CS445: Computational Photography

Final Project: Daily Activities of cats

Load libraries and data

In [1]:

```
# jupyter extension that allows reloading functions from imports without clearing kernel
%load_ext autoreload
%autoreload 2
```

In [2]:

```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [3]:

```
# system imports
import os
from os import path
import math

# third-party Imports
import cv2
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import griddata

%matplotlib inline
from random import random
import time
import scipy
import scipy.sparse.linalg

from IPython.display import HTML
from IPython.display import Video
import shutil
import warnings

# modify to where you store your project data including utils
datadir = "C:/Users/chaob/Desktop/445/finalproject/fp/FinalProject"

utilfn = datadir + "utils"
# !cp -r "$utilfn" .
samplesfn = datadir + "samples"
# !cp -r "$samplesfn" .

# can change this to your output directory of choice
# !mkdir "images"
# !mkdir "images/outputs"

# import starter code
import utils
from utils.io import read_image, write_image, read_hdr_image, write_hdr_image
from utils.display import display_images_linear_rescale, rescale_images_linear
from utils.hdr_helpers import gsolve
from utils.hdr_helpers import get_equirectangular_image
from utils.bilateral_filter import bilateral_filter
from utils.bilateral_filter import bilateral_filter
import utils.points
```

Reading Videos

In [4]:

```python
def import_video(video_path):

    # create a directory to store the extracted frames
    output_folder = 'frames'+ '/' + vname

    # check if the directory exists
    if os.path.exists(output_folder):
        # if it exists, remove the directory and its contents
        shutil.rmtree(output_folder)
    os.makedirs(output_folder, exist_ok=True)

    # open the video file
    video = cv2.VideoCapture(video_path)

    # get the video properties
    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(video.get(cv2.CAP_PROP_FPS))
    total_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))

    # initialize variables
    frame_count = 0
    frames = []

    # read frames from the video
    while True:
        ret, frame = video.read()
        if not ret:
            break

        # save the frame as an image in the output folder
        frame_path = os.path.join(output_folder, f"frame_{frame_count:04d}.jpg")
        cv2.imwrite(frame_path, frame)

        # append the frame to the list
        frames.append(frame)

        frame_count += 1

    # release the video object
    video.release()

    print(f"Total frames: {total_frames}")
    print(f"Frames extracted: {frame_count}")
    print(f"Frames saved in the folder: {output_folder}")

    return frames, width, height, fps
```

In [5]:

```python
imdir = 'samples'
vname = 'cat1'

# specify the path to the video file
video_path = imdir + '/' + vname +'.mp4'

# import the video and get the frames and video properties
frames, width, height, fps = import_video(video_path)
```

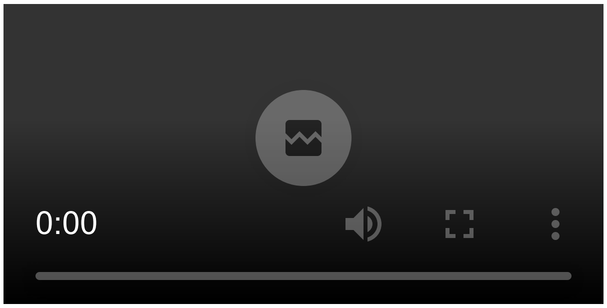
```
Total frames: 144
Frames extracted: 144
Frames saved in the folder: frames/cat1
```

In [6]:

```python
# display the video in the notebook
video = Video(video_path)
```

```
print(video_path)
display(video)
```

```
samples/cat1.mp4
```



In [7]:

```
# first frame is background, or can be frames[0]
background_image_file = 'frames'+ '/' + vname + '/' +'frame_0000.jpg'
background_image = read_image(background_image_file)

# background_image_file = imdir + '/' + 'empty.jpg'
# background_image = read_image(background_image_file)
```

Background removal

In [8]:

```
def background_subtraction_p(frames, width, height, fps, vname):
    # create directories to store the output frames and video
    output_folder_bs = "background_removed_frames_p" + '/' + vname

    # check if the directory exists
    if os.path.exists(output_folder_bs):
        # if it exists, remove the directory and its contents
        shutil.rmtree(output_folder_bs)
    os.makedirs(output_folder_bs, exist_ok=True)

    # read the first frame as the background image
    background = cv2.cvtColor(frames[0], cv2.COLOR_BGR2GRAY)

    # apply GaussianBlur to the background frame
    blurred_background = cv2.GaussianBlur(background, (15, 15), 0)

    # initialize variables
    processed_frames = []

    # create a kernel
    kernel = np.ones((20, 20), np.uint8)

    # process frames for background subtraction
    for i, frame in enumerate(frames):
        # convert the frame to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # apply GaussianBlur to the grayscale frame
        blurred_frame = cv2.GaussianBlur(gray_frame, (15, 15), 0)

        # perform background subtraction
        diff = cv2.absdiff(blurred_background, blurred_frame)

        # apply thresholding
        threshold = 60
        _, thresh = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

        # save the thresholded frame as an image in the output folder
        frame_path_bs = os.path.join(output_folder_bs, f"frame_{i:04d}.jpg")
        cv2.imwrite(frame_path_bs, thresh)
        # append the processed frame to the list
        processed_frames.append(thresh)

    print(f"Background subtraction completed.")
    print(f"Output frames saved in the folder: {output_folder_bs}")

    return processed_frames, output_folder_bs
```

In [9]:

