**TASK 1**  
pip install torch torchvision segment-anything diffusers transformers

!wget https://dl.fbaipublicfiles.com/segment\_anything/sam\_vit\_h\_4b8939.pth

def load\_sam\_model():

    sam\_checkpoint = "/content/sam\_vit\_h\_4b8939.pth"  # Correct path to the downloaded file

    model\_type = "vit\_h"

    sam = sam\_model\_registry[model\_type](checkpoint=sam\_checkpoint)

    return sam

!ls /content

import torch

from segment\_anything import sam\_model\_registry, SamPredictor

import cv2

import numpy as np

# Load the SAM model

def load\_sam\_model():

    sam\_checkpoint = "/content/sam\_vit\_h\_4b8939.pth"  # Correct path after downloading the model

    model\_type = "vit\_h"

    sam = sam\_model\_registry[model\_type](checkpoint=sam\_checkpoint)

    return sam

def segment\_object(image\_path, object\_class, output\_path):

    # Load SAM model

    sam = load\_sam\_model()

    predictor = SamPredictor(sam)

    # Read the image

    image = cv2.imread(image\_path)

    predictor.set\_image(image)

    # Dummy object prompt for bounding box (Replace with actual detection model for the object class)

    # This example uses a static bounding box; in a real scenario, you would replace it with automatic detection

    box = np.array([50, 50, 300, 300])  # Example bounding box for testing

    # Predict mask for the detected object

    masks, \_, \_ = predictor.predict(box=box)

    # Apply the mask on the image

    mask\_image = np.zeros\_like(image)

    mask\_image[masks[0]] = [0, 0, 255]  # Red color mask for the object

    # Blend original image and mask

    output\_image = cv2.addWeighted(image, 0.7, mask\_image, 0.3, 0)

    # Save the output

    cv2.imwrite(output\_path, output\_image)

    print(f"Segmentation result saved to {output\_path}")

# Example usage

image\_path = '/content/chair.jpg'  # Path to your input image

object\_class = 'chair'  # The class of the object you want to segment

output\_path = '/content/segmented.png'  # Output file path

segment\_object(image\_path, object\_class, output\_path)

This was my input image



This was my output image



**TASK 2**

import torch

from segment\_anything import sam\_model\_registry, SamPredictor

import cv2

import numpy as np

# Load the SAM model

def load\_sam\_model():

    sam\_checkpoint = "/content/sam\_vit\_h\_4b8939.pth"  # Correct path after downloading the model

    model\_type = "vit\_h"

    sam = sam\_model\_registry[model\_type](checkpoint=sam\_checkpoint)

    return sam

def segment\_object(image\_path, object\_class, output\_path, azimuth\_angle=0, polar\_angle=0):

    # Load SAM model

    sam = load\_sam\_model()

    predictor = SamPredictor(sam)

    # Read the image

    image = cv2.imread(image\_path)

    predictor.set\_image(image)

    # Dummy object prompt for bounding box (Replace with actual detection model for the object class)

    # This example uses a static bounding box; in a real scenario, you would replace it with automatic detection

    box = np.array([50, 50, 300, 300])  # Example bounding box for testing

    # Predict mask for the detected object

    masks, \_, \_ = predictor.predict(box=box)

    # Apply the mask on the image

    mask\_image = np.zeros\_like(image)

    mask\_image[masks[0]] = [0, 0, 255]  # Red color mask for the object

    # Get the object mask

    object\_mask = np.zeros\_like(image)

    object\_mask[masks[0]] = image[masks[0]]

    # Rotate the object

    (h, w) = object\_mask.shape[:2]

    center = (w // 2, h // 2)

    M = cv2.getRotationMatrix2D(center, azimuth\_angle, 1.0)

    rotated\_object = cv2.warpAffine(object\_mask, M, (w, h))

    # Create a mask for the rotated object

    rotated\_mask = np.zeros\_like(rotated\_object)

    rotated\_mask[rotated\_object != 0] = 255

    # Remove the rotated object from the original image

    image\_without\_object = np.copy(image)

    image\_without\_object[masks[0]] = 0

    # Add the rotated object to the original image

    output\_image = np.copy(image\_without\_object)

    output\_image[rotated\_mask != 0] = rotated\_object[rotated\_mask != 0]

    # Save the output

    cv2.imwrite(output\_path, output\_image)

    print(f"Segmentation result saved to {output\_path}")

# Example usage

image\_path = '/content/chair.jpg'  # Path to your input image

object\_class = 'chair'  # The class of the object you want to segment

output\_path = '/content/rotated.png'  # Output file path

azimuth\_angle = 72  # Azimuth angle of rotation

polar\_angle = 0  # Polar angle of rotation (not used in this example)

segment\_object(image\_path, object\_class, output\_path, azimuth\_angle, polar\_angle)



import torch

from segment\_anything import sam\_model\_registry, SamPredictor

import cv2

import numpy as np

# Load the SAM model

def load\_sam\_model():

    sam\_checkpoint = "/content/sam\_vit\_h\_4b8939.pth"  # Correct path after downloading the model

    model\_type = "vit\_h"

    sam = sam\_model\_registry[model\_type](checkpoint=sam\_checkpoint)

    return sam

def segment\_object(image\_path, object\_class, output\_path, azimuth\_angle=0, polar\_angle=0):

    # Load SAM model

    sam = load\_sam\_model()

    predictor = SamPredictor(sam)

    # Read the image

    image = cv2.imread(image\_path)

    predictor.set\_image(image)

    # Dummy object prompt for bounding box (Replace with actual detection model for the object class)

    # This example uses a static bounding box; in a real scenario, you would replace it with automatic detection

    box = np.array([50, 50, 300, 300])  # Example bounding box for testing

    # Predict mask for the detected object

    masks, \_, \_ = predictor.predict(box=box)

    # Apply the mask on the image

    mask\_image = np.zeros\_like(image)

    mask\_image[masks[0]] = [0, 0, 255]  # Red color mask for the object

    # Get the object mask

    object\_mask = np.zeros\_like(image)

    object\_mask[masks[0]] = image[masks[0]]

    # Get the bounding box of the object

    x, y, w, h = box

    # Crop the object from the original image

    object\_image = image[y:y+h, x:x+w]

    # Rotate the object

    (h, w) = object\_image.shape[:2]

    center = (w // 2, h // 2)

    M = cv2.getRotationMatrix2D(center, azimuth\_angle, 1.0)

    rotated\_object = cv2.warpAffine(object\_image, M, (w, h))

    # Create a new image with the same size as the original image

    new\_image = np.copy(image)

    # Remove the original object from the new image

    new\_image[y:y+h, x:x+w] = 0

    # Get the bounding box of the rotated object

    (h, w) = rotated\_object.shape[:2]

    x = (image.shape[1] - w) // 2

    y = (image.shape[0] - h) // 2

    # Add the rotated object to the new image

    new\_image[y:y+h, x:x+w] = rotated\_object

    # Save the output

    cv2.imwrite(output\_path, new\_image)

    print(f"Segmentation result saved to {output\_path}")

# Example usage

image\_path = '/content/chair.jpg'  # Path to your input image

object\_class = 'chair'  # The class of the object you want to segment

output\_path = '/content/rotated.png'  # Output file path

azimuth\_angle = 72  # Azimuth angle of rotation

polar\_angle = 0  # Polar angle of rotation (not used in this example)

segment\_object(image\_path, object\_class, output\_path, azimuth\_angle, polar\_angle)

