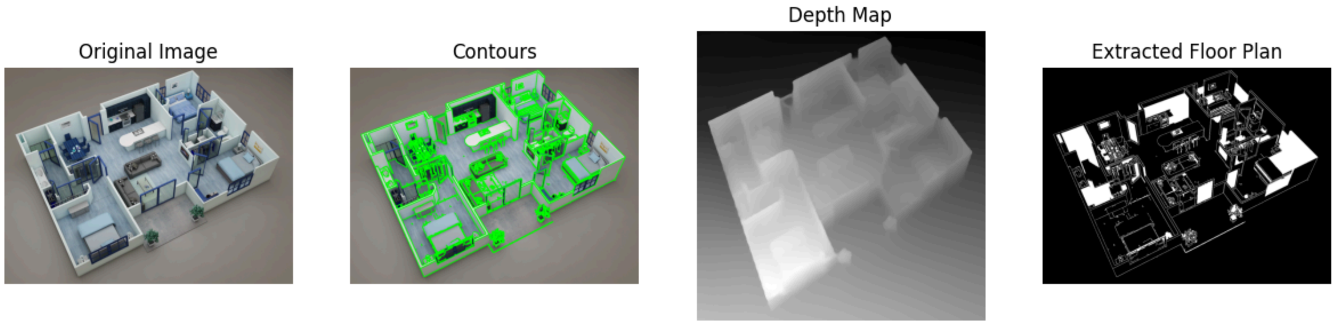
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import torch
import torchvision.transforms as T
from PIL import Image

# Load MiDaS model
midas_model = torch.hub.load("intel-isl/MiDaS", "DPT_Hybrid")

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
midas_model.to(device)
midas_model.eval()

def load_image(image_path):
    """Load an image from the specified path."""
    try:
        image = cv2.imread(image_path)
        if image is None:
            raise FileNotFoundError(f"Image not found at {image_path}")
        return image
    except Exception as e:
        print(f"Error loading image: {e}")
        return None

def preprocess_image(image):
    """Preprocess the image to fit the requirements of MiDaS model."""
    # Convert to RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Resize to fit model input size
    image_resized = cv2.resize(image_rgb, (384, 384))
    # Normalize the image
    image_normalized = image_resized / 255.0  # Normalize pixel values to [0, 1]
    # Convert to tensor and add batch dimension
    image_tensor = torch.tensor(image_normalized, dtype=torch.float32).permute(2, 0, 1).unsqueeze(0)
    return image_tensor

def predict_depth(image_tensor):
    """Predict depth map."""
    # Predict depth map
    with torch.no_grad():
        prediction = midas_model(image_tensor.to(device))
    depth_map = prediction.squeeze().cpu().numpy()
    return depth_map

def detect_edges(image):
    """Detect edges using the Canny edge detection algorithm."""
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray_image, 50, 150)
    return edges

def find_contours(image):
    """Find contours in the image."""
    contours, _ = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    return contours

def draw_contours(image, contours):
    """Draw contours on the image."""
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
    return contour_image

def extract_floor_plan(edges):
    """Extract floor plan from edge-detected image."""
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    floor_plan = np.zeros_like(edges)
    cv2.drawContours(floor_plan, contours, -1, (255), thickness=cv2.FILLED)
    return floor_plan

def display_images(images, titles):
    """Display multiple images side by side."""
    plt.figure(figsize=(15, 5))
    for i in range(len(images)):
        plt.subplot(1, len(images), i + 1)
        if len(images[i].shape) == 2:
            plt.imshow(images[i], cmap='gray')
        else:
            plt.imshow(images[i])
        plt.title(titles[i])
        plt.axis('off')
    plt.show()

def process_image(image_path):
    """Process the image to predict depth map and extract floor plan."""
    # Load the image
    image = load_image(image_path)
    if image is None:
        return

    # Preprocess the image for MiDaS model
    image_tensor = preprocess_image(image)

    # Predict depth map
    depth_map = predict_depth(image_tensor)

    # Detect edges
    edges = detect_edges(image)

    # Find contours
    contours = find_contours(edges)

    # Draw contours on the original image
    contour_image = draw_contours(image, contours)

    # Extract floor plan from edges
    floor_plan = extract_floor_plan(edges)

    # Display results
    display_images([image, contour_image, depth_map, floor_plan],
                   ['Original Image', 'Contours', 'Depth Map', 'Extracted Floor Plan'])

if __name__ == "__main__":
    # Provide the path to the image
    image_path = '/content/FP1.jpg'  # Change this to the path of your image