```
# perform background subtraction on the frames and get the processed frames
processed_frames_p, output_folder_bs_p = background_subtraction_p(frames, width, height,
```

```
Background subtraction completed.
Output frames saved in the folder: background_removed_frames_p/cat1
```

In [10]:

```
%%capture
!ffmpeg -framerate 30 -i "{output_folder_bs_p}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv
```

In [11]:

```
def background_subtraction_o(frames, width, height, fps, vname):
    # create directories to store the output frames and video
    output_folder_bs = "background_removed_frames_o" + '/' + vname

    # check if the directory exists
    if os.path.exists(output_folder_bs):
        # if it exists, remove the directory and its contents
        shutil.rmtree(output_folder_bs)
    os.makedirs(output_folder_bs, exist_ok=True)

    # read the first frame as the background image
    background = cv2.cvtColor(frames[0], cv2.COLOR_BGR2GRAY)

    # apply GaussianBlur to the background frame
    blurred_background = cv2.GaussianBlur(background, (15, 15), 0)

    # initialize variables
    processed_frames = []

    # create a kernel
    kernel = np.ones((5, 5), np.uint8)

    # process frames for background subtraction
    for i, frame in enumerate(frames):
        # convert the frame to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # apply GaussianBlur to the grayscale frame
        blurred_frame = cv2.GaussianBlur(gray_frame, (15, 15), 0)

        # perform background subtraction
        diff = cv2.absdiff(blurred_background, blurred_frame)

        # apply thresholding
        threshold = 20
        _, thresh = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

        # apply erosion
        eroded = cv2.erode(thresh, kernel, iterations=2)

        # # apply dilation
        dilated = cv2.dilate(eroded, kernel, iterations=2)

        # save the thresholded frame as an image in the output folder
        frame_path_bs = os.path.join(output_folder_bs, f"frame_{i:04d}.jpg")
        cv2.imwrite(frame_path_bs, dilated)
        # append the processed frame to the list
        processed_frames.append(thresh)

    print(f"Background subtraction completed.")
    print(f"Output frames saved in the folder: {output_folder_bs}")

    return processed_frames, output_folder_bs
```

In [12]:

```
# perform background subtraction on the frames and get the processed frames
processed_frames_o, output_folder_bs_o = background_subtraction_o(frames, width, height,
```

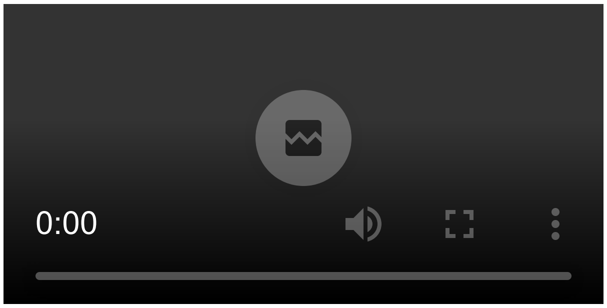
```
Background subtraction completed.
Output frames saved in the folder: background_removed_frames_o/cat1
```

In [13]:

```
%%capture
!ffmpeg -framerate 30 -i "{output_folder_bs_o}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv
```

In [14]:

```
# # Display the video in the notebook
v="background_removed_frames_o/cat1/bs_video.mp4"
video = Video(v,embed=True)
display(video)
```



In [15]:

```
def background_subtraction_improve(frames, width, height, fps, vname, kernel,GaussianBlu
    # create directories to store the output frames and video
    output_folder_bs = "background_removed_frames_improve" + '/' + vname

    # check if the directory exists
    if os.path.exists(output_folder_bs):
        # if it exists, remove the directory and its contents
        shutil.rmtree(output_folder_bs)
    os.makedirs(output_folder_bs, exist_ok=True)

    # initialize variables
    processed_frames = []

#     # create a kernel for morphological operations
#     kernel = np.ones((5, 5), np.uint8)

    # create a GMM-based background subtractor
    background_subtractor = cv2.createBackgroundSubtractorKNN(detectShadows=True)

    # define the number of initial frames to skip or use for training
    num_initial_frames = 5

    # train the background subtractor on initial frames
    for i in range(num_initial_frames):
        frame = frames[i]
        blurred_frame = cv2.GaussianBlur(frame, (GaussianBlurframe, GaussianBlurframe),
        _ = background_subtractor.apply(blurred_frame)

    # process frames for background subtraction
    for i, frame in enumerate(frames):

        blurred_frame = cv2.GaussianBlur(frame, (GaussianBlurframe, GaussianBlurframe),
        # apply background subtraction
        mask = background_subtractor.apply(blurred_frame)

        # apply thresholding to the mask
        _, thresh = cv2.threshold(mask, 250, 255, cv2.THRESH_BINARY)


        # apply erosion
        eroded = cv2.erode(thresh, kernel, iterations=i1)

        # # apply dilation
        dilated = cv2.dilate(eroded, kernel, iterations=i2)

        # save the thresholded frame as an image in the output folder
        frame_path_bs = os.path.join(output_folder_bs, f"frame_{i:04d}.jpg")
        cv2.imwrite(frame_path_bs, dilated)

