / 3)

            # Calculate the positions for the three smaller circles (always horizontal)
            offsets = [-small_radius, 0, small_radius]
            centroids = [(int(x + offset), int(y)) for offset in offsets]

            for cX, cY in centroids:
                # Draw the circle
                cv2.circle(frame, (cX, cY), small_radius, (0, 255, 0), 2)

                # Draw the line from the center of the small circle to every 10th pixel of the object's outline
                for i, point in enumerate(contour):
                    if i % 10 == 0:
                        point = point[0]
                        angle = np.arctan2(point[1] - cY, point[0] - cX)
                        endX = int(cX + small_radius * np.cos(angle))
                        endY = int(cY + small_radius * np.sin(angle))
                        cv2.line(frame, (cX, cY), (endX, endY), (0, 255, 255), 2)  # Yellow line

    motion_detected_frames.append(frame)

# Save the frames with motion detection as a new GIF
imageio.mimsave(os.path.join(save_directory, 'motion_detected_circlesV5.gif'), motion_detected_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/motion_detected_circlesV5.gif')
```



```
import cv2
import imageio
import numpy as np

# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

# List to store the outline points for each frame
outlines = []

for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Threshold the frame to get a binary image
    _, thresh = cv2.threshold(gray_frame, 127, 255, cv2.THRESH_BINARY)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Extract the largest contour (assuming it's the main object)
    if contours:
        main_contour = max(contours, key=cv2.contourArea)
        outline_points = [(point[0][0], point[0][1]) for point in main_contour]
        outlines.append(outline_points)

# Now, the 'outlines' list contains the (x, y) coordinates of the outline points for each frame
print(outlines)
```

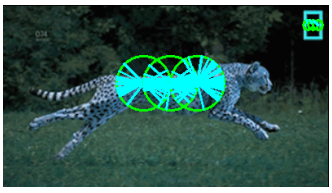
```
[[(284, 71), (283, 72), (283, 73), (282, 74), (281, 74), (280, 73), (280, 72), (278, 72), (277, 73), (273, 73), (272, 74), (271, 73), (269,
```

Point Array

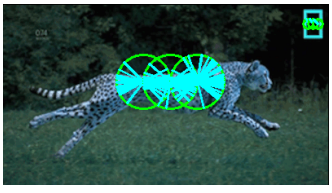
```
import cv2
import imageio
import numpy as np
import csv

# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

# List to store the outline points for each frame
outlines = []

for frame in frames:
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Threshold the frame to get a binary image
    _, thresh = cv2.threshold(gray_frame, 127, 255, cv2.THRESH_BINARY)

    # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Extract the largest contour (assuming it's the main object)
    if contours:
        main_contour = max(contours, key=cv2.contourArea)
        outline_points = [(point[0][0], point[0][1]) for point in main_contour]
        outlines.append(outline_points)

mypath = os.path.join(save_directory, 'outlines.csv')

# Save the outline points to a CSV file
with open(mypath, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["Frame", "X", "Y"])  # Header row
    for frame_num, outline in enumerate(outlines):
        for point in outline:
            writer.writerow([frame_num] + list(point))
```

```
import cv2
import imageio
import numpy as np

# Load the GIF using imageio
gif_path =  "/content/drive/MyDrive/AAAI/cheetah (5).gif"  # Replace with your GIF path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Convert frames to OpenCV format
frames = [cv2.cvtColor(frame, cv2.COLOR_RGBA2BGR) for frame in frames]

# List to store the frames with only the green outlines
green_outline_frames = []

for frame in frames:
    # Create a mask where the green channel is dominant
    green_mask = np.where((frame[:,:,1] > frame[:,:,0]) & (frame[:,:,1] > frame[:,:,2]), 1, 0).astype(np.uint8)

    # Use the mask to extract the green outlines from the original frame
    green_outlines = cv2.bitwise_and(frame, frame, mask=green_mask)

    # Convert the processed frame back to imageio format
    processed_frame = cv2.cvtColor(green_outlines, cv2.COLOR_BGR2RGBA)
    green_outline_frames.append(processed_frame)

imageio.mimsave(os.path.join(save_directory, 'green_outlines_only.gif'), motion_detected_frames, duration=0.1)
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/green_outlines_only.gif')
```



```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/outlined_objectsV7.gif')
```



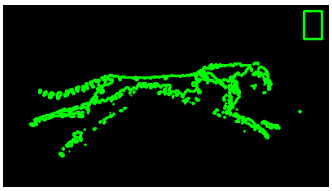
GIF of Outline

```
import imageio
import cv2
import numpy as np
import os

# Read the GIF file
gif_path = save_path
frames = [frame for frame in imageio.get_reader(gif_path)]
```

```
# Determine the maximum width and height across all frames
max_width = max([frame.shape[1] for frame in frames])
max_height = max([frame.shape[0] for frame in frames])