    # Process the image
    process_image(image_path)
```

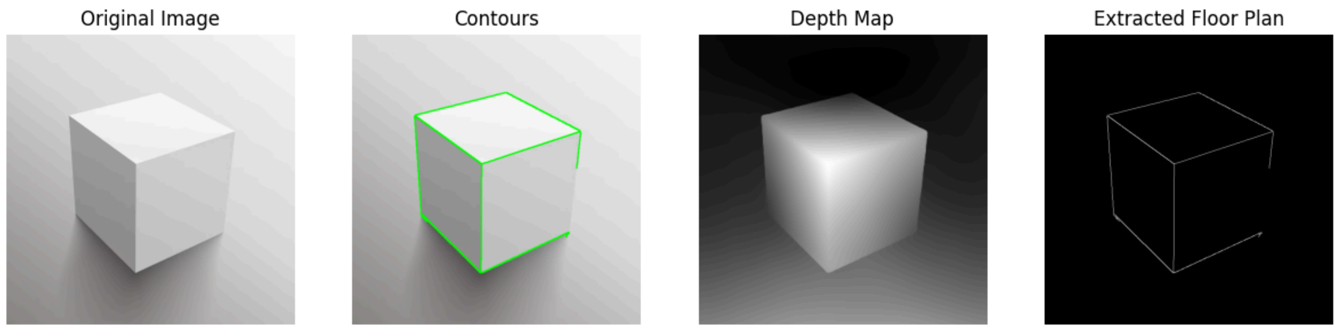
```
Using cache found in /root/.cache/torch/hub/intel-isl_MiDaS_master
```



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import torch
import torchvision.transforms as T
from PIL import Image

# Load MiDaS model
midas_model = torch.hub.load("intel-isl/MiDaS", "DPT_Hybrid")

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
midas_model.to(device)
midas_model.eval()

def load_image(image_path):
    """Load an image from the specified path."""
    try:
        image = cv2.imread(image_path)
        if image is None:
            raise FileNotFoundError(f"Image not found at {image_path}")
        return image
    except Exception as e:
        print(f"Error loading image: {e}")
        return None

def preprocess_image(image):
    """Preprocess the image to fit the requirements of MiDaS model."""
    # Convert to RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Resize to fit model input size
    image_resized = cv2.resize(image_rgb, (384, 384))
    # Normalize the image
    image_normalized = image_resized / 255.0  # Normalize pixel values to [0, 1]
    # Convert to tensor and add batch dimension
    image_tensor = torch.tensor(image_normalized, dtype=torch.float32).permute(2, 0, 1).unsqueeze(0)
    return image_tensor

def predict_depth(image_tensor):
    """Predict depth map."""
    # Predict depth map
    with torch.no_grad():
        prediction = midas_model(image_tensor.to(device))
    depth_map = prediction.squeeze().cpu().numpy()
    return depth_map

def detect_edges(image):
    """Detect edges using the Canny edge detection algorithm."""
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray_image, 50, 150)
    return edges

def find_contours(image):
    """Find contours in the image."""
    contours, _ = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    return contours

def draw_contours(image, contours):
    """Draw contours on the image."""
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
    return contour_image

def extract_floor_plan(edges):
    """Extract floor plan from edge-detected image."""
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    floor_plan = np.zeros_like(edges)
    cv2.drawContours(floor_plan, contours, -1, (255), thickness=cv2.FILLED)
    return floor_plan

def display_images(images, titles):
    """Display multiple images side by side."""
    plt.figure(figsize=(15, 5))
    for i in range(len(images)):
        plt.subplot(1, len(images), i + 1)
        if len(images[i].shape) == 2:
            plt.imshow(images[i], cmap='gray')
        else:
            plt.imshow(images[i])
        plt.title(titles[i])
        plt.axis('off')
    plt.show()
```

```
def process_image(image_path):
    """Process the image to predict depth map and extract floor plan."""
    # Load the image
    image = load_image(image_path)
    if image is None:
        return

    # Preprocess the image for MiDaS model
    image_tensor = preprocess_image(image)

    # Predict depth map
    depth_map = predict_depth(image_tensor)

    # Detect edges
    edges = detect_edges(image)

    # Find contours
    contours = find_contours(edges)

    # Draw contours on the original image
    contour_image = draw_contours(image, contours)

    # Extract floor plan from edges
    floor_plan = extract_floor_plan(edges)

    # Display results
    display_images([image, contour_image, depth_map, floor_plan],
                   ['Original Image', 'Contours', 'Depth Map', 'Extracted Floor Plan'])

if __name__ == "__main__":
    # Provide the path to the image
    image_path = '/content/box-mockup.jpg'  # Change this to the path of your image

    # Process the image
    process_image(image_path)
```

```
Using cache found in /root/.cache/torch/hub/intel-isl_MiDaS_master
```


Original Image
Contours
Depth Map
Extracted Floor Plan