        # append the processed frame to the list
        processed_frames.append(thresh)

    print(f"Background subtraction completed.")
    print(f"Output frames saved in the folder: {output_folder_bs}")

    return processed_frames, output_folder_bs
```

In [16]:

```
# create a kernel for morphological operations
kernel = np.ones((5, 5), np.uint8)

GaussianBlurframe=15

#erode and dilate iteration
i1=1
i2=3

# perform background subtraction on the frames and get the processed frames
processed_frames_improve, output_folder_bs_improve = background_subtraction_improve(fram
```

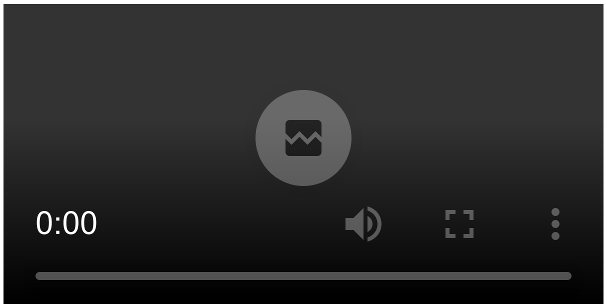
```
Background subtraction completed.
Output frames saved in the folder: background_removed_frames_improve/cat1
```

In [17]:

```
%%capture
!ffmpeg -framerate 30 -i "{output_folder_bs_improve}/frame_%04d.jpg" -c:v libx264 -pix_f
```

In [18]:

```
# # Display the video in the notebook
v="background_removed_frames_improve/cat1/bs_video.mp4"
video = Video(v,embed=True)
display(video)
```



Remove not interested objects

In [19]:

```
def filter_white_regions(image_path, thres):
    # read the image in grayscale
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # find contours in the binary image
    contours, _ = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # check if any contours were found
    if len(contours) > 0:
        # find the area of the largest contour
        largest_area = max(cv2.contourArea(contour) for contour in contours)

        # set the area threshold
        area_threshold = largest_area - 0.05*largest_area

        # create a mask to store the filtered white regions
        mask = np.zeros_like(image)

        # loop contour
        for contour in contours:
            # calculate the area of the contour
            area = cv2.contourArea(contour)

            # if the area is greater than or equal to the threshold, draw the contour on
            if area > thres and area >= area_threshold:
                cv2.drawContours(mask, [contour], 0, 255, -1)

        # apply the mask to the original image
        result = cv2.bitwise_and(image, image, mask=mask)
    else:
        # if no contours were found, return the original image
        result = image

    return result
```

In [20]:

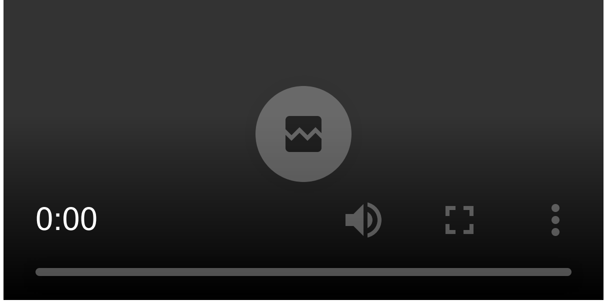
```
# define the folder path containing the black and white images
folder_path = output_folder_bs_improve

# create a new folder to store the processed images
out_folder = "obj_frames_improve" + '/' + vname
# check if the directory exists
if os.path.exists(out_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(out_folder)
os.makedirs(out_folder, exist_ok=True)

thres=2000

# loop image file in the folder
for filename in os.listdir(folder_path):
    if filename.endswith(".jpg") :  # adjust the file extensions as needed
        image_path = os.path.join(folder_path, filename)

        # process the image to keep the white regions with areas greater than or equal t
        processed_image = filter_white_regions(image_path, thres)

        # save the processed image to the output folder
        output_path = os.path.join(out_folder, filename)
        cv2.imwrite(output_path, processed_image)
```

In [21]:

```
%%capture
!ffmpeg -framerate 30 -i "{out_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p "{o
```

In [22]:

```
# # Display the video in the notebook
v="obj_frames_improve/cat1/obj_video.mp4"
video = Video(v,embed=True)
display(video)
```



putdown marker

In [23]:

```python
def find_connected_components(mask, max_distance, min_area_threshold):
    # threshold the mask to convert it into binary
    mask = cv2.threshold(mask, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

    # find connected components and their labels
    ret, labels = cv2.connectedComponents(mask)
    unique_labels, counts = np.unique(labels, return_counts=True)

    # create label map and rectangle map
    label_map = {label: label for label in unique_labels if label != 0}
    rectangle_map = {}

    # loop connected components
    for label, count in zip(unique_labels, counts):
        if label == 0:  # background
            continue

        if count < min_area_threshold:
            labels[labels == label] = 0  # set labels with small counts to 0 (black)
            continue

        # get the mask of the current component
        component_mask = (labels == label).astype(np.uint8)
        contours, _ = cv2.findContours(component_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPR

        # loop through contours
        for i in range(len(contours)):
            x1, y1, w1, h1 = cv2.boundingRect(contours[i])
            rect1 = np.array([[x1, y1], [x1 + w1, y1], [x1 + w1, y1 + h1], [x1, y1 + h1]

            for j in range(i+1, len(contours)):
                x2, y2, w2, h2 = cv2.boundingRect(contours[j])
                rect2 = np.array([[x2, y2], [x2 + w2, y2], [x2 + w2, y2 + h2], [x2, y2 +

                # calculate distances between the contours
                distances = [cv2.pointPolygonTest(rect1, tuple(point), True) for point i
                # check if any distance is within the maximum distance threshold
                if any(abs(dist) <= max_distance for dist in distances):
                    component_mask = cv2.drawContours(component_mask, contours, j, 1, -1

        # update labels and rectangle map for the component
        labels[component_mask == 1] = label
        rectangle_map[label] = rect1


    for label1, rect1 in rectangle_map.items():
        for label2, rect2 in rectangle_map.items():
            if label1 >= label2:
                continue

            if len(rect2)>0:
```

```
#                 print(rect1,"and",rect2)
                distances = [cv2.pointPolygonTest(rect1, tuple([int(round(point[0]) ), i
                if any(abs(dist) <= max_distance for dist in distances):
                    min_label = min(label_map[label1], label_map[label2])
                    max_label = max(label_map[label1], label_map[label2])
                    label_map[max_label] = min_label

    # update labels based on merged components
    new_labels = np.zeros_like(labels)
    for old_label, new_label in label_map.items():
        new_labels[labels == old_label] = label_map[new_label]

    return new_labels
```

In [24]:
```
def draw_bounding_boxes(image, labels):
    # apply thresholding
#     thresholded_image = cv2.threshold(labels, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_O

    unique_labels = np.unique(labels)
#     print(unique_labels)

    for label in unique_labels:
        if label == 0:  # Background
            continue
        # create a mask for the current component
        component_mask = (labels == label).astype(np.uint8)

        # find the indices of non-zero pixels in the component mask
        non_zero_indices = np.nonzero(component_mask)

        if len(non_zero_indices[0]) > 0:
            # find the minimum and maximum values of x and y coordinates
            y_min = np.min(non_zero_indices[0])
            y_max = np.max(non_zero_indices[0])
            x_min = np.min(non_zero_indices[1])
            x_max = np.max(non_zero_indices[1])

            # draw the bounding box rectangle
            cv2.rectangle(image, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
            cv2.putText(image, "cat", (x_min, y_min-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,

    return image
```

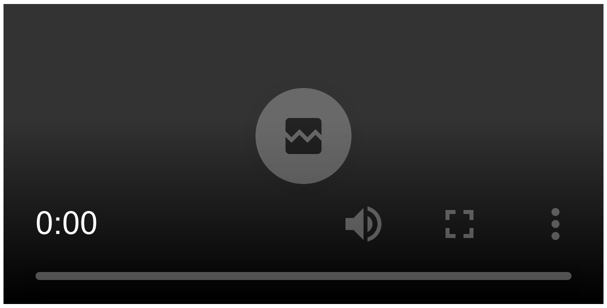
In [25]:
```
# specify the folder paths
original_frames_folder = 'frames/cat1'
white_frames_folder = 'obj_frames_improve/cat1'
marked_folder = 'marked/cat1'

# check if the directory exists
if os.path.exists(marked_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(marked_folder)
# create the output folder if it doesn't exist
os.makedirs(marked_folder, exist_ok=True)

# set the maximum distance threshold for connecting components
max_distance = 100
min_area_threshold=2000

# get the list of frame filenames in the original frames folder
original_frame_filenames = sorted(os.listdir(original_frames_folder))

# iterate through each frame filename
for filename in original_frame_filenames:
    if filename.endswith(".jpg") or filename.endswith(".png"):
        mask_path = os.path.join(white_frames_folder, filename)
        original_path = os.path.join(original_frames_folder, filename)
        output_path = os.path.join(marked_folder, filename)

        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        original = cv2.imread(original_path)

        labels = find_connected_components(mask, max_distance, min_area_threshold)
        result = draw_bounding_boxes(original, labels)

        cv2.imwrite(output_path, result)
```

In [26]:

```
%%capture
!ffmpeg -framerate 30 -i "{marked_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p
```

In [27]:

```
# # Display the video in the notebook
v="marked/cat1/marked_video.mp4"
video = Video(v,embed=True)
display(video)
```



second example

In [28]:

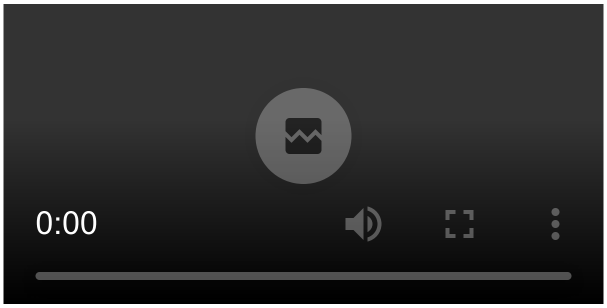
```
imdir = 'samples'
vname = 'cat2'

# specify the path to the video file
video_path = imdir + '/' + vname +'.mp4'

# import the video and get the frames and video properties
frames, width, height, fps = import_video(video_path)
```

```
Total frames: 129
Frames extracted: 129
Frames saved in the folder: frames/cat2
```

In [29]:

```
# display the video in the notebook
video = Video(video_path)
print(video_path)
display(video)
```

```
samples/cat2.mp4
```



In [30]:

```
# first frame is background, or can be frames[0]
```

```
background_image_file = 'frames'+ '/' + vname + '/' +'frame_0000.jpg'
background_image = read_image(background_image_file)

# background_image_file = imdir + '/' + 'empty.jpg'
# background_image = read_image(background_image_file)
```

In [31]:

```
# create a kernel for morphological operations
kernel = np.ones((20, 20), np.uint8)

GaussianBlurframe=25

i1=2
i2=2

# perform background subtraction on the frames and get the processed frames
processed_frames_improve, output_folder_bs_improve = background_subtraction_improve(fram
```

```
Background subtraction completed.
Output frames saved in the folder: background_removed_frames_improve/cat2
```

In [32]:

```
%%capture
!ffmpeg -framerate 30 -i "{output_folder_bs_improve}/frame_%04d.jpg" -c:v libx264 -pix_f
```

In [33]:

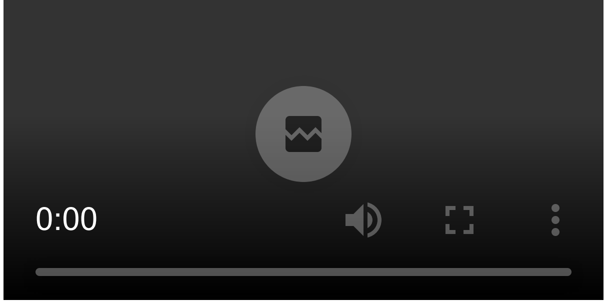
```
# specify the folder path containing the black and white images
folder_path = output_folder_bs_improve

# create a new folder to store the processed images
out_folder = "obj_frames_improve" + '/' + vname
# check if the directory exists
if os.path.exists(out_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(out_folder)
os.makedirs(out_folder, exist_ok=True)

thres=2000

# iterate over each image file in the folder
for filename in os.listdir(folder_path):
    if filename.endswith(".jpg") :  # Adjust the file extensions as needed
        image_path = os.path.join(folder_path, filename)

        # process the image to keep the white regions with areas greater than or equal t
        processed_image = filter_white_regions(image_path, thres)

        # save the processed image to the output folder
        output_path = os.path.join(out_folder, filename)
        cv2.imwrite(output_path, processed_image)
```

In [34]:

```
%%capture
!ffmpeg -framerate 30 -i "{out_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p "{o
```

In [35]:

```
# # Display the video in the notebook
v="obj_frames_improve/cat2/obj_video.mp4"
video = Video(v,embed=True)
display(video)
```



In [36]:

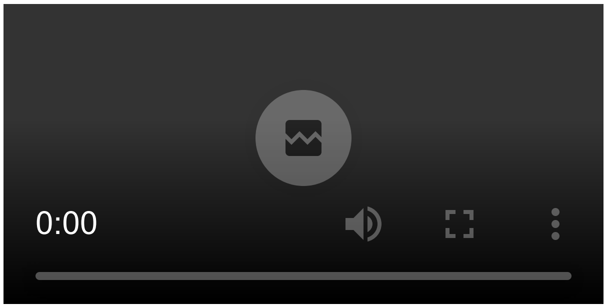
```
# specify the folder paths
original_frames_folder = 'frames/cat2'
white_frames_folder = 'obj_frames_improve/cat2'
marked_folder = 'marked/cat2'

# check if the directory exists
if os.path.exists(marked_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(marked_folder)
# create the output folder if it doesn't exist
os.makedirs(marked_folder, exist_ok=True)

# set the maximum distance threshold for connecting components
max_distance = 100
min_area_threshold=2000

# get the list of frame filenames in the original frames folder
original_frame_filenames = sorted(os.listdir(original_frames_folder))

# iterate through each frame filename
for filename in original_frame_filenames:
    if filename.endswith(".jpg") or filename.endswith(".png"):
        mask_path = os.path.join(white_frames_folder, filename)
        original_path = os.path.join(original_frames_folder, filename)
        output_path = os.path.join(marked_folder, filename)

        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        original = cv2.imread(original_path)

        labels = find_connected_components(mask, max_distance, min_area_threshold)
        result = draw_bounding_boxes(original, labels)

        cv2.imwrite(output_path, result)
```

In [37]:

```
%%capture
!ffmpeg -framerate 30 -i "{marked_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p
```

In [38]:

```
# # Display the video in the notebook
v="marked/cat2/marked_video.mp4"
video = Video(v,embed=True)
display(video)
```



check eating

In [39]:

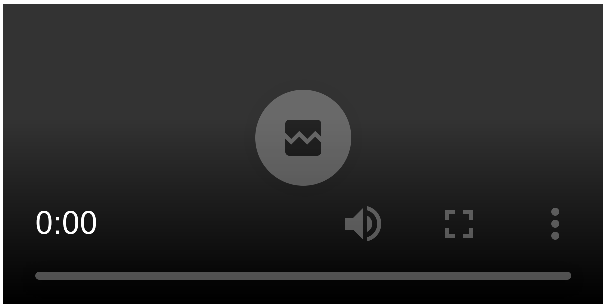
```
imdir = 'samples'
vname = 'catwithbowl'