# Process each frame
outlines = []
for frame in frames:
    # Resize the frame with padding
    delta_w = max_width - frame.shape[1]
    delta_h = max_height - frame.shape[0]
    top, bottom = delta_h // 2, delta_h - (delta_h // 2)
    left, right = delta_w // 2, delta_w - (delta_w // 2)
    frame = cv2.copyMakeBorder(frame, top, bottom, left, right, cv2.BORDER_CONSTANT, value=[0, 0, 0])

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Create a blank black image of the same size as the frame
    blank_image = np.zeros_like(frame)

    # Draw contours on the blank image
    outline = cv2.drawContours(blank_image, contours, -1, (0, 255, 0), 2)

    # Ensure the outline has only 3 channels (RGB)
    outline = outline[:, :, :3]

    outlines.append(outline)

# Save the frames with outlines as a new GIF
outline_save_directory = os.path.join(save_directory, 'outlined_objects2.gif')
imageio.mimsave(outline_save_directory, outlines, duration=0.1)
print(outline_save_directory)
```

```
/content/drive/MyDrive/AAAI/outlined_objects2.gif
```

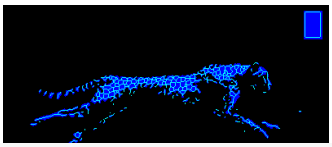
```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/outlined_objects2.gif')
```



```
import imageio
import cv2
import numpy as np
import os
from skimage.morphology import skeletonize

# Read the GIF file
gif_path = save_path
frames = [frame for frame in imageio.get_reader(gif_path)]

# Determine the maximum width and height across all frames
max_width = max([frame.shape[1] for frame in frames])
max_height = max([frame.shape[0] for frame in frames])

# Process each frame
outlines = []
for frame in frames:
    # Resize the frame with padding
    delta_w = max_width - frame.shape[1]
    delta_h = max_height - frame.shape[0]
    top, bottom = delta_h // 2, delta_h - (delta_h // 2)
    left, right = delta_w // 2, delta_w - (delta_w // 2)
    frame = cv2.copyMakeBorder(frame, top, bottom, left, right, cv2.BORDER_CONSTANT, value=[0, 0, 0])

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGBA2GRAY)

    # Thresholding
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Create a blank black image of the same size as the frame with 3 channels (RGB)
    blank_image = np.zeros((frame.shape[0], frame.shape[1], 3), dtype=np.uint8)

    # Fill the object within the outline with red color
    cv2.drawContours(blank_image, contours, -1, (0, 0, 255), -1)  # -1 fill the contour

    # Compute the skeleton of the object
    skeleton = skeletonize(thresh == 255)
    skeleton = np.array(skeleton, dtype=np.uint8) * 255

    # Overlay the yellow skeleton on the red object
    blank_image[skeleton == 255, :] = [0, 255, 255]  # Set the skeleton pixels to yellow

    outlines.append(blank_image)

# Save the frames with outlines as a new GIF
outline_save_directory = os.path.join(save_directory, 'outlined_objects_with_skeleton3.gif')
imageio.mimsave(outline_save_directory, outlines, duration=0.1)
print(outline_save_directory)
```

```
/content/drive/MyDrive/AAAI/outlined_objects_with_skeleton3.gif
```

```
from IPython.display import Image

# Display the outlined GIF
Image(filename='/content/drive/MyDrive/AAAI/outlined_objects_with_skeleton3.gif')
```



```
import cv2
import numpy as np
from skimage.morphology import skeletonize
from google.colab.patches import cv2_imshow  # Use this for displaying images in Colab

def compute_skeleton(frame):
    """Compute the skeleton of a binary frame."""
    # Convert frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Binarize the image
    _, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)
    # Compute the skeleton
    skeleton = skeletonize(binary // 255)
    return skeleton.astype(np.uint8) * 255

def detect_motion(skeleton_prev, skeleton_curr):
    """Detect motion by comparing two skeletons."""
    # Compute the difference between the two skeletons
    diff = cv2.absdiff(skeleton_prev, skeleton_curr)
    # Apply a threshold to detect significant motion
    _, motion_mask = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)
    return motion_mask

# Load the video or sequence of frames
cap = cv2.VideoCapture('/content/drive/MyDrive/AAAI/mov_bbb.mp4')

ret, frame_prev = cap.read()
skeleton_prev = compute_skeleton(frame_prev)

while True:
    ret, frame_curr = cap.read()
    if not ret:
        break

    skeleton_curr = compute_skeleton(frame_curr)
    motion_mask = detect_motion(skeleton_prev, skeleton_curr)

    # Highlight areas of motion in the frame
    frame_curr[motion_mask == 255] = [0, 0, 255]  # Red color for motion areas

    cv2_imshow(frame_curr)  # Display the frame in Colab

    # Update the previous frame and skeleton
    frame_prev = frame_curr.copy()
    skeleton_prev = skeleton_curr.copy()

cap.release()
```