# specify the path to the video file
video_path = imdir + '/' + vname +'.mp4'

# import the video and get the frames and video properties
frames, width, height, fps = import_video(video_path)
```

```
Total frames: 674
Frames extracted: 674
Frames saved in the folder: frames/catwithbowl
```

In [40]:

```
# display the video in the notebook
video = Video(video_path)
print(video_path)
display(video)
```

```
samples/catwithbowl.mp4
```



In [41]:

```
# first frame is background, or can be frames[0]
# background_image_file = 'frames'+ '/' + vname + '/' +'frame_0000.jpg'
# background_image = read_image(background_image_file)

# background_image_file = imdir + '/' + 'empty.jpg'
# background_image = read_image(background_image_file)
```

In [42]:

```
#color
im1_file_c = 'frames/catwithbowl/frame_0000.jpg'

im1_c = np.float32(cv2.imread(im1_file_c) / 255.0)

#convert BGR to RGB
im1_c = cv2.cvtColor(im1_c, cv2.COLOR_BGR2RGB)
```

In [43]:

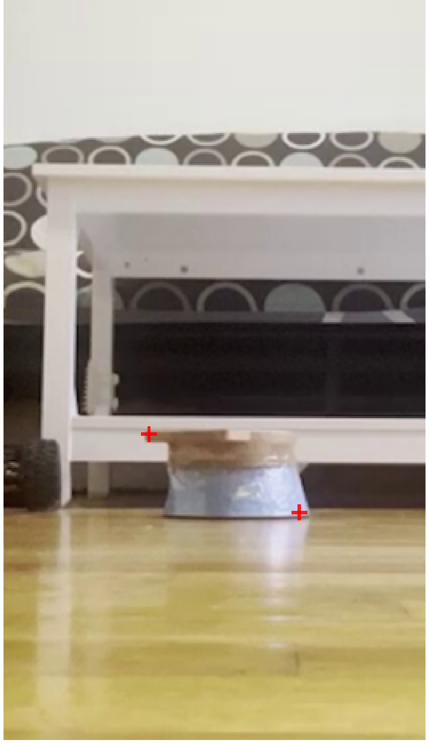
```
def get_bounding_box(xs, ys):
    x1, y1 = min(xs), min(ys)
    x2, y2 = max(xs), max(ys)
    return [int(y1), int(y2), int(x1), int(x2)]
```

In [44]:

```
def get_draw_mask(ys, xs, img):

    # get the bounding box coordinates
    y1, y2, x1, x2 = get_bounding_box(xs, ys)

    # draw the bounding box on the image
    fig, ax = plt.subplots(1, 1, figsize=(5, 5))

    img_with_bbox = img.copy()
    rect = plt.Rectangle((x1, y1), x2 - x1, y2 - y1, linewidth=2, edgecolor='r', facecol
    ax.imshow(img_with_bbox, cmap='gray')
    ax.add_patch(rect)
    ax.set_title('Image with Bounding Box')
    ax.axis('off')

    plt.show()

    return [y1, y2, x1, x2]
```

In [45]:
```
import matplotlib.pyplot as plt
%matplotlib notebook
pts = utils.points.specify_mask(im1_c)
```

```
If it doesn't get you to the drawing mode, then rerun this function again.
```



In [47]:
```
xs, ys = pts[0], pts[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
bowlcoord = get_draw_mask(ys, xs, im1_c)
```

```
<Figure size 640x480 with 0 Axes>
```

Image with Bounding Box

In [48]:
```
# create a kernel for morphological operations
kernel = np.ones((9, 9), np.uint8)

GaussianBlurframe=3

i1=1
i2=3

# perform background subtraction on the frames and get the processed frames
processed_frames_improve, output_folder_bs_improve = background_subtraction_improve(fram
```

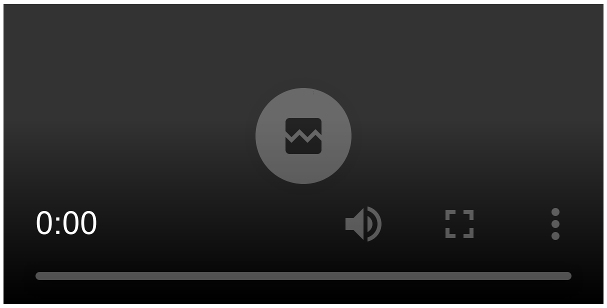
```
Background subtraction completed.
Output frames saved in the folder: background_removed_frames_improve/catwithbowl
```

In [49]:
```
%%capture
!ffmpeg -framerate 30 -i "{output_folder_bs_improve}/frame_%04d.jpg" -c:v libx264 -pix_f
```

In [50]:
```
# specify the folder path containing the black and white images
folder_path = output_folder_bs_improve

# create a new folder to store the processed images
out_folder = "obj_frames_improve" + '/' + vname
# check if the directory exists
if os.path.exists(out_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(out_folder)
os.makedirs(out_folder, exist_ok=True)

thres=100

# iterate over each image file in the folder
for filename in os.listdir(folder_path):
    if filename.endswith(".jpg") :  # Adjust the file extensions as needed
        image_path = os.path.join(folder_path, filename)

        # process the image to keep the white regions with areas greater than or equal t
        processed_image = filter_white_regions(image_path, thres)
        # save the processed image to the output folder
        output_path = os.path.join(out_folder, filename)
        cv2.imwrite(output_path, processed_image)
```

In [51]:
```
%%capture
!ffmpeg -framerate 30 -i "{out_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p "{o
```

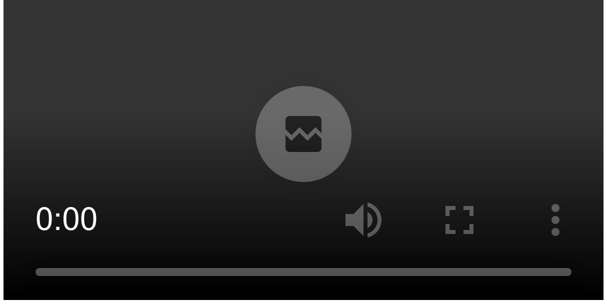
In [52]:
```
# # Display the video in the notebook
v="obj_frames_improve/catwithbowl/obj_video.mp4"
video = Video(v,embed=True)
display(video)
```



In [53]:
```
# specify the folder paths
original_frames_folder = 'frames/catwithbowl'
white_frames_folder = 'obj_frames_improve/catwithbowl'
marked_folder = 'marked/catwithbowl'

# check if the directory exists
if os.path.exists(marked_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(marked_folder)
# create the output folder if it doesn't exist
os.makedirs(marked_folder, exist_ok=True)

# set the maximum distance threshold for connecting components
max_distance = 50
min_area_threshold=100

# get the list of frame filenames in the original frames folder
original_frame_filenames = sorted(os.listdir(original_frames_folder))

# iterate through each frame filename
for filename in original_frame_filenames:
    if filename.endswith(".jpg") or filename.endswith(".png"):
        mask_path = os.path.join(white_frames_folder, filename)
        original_path = os.path.join(original_frames_folder, filename)
        output_path = os.path.join(marked_folder, filename)

        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        original = cv2.imread(original_path)

        labels = find_connected_components(mask, max_distance, min_area_threshold)
        result = draw_bounding_boxes(original, labels)

        cv2.imwrite(output_path, result)
```

In [54]:
```
%%capture
!ffmpeg -framerate 30 -i "{marked_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p
```

In [55]:
```
# # Display the video in the notebook
v="marked/catwithbowl/marked_video.mp4"
video = Video(v,embed=True)
display(video)
```



In [56]:

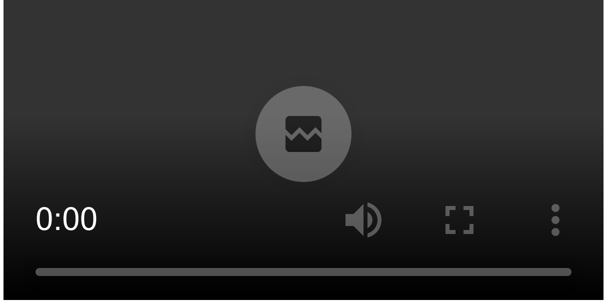
```
# specify the folder paths
# white_frames_folder
eat_folder = 'checkeat/catwithbowl'

# check if the directory exists
if os.path.exists(eat_folder):
    # if it exists, remove the directory and its contents
    shutil.rmtree(eat_folder)

# create the output folder if it doesn't exist
os.makedirs(eat_folder, exist_ok=True)

# set the maximum distance threshold for connecting components
margin = 50

# get the list of frame filenames in the original frames folder
mask_frame_filenames = sorted(os.listdir(white_frames_folder))

# iterate through each frame filename
for filename in mask_frame_filenames:
    if filename.endswith(".jpg") or filename.endswith(".png"):
        mask_path = os.path.join(white_frames_folder, filename)
        output_path = os.path.join(eat_folder, filename)

        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        frame = cv2.imread(os.path.join(original_frames_folder, filename))

        # check if the white area overlaps or contains the bowlcoord area
        y1, y2, x1, x2 = bowlcoord
        roi = mask[y1-margin:y2, x1:x2]
#         print(mask.shape)
#         print(bowlcoord, roi)

        if cv2.countNonZero(roi) > 0:
            # draw a bounding box around the bowlcoord area
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

            # add the "eating" tag to the frame
            cv2.putText(frame, "eating", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (

        cv2.imwrite(output_path, frame)
```

In [57]:

```
%%capture
!ffmpeg -framerate 30 -i "{eat_folder}/frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p "{e
```

In [58]:

```
# # Display the video in the notebook
v="checkeat/catwithbowl/eat_video.mp4"
video = Video(v,embed=True)
display(video)
```



3D Reconstruction

Building a 3D mobdel from 2D-image

Using a Spidery Mesh Interface to Make Animation from a Single Image

In [2]:
```
%%capture
#prepare the enviornment -- check the requirenments file
#image processing

#library import
import matplotlib
#matplotlib.use('TkAgg')
from matplotlib import pyplot as plt
#from PIL import Image
import PIL.Image
import torch
from transformers import GLPNImageProcessor, GLPNForDepthEstimation
from IPython.display import Image, display
```

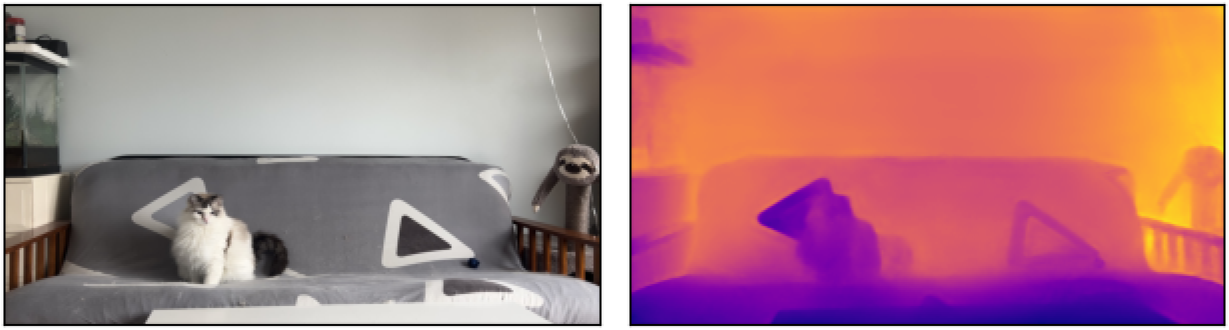
In [3]:
```
def resizeImage(width, height):
    new_height = 480 if height > 480 else height
    new_height -= (new_height % 32)
    new_width = int(new_height * width / height)
    diff = new_width % 32

    #resize
    new_width = new_width - diff if diff < 16 else new_width + 32 - diff
    new_size = (new_width, new_height)

    return new_size
```

In [4]:
```
#getting model
feature_extractor = GLPNImageProcessor.from_pretrained("vinvino02/glpn-nyu")
model = GLPNForDepthEstimation.from_pretrained("vinvino02/glpn-nyu")
```

In [5]:
```
#loading and resizing the image
image = PIL.Image.open("samples/2d_image.jpg")
new_size = resizeImage(image.width, image.height)
image = image.resize(new_size)

#show the image
display(image)
```

In [6]:

```
#prepare the image for the model
inputs = feature_extractor(images=image, return_tensors="pt")

#get the prediction from the model
with torch.no_grad():
    outputs = model(**inputs)
    predicted_depth = outputs.predicted_depth

#post-processing
pad = 16
output = predicted_depth.squeeze().cpu().numpy()*1000.0
output = output[pad:-pad, pad:-pad]
image = image.crop((pad,pad,image.width - pad, image.height - pad))

#visualize the prediction
fig, ax = plt.subplots(1,2)
ax[0].imshow(image)
ax[0].tick_params(left=False, bottom = False, labelleft=False, labelbottom=False)
ax[1].imshow(output,cmap='plasma')
ax[1].tick_params(left=False, bottom = False, labelleft=False, labelbottom=False)
plt.tight_layout()
plt.pause(5)
```



In [7]:

```
#import the libaries
import numpy as np
import open3d as o3d
```

```
def create3dObjectImprove(image):
```

In [8]:

```
    #prepare the depth image for open3d
    width, height = image.size
    depth_image = (output * 255 / np.max(output)).astype(np.uint8)
    image = np.array(image)

    #create rgbd image
    depth_o3d = o3d.geometry.Image(depth_image)
    image_o3d = o3d.geometry.Image(image)
    rgbd_image = o3d.geometry.RGBDImage.create_from_color_and_depth(image_o3d,depth_o3d,

    #create the camera
    camera_intrinsic = o3d.camera.PinholeCameraIntrinsic()
    camera_intrinsic.set_intrinsics(width,height,500,500,width/2,height/2)

    #create o3d point cloud
    pcd_raw = o3d.geometry.PointCloud.create_from_rgbd_image(rgbd_image, camera_intrinsi
    #o3d.visualization.draw_geometries([pcd_raw])
    #display((pcd)).to_html5_video()

    #post-processing the 3D point cloud

    #outline removal
    cl,ind = pcd_raw.remove_statistical_outlier(nb_neighbors=20, std_ratio=6.0)
    pcd = pcd_raw.select_by_index(ind)

    #estimate normals
    pcd.estimate_normals()
    pcd.orient_normals_to_align_with_direction()

    #surface reconstruction

    mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(pcd, depth=10,n_thr

    #rotate the mesh
    rotation = mesh.get_rotation_matrix_from_xyz((np.pi,0,0))
    mesh.rotate(rotation,center=(0,0,0))

    #if want to change color
    #mesh_uniform = mesh.paint_uniform_color([0.9,0.8,0.9])
    #mesh_uniform.compute_vertex_normals()


    return mesh
```

In [9]:

```
#create 3d object
mesh = create3dObjectImprove(image)

#3D Mesh Export
o3d.io.write_triangle_mesh("outputs/3dObject.obj",mesh)
```

Out[9]:

```
True
```

In [ ]:

```
#visualize the mesh, make the mesh show front
o3d.visualization.draw_geometries([mesh], mesh_show_back_face=True)
```

In [ ]: