sam2_example

June 24, 2025

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

---

[1]:
```
# Copyright (c) Meta Platforms, Inc. and affiliates.
```

[2]:
```
"""
First, convert mp4s to imagedirs
"""

import cv2
import os

def extract_frames(video_path, output_dir, n=None, frame_rate=1):
    """
    Extract frames from a video file and save them as images in a directory.

    Parameters:
    - video_path: Path to the video file.
    - output_dir: Directory to save extracted frames.
    - n: Maximum number of frames to extract (optional).
    - frame_rate: Extract every 'frame_rate'-th frame (default is 1, meaning␣
 ↪every frame).
    """
    # Check if the output directory exists, if not, create it
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Open the video file
    video_capture = cv2.VideoCapture(video_path)

    # Check if the video file was successfully opened
    if not video_capture.isOpened():
        print(f"Error: Could not open video file {video_path}")
        return
```

```
    frame_count = 0
    extracted_count = 0

    while True:
        # Read a frame from the video
        success, frame = video_capture.read()

        # Break the loop if there are no more frames
        if not success or (n is not None and extracted_count >= n):
            break

        # Only save frames that match the frame_rate condition
        if frame_count % frame_rate == 0:
            # Construct the filename for the frame
            frame_filename = os.path.join(output_dir, f"{extracted_count:04d}.
↪jpg")

            # Save the frame as an image file
            cv2.imwrite(frame_filename, frame)

            extracted_count += 1

        frame_count += 1

    # Release the video capture object
    video_capture.release()

    print(f"Extracted {extracted_count} frames to {output_dir}")

def process_videos(video_dir, output_base_dir, n=None, frame_rate=1):
    """
    Process all MP4 files in a directory and extract frames from each.

    Parameters:
    - video_dir: Directory containing MP4 files.
    - output_base_dir: Directory to save extracted frames.
    - n: Maximum number of frames to extract per video (optional).
    - frame_rate: Extract every 'frame_rate'-th frame (default is 1, meaning␣
↪every frame).
    """
    # Check if the video directory exists
    if not os.path.exists(video_dir):
        print(f"Error: Video directory {video_dir} does not exist")
        return

    # Check if the output base directory exists, if not, create it
    if not os.path.exists(output_base_dir):
```

```
        os.makedirs(output_base_dir)

    # Iterate through all MP4 files in the video directory
    for filename in os.listdir(video_dir):
        if filename.endswith(".mp4"):
            video_path = os.path.join(video_dir, filename)
            output_dir = os.path.join(output_base_dir, os.path.
 ↪splitext(filename)[0])  # Remove .mp4 extension

            print(f"Processing {video_path}...")
            extract_frames(video_path, output_dir, n, frame_rate)

# Example usage
video_dir = "/data1/Video/CVP/data/aldair/aldair_videos"  # Directory␣
 ↪containing MP4 files
output_base_dir = "/data1/Video/CVP/data/aldair/aldair_imagedirs"  # Directory␣
 ↪to store extracted images
n = 80  # Maximum number of frames to extract per video (optional)
frame_rate = 10  # Extract every 5th frame
process_videos(video_dir, output_base_dir, n, frame_rate)
```

```
Processing
/data1/Video/CVP/data/aldair/aldair_videos/CV_TrialRun_1_VialsNoColor_v1.mp4…
Extracted 80 frames to
/data1/Video/CVP/data/aldair/aldair_imagedirs/CV_TrialRun_1_VialsNoColor_v1
Processing /data1/Video/CVP/data/aldair/aldair_videos/CV_TrialRun_2_VialsWithCol
or_A2B_v1.mp4…
Extracted 80 frames to /data1/Video/CVP/data/aldair/aldair_imagedirs/CV_TrialRun
_2_VialsWithColor_A2B_v1
Processing /data1/Video/CVP/data/aldair/aldair_videos/CV_TrialRun_2_VialsWithCol
or_B2A_v1.mp4…
Extracted 80 frames to /data1/Video/CVP/data/aldair/aldair_imagedirs/CV_TrialRun
_2_VialsWithColor_B2A_v1
Processing /data1/Video/CVP/data/aldair/aldair_videos/CV_TrialRun_2_VialsWithCol
or_v1.mp4…
Extracted 80 frames to
/data1/Video/CVP/data/aldair/aldair_imagedirs/CV_TrialRun_2_VialsWithColor_v1
```

1 Video segmentation with SAM 2

This notebook shows how to use SAM 2 for interactive segmentation in videos. It will cover the following:

- adding clicks (or box) on a frame to get and refine *masklets* (spatio-temporal masks)
- propagating clicks (or box) to get *masklets* throughout the video
- segmenting and tracking multiple objects at the same time

We use the terms *segment* or *mask* to refer to the model prediction for an object on a single frame,

and *masklet* to refer to the spatio-temporal masks across the entire video.

1.1 Environment Set-up

If running locally using jupyter, first install `sam2` in your environment using the installation instructions in the repository.

If running from Google Colab, set `using_colab=True` below and run the cell. In Colab, be sure to select 'GPU' under 'Edit'->'Notebook Settings'->'Hardware accelerator'. Note that it's recommended to use **A100 or L4 GPUs when running in Colab** (T4 GPUs might also work, but could be slow and might run out of memory in some cases).

[3]:
```
using_colab = False
```

[4]:
```
if using_colab:
    import torch
    import torchvision
    print("PyTorch version:", torch.__version__)
    print("Torchvision version:", torchvision.__version__)
    print("CUDA is available:", torch.cuda.is_available())
    import sys
    !{sys.executable} -m pip install opencv-python matplotlib
    !{sys.executable} -m pip install 'git+https://github.com/facebookresearch/
 ↪sam2.git'

    !mkdir -p videos
    !wget -P videos https://dl.fbaipublicfiles.com/segment_anything_2/assets/
 ↪bedroom.zip
    !unzip -d videos videos/bedroom.zip

    !mkdir -p ../checkpoints/
    !wget -P ../checkpoints/ https://dl.fbaipublicfiles.com/segment_anything_2/
 ↪092824/sam2.1_hiera_large.pt
```

1.2 Set-up

[5]:
```
import os
# if using Apple MPS, fall back to CPU for unsupported ops
os.environ["PYTORCH_ENABLE_MPS_FALLBACK"] = "1"
import numpy as np
import torch
import matplotlib.pyplot as plt
from PIL import Image
```

[6]:
```
# select the device for computation
if torch.cuda.is_available():
    device = torch.device("cuda")
elif torch.backends.mps.is_available():
```

```
    device = torch.device("mps")
else:
    device = torch.device("cpu")
print(f"using device: {device}")

if device.type == "cuda":
    # use bfloat16 for the entire notebook
    torch.autocast("cuda", dtype=torch.bfloat16).__enter__()
    # turn on tfloat32 for Ampere GPUs (https://pytorch.org/docs/stable/notes/
 ↪cuda.html#tensorfloat-32-tf32-on-ampere-devices)
    if torch.cuda.get_device_properties(0).major >= 8:
        torch.backends.cuda.matmul.allow_tf32 = True
        torch.backends.cudnn.allow_tf32 = True
elif device.type == "mps":
    print(
        "\nSupport for MPS devices is preliminary. SAM 2 is trained with CUDA␣
 ↪and might "
        "give numerically different outputs and sometimes degraded performance␣
 ↪on MPS. "
        "See e.g. https://github.com/pytorch/pytorch/issues/84936 for a␣
 ↪discussion."
    )
```

```
using device: cuda
```

1.2.1 Loading the SAM 2 video predictor

[7]:
```
from sam2.build_sam import build_sam2_video_predictor

sam2_checkpoint = "../checkpoints/sam2.1_hiera_large.pt"
model_cfg = "configs/sam2.1/sam2.1_hiera_l.yaml"

predictor = build_sam2_video_predictor(model_cfg, sam2_checkpoint,␣
 ↪device=device)
```

[8]:
```
def show_mask(mask, ax, obj_id=None, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        cmap = plt.get_cmap("tab10")
        cmap_idx = 0 if obj_id is None else obj_id
        color = np.array([*cmap(cmap_idx)[:3], 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)
```

```
def show_points(coords, labels, ax, marker_size=200):
    pos_points = coords[labels==1]
    neg_points = coords[labels==0]
    ax.scatter(pos_points[:, 0], pos_points[:, 1], color='green', marker='*',␣
 ↪s=marker_size, edgecolor='white', linewidth=1.25)
    ax.scatter(neg_points[:, 0], neg_points[:, 1], color='red', marker='*',␣
 ↪s=marker_size, edgecolor='white', linewidth=1.25)


def show_box(box, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green', facecolor=(0,␣
 ↪0, 0, 0), lw=2))
```

Select an example video We assume that the video is stored as a list of JPEG frames with filenames like `<frame_index>.jpg`.

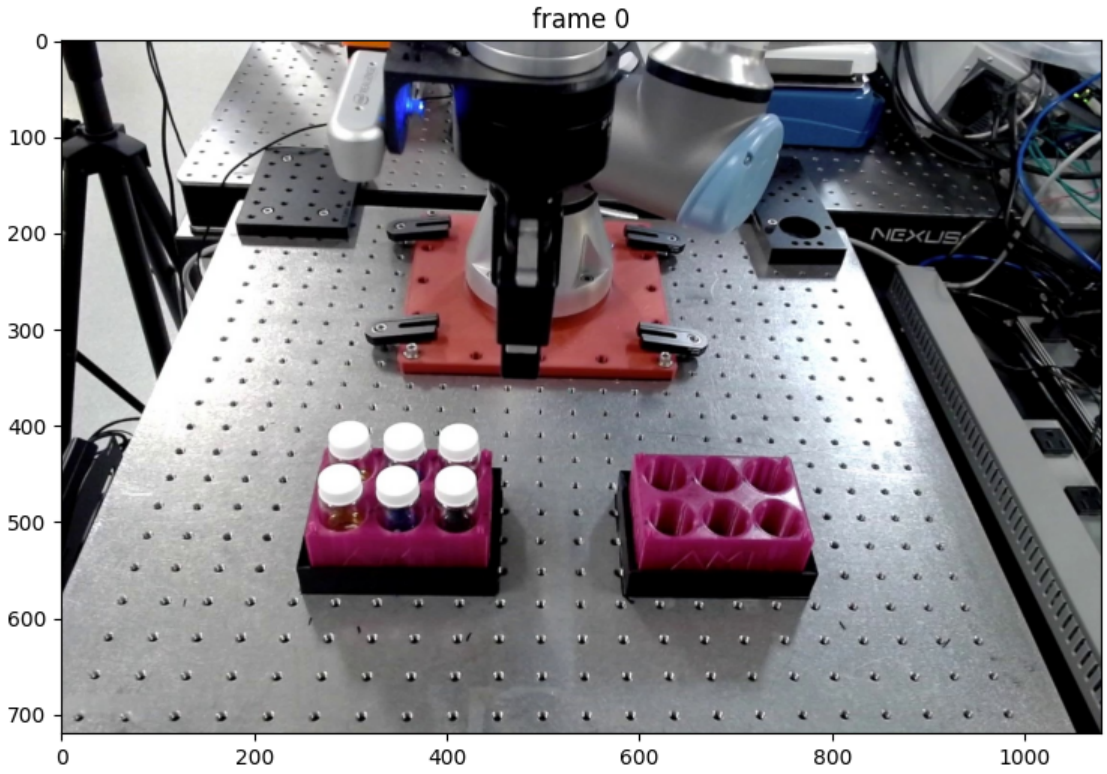
For your custom videos, you can extract their JPEG frames using ffmpeg (https://ffmpeg.org/) as follows:

```
ffmpeg -i <your_video>.mp4 -q:v 2 -start_number 0 <output_dir>/'%05d.jpg'
```

where `-q:v` generates high-quality JPEG frames and `-start_number 0` asks ffmpeg to start the JPEG file from `00000.jpg`.

[9]:
```
# `video_dir` a directory of JPEG frames with filenames like `<frame_index>.jpg`
#video_dir = "./videos/bedroom"
video_dir = '/data1/Video/CVP/data/aldair/aldair_imagedirs/
 ↪CV_TrialRun_2_VialsWithColor_A2B_v1/'

# scan all the JPEG frame names in this directory
frame_names = [
    p for p in os.listdir(video_dir)
    if os.path.splitext(p)[-1] in [".jpg", ".jpeg", ".JPG", ".JPEG"]
]
frame_names.sort(key=lambda p: int(os.path.splitext(p)[0]))

# take a look the first video frame
frame_idx = 0
plt.figure(figsize=(9, 6))
plt.title(f"frame {frame_idx}")
plt.imshow(Image.open(os.path.join(video_dir, frame_names[frame_idx])))
```

[9]:
```
<matplotlib.image.AxesImage at 0x7fb5fc1fd910>
```



Initialize the inference state SAM 2 requires stateful inference for interactive video segmentation, so we need to initialize an **inference state** on this video.

During initialization, it loads all the JPEG frames in `video_path` and stores their pixels in `inference_state` (as shown in the progress bar below).

[10]:
```
inference_state = predictor.init_state(video_path=video_dir)
```

```
frame loading (JPEG):   0%|          | 0/80 [00:00<?, ?it/s]
frame loading (JPEG): 100%|     | 80/80 [00:02<00:00, 28.84it/s]
```

1.2.2 Example 1: Segment & track one object

Note: if you have run any previous tracking using this `inference_state`, please reset it first via `reset_state`.

(The cell below is just for illustration; it's not needed to call `reset_state` here as this `inference_state` is just freshly initialized above.)

[11]:
```
predictor.reset_state(inference_state)
```

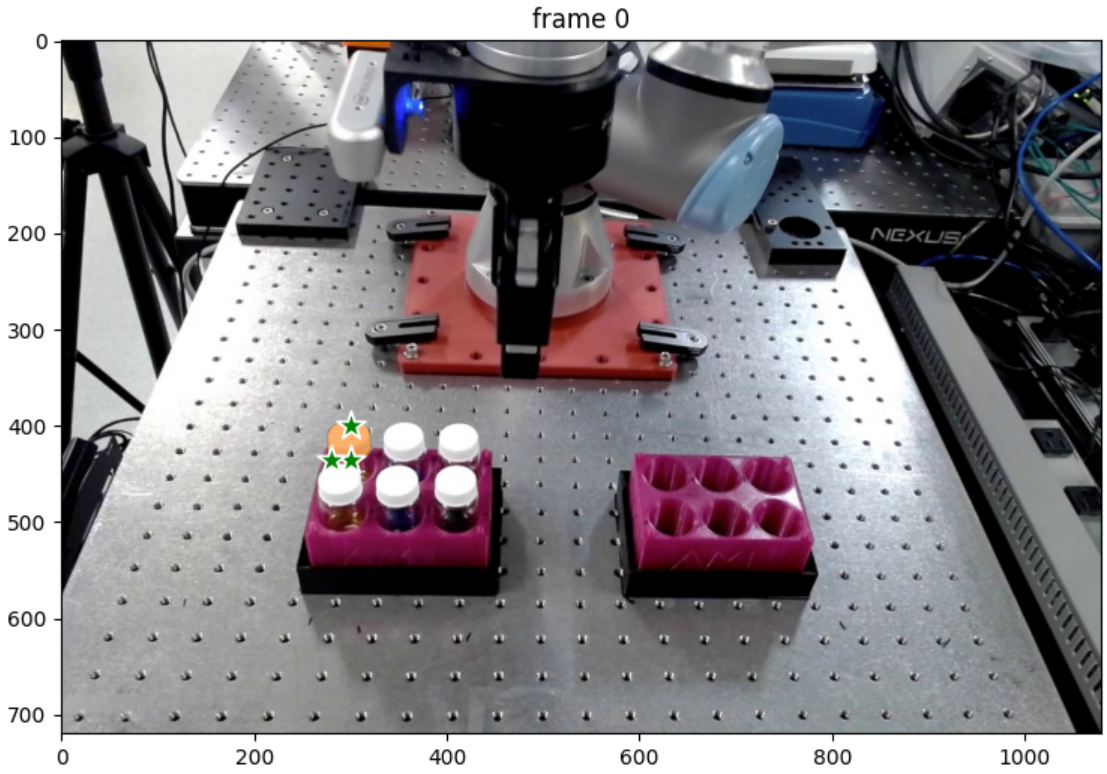
Step 1: Add a first click on a frame To get started, let's try to segment the child on the left.

Here we make a **positive click** at (x, y) = (210, 350) with label `1`, by sending their coordinates and labels into the `add_new_points_or_box` API.

Note: label `1` indicates a *positive click (to add a region)* while label `0` indicates a *negative click (to remove a region)*.

[12]:
```
ann_frame_idx = 0  # the frame index we interact with
ann_obj_id = 1  # give a unique id to each object we interact with (it can be␣
 ↪any integers)

# Let's add a positive click at (x, y) = (210, 350) to get started
#points = np.array([[210, 350]], dtype=np.float32)

points = np.array([[300, 400], [300,435], [280,435]], dtype=np.float32)

# for labels, `1` means positive click and `0` means negative click
labels = np.array([1,1,1], np.int32)
_, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
    inference_state=inference_state,
    frame_idx=ann_frame_idx,
    obj_id=ann_obj_id,
    points=points,
    labels=labels,
)

# show the results on the current (interacted) frame
plt.figure(figsize=(9, 6))
plt.title(f"frame {ann_frame_idx}")
plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
show_points(points, labels, plt.gca())
show_mask((out_mask_logits[0] > 0.0).cpu().numpy(), plt.gca(),␣
 ↪obj_id=out_obj_ids[0])
```

```
/home/moore278/DSC/sam2/sam2/sam2_video_predictor.py:786: UserWarning: cannot
import name '_C' from 'sam2' (/home/moore278/DSC/sam2/sam2/__init__.py)

Skipping the post-processing step due to the error above. You can still use SAM
2 and it's OK to ignore the error above, although some post-processing
functionality may be limited (which doesn't affect the results in most cases;
see https://github.com/facebookresearch/sam2/blob/main/INSTALL.md).
  pred_masks_gpu = fill_holes_in_mask_scores(
```



Step 2: Add a second click to refine the prediction Hmm, it seems that although we wanted to segment the child on the left, the model predicts the mask for only the shorts – this can happen since there is ambiguity from a single click about what the target object should be. We can refine the mask on this frame via another positive click on the child's shirt.

Here we make a **second positive click** at (x, y) = (250, 220) with label `1` to expand the mask.

Note: we need to send **all the clicks and their labels** (i.e. not just the last click) when calling `add_new_points_or_box`.

[13]:
```
if False:
    ann_frame_idx = 0  # the frame index we interact with
    ann_obj_id = 1  # give a unique id to each object we interact with (it can␣
 ↪be any integers)

    # Let's add a 2nd positive click at (x, y) = (250, 220) to refine the mask
    # sending all clicks (and their labels) to `add_new_points_or_box`
    points = np.array([[210, 350], [250, 220]], dtype=np.float32)
    # for labels, `1` means positive click and `0` means negative click
    labels = np.array([1, 1], np.int32)
    _, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
        inference_state=inference_state,
        frame_idx=ann_frame_idx,
```

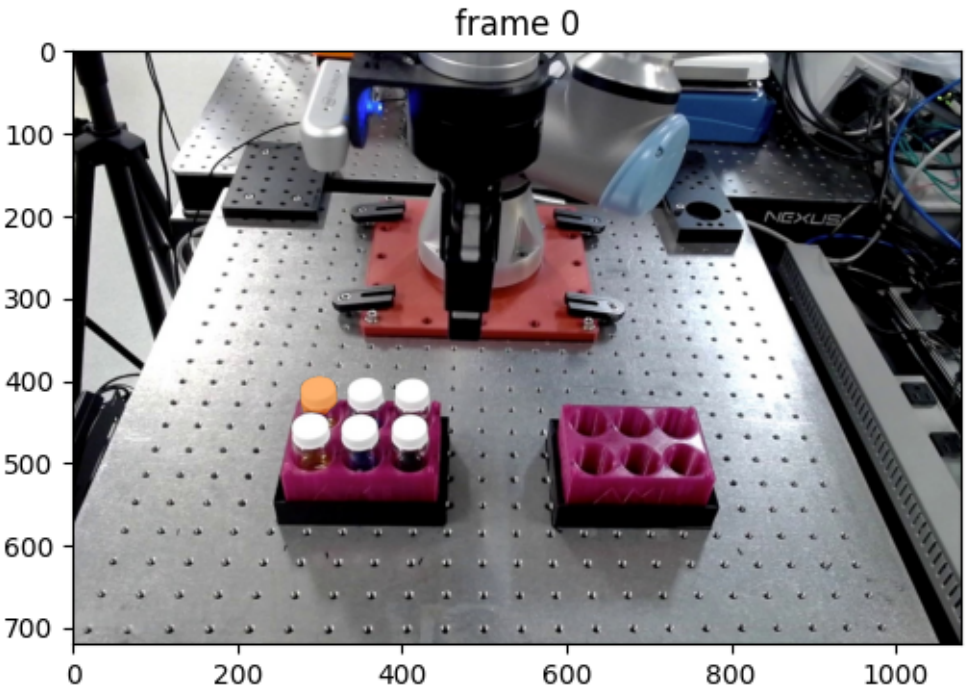
```
        obj_id=ann_obj_id,
        points=points,
        labels=labels,
    )

    # show the results on the current (interacted) frame
    plt.figure(figsize=(9, 6))
    plt.title(f"frame {ann_frame_idx}")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
    show_points(points, labels, plt.gca())
    show_mask((out_mask_logits[0] > 0.0).cpu().numpy(), plt.gca(),␣
↪obj_id=out_obj_ids[0])
```

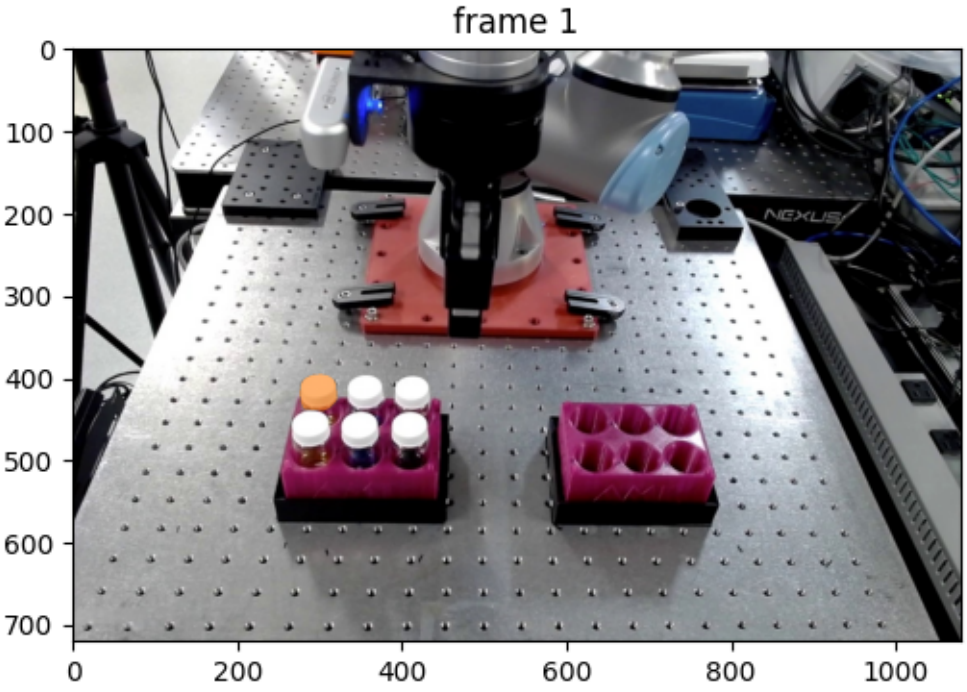
With this 2nd refinement click, now we get a segmentation mask of the entire child on frame 0.

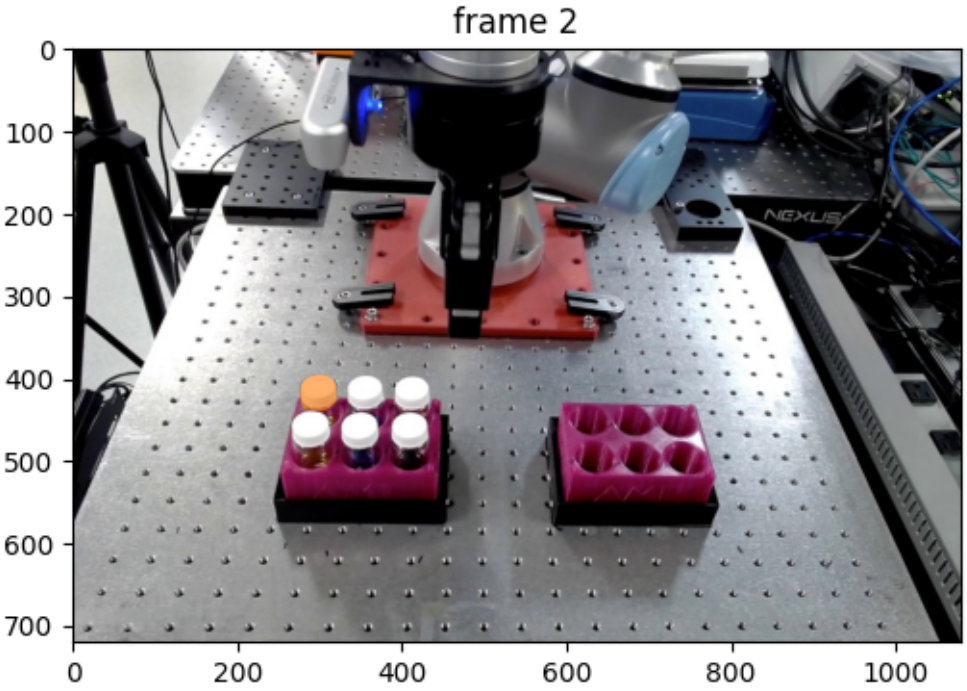
Step 3: Propagate the prompts to get the masklet across the video To get the masklet throughout the entire video, we propagate the prompts using the `propagate_in_video` API.

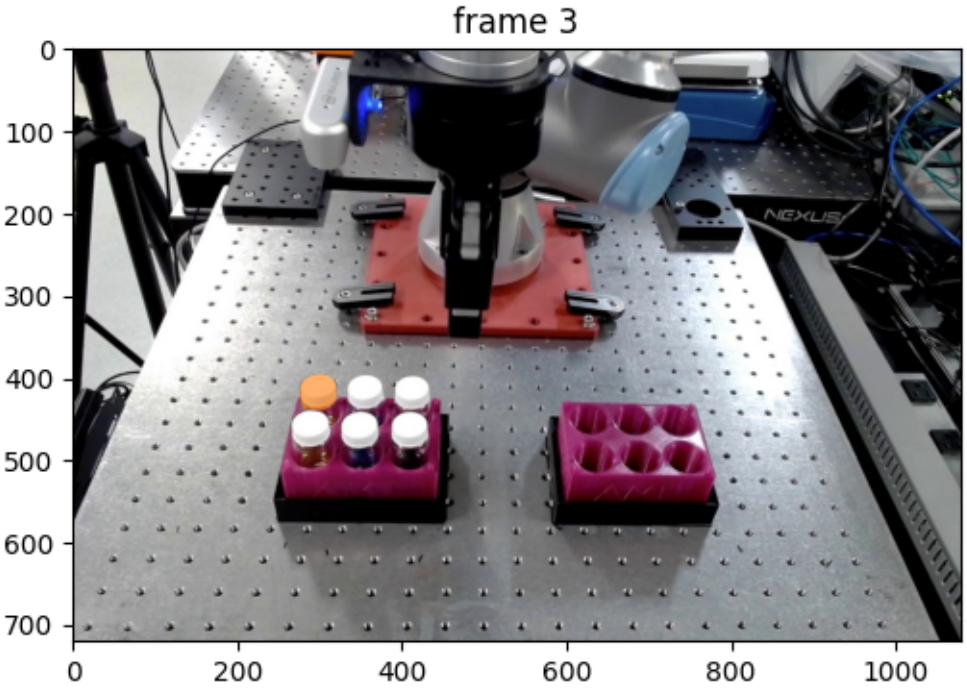
[14]:
```
# run propagation throughout the video and collect the results in a dict
video_segments = {}  # video_segments contains the per-frame segmentation␣
↪results
for out_frame_idx, out_obj_ids, out_mask_logits in predictor.
↪propagate_in_video(inference_state):
    video_segments[out_frame_idx] = {
        out_obj_id: (out_mask_logits[i] > 0.0).cpu().numpy()
        for i, out_obj_id in enumerate(out_obj_ids)
    }

# render the segmentation results every few frames
vis_frame_stride = 1
plt.close("all")
for out_frame_idx in range(0, len(frame_names), vis_frame_stride):
    plt.figure(figsize=(6, 4))
    plt.title(f"frame {out_frame_idx}")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[out_frame_idx])))
    for out_obj_id, out_mask in video_segments[out_frame_idx].items():
        show_mask(out_mask, plt.gca(), obj_id=out_obj_id)
```

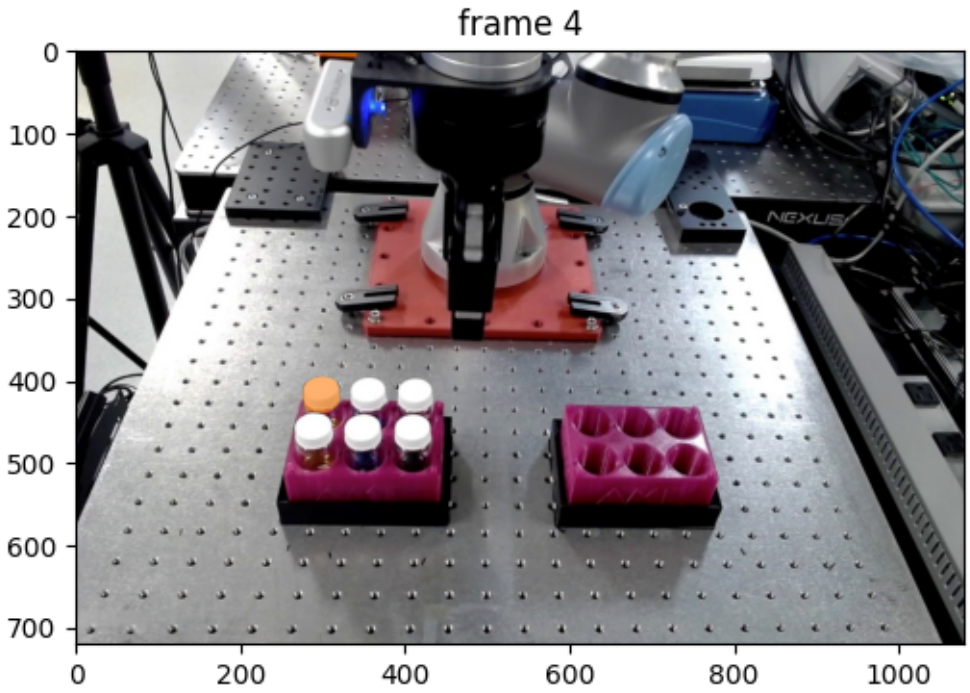
```
propagate in video:   0%|          | 0/80 [00:00<?, ?it/s]

propagate in video: 100%|     | 80/80 [00:03<00:00, 24.95it/s]
/tmp/ipykernel_2181828/4151343840.py:13: RuntimeWarning: More than 20 figures
have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  plt.figure(figsize=(6, 4))
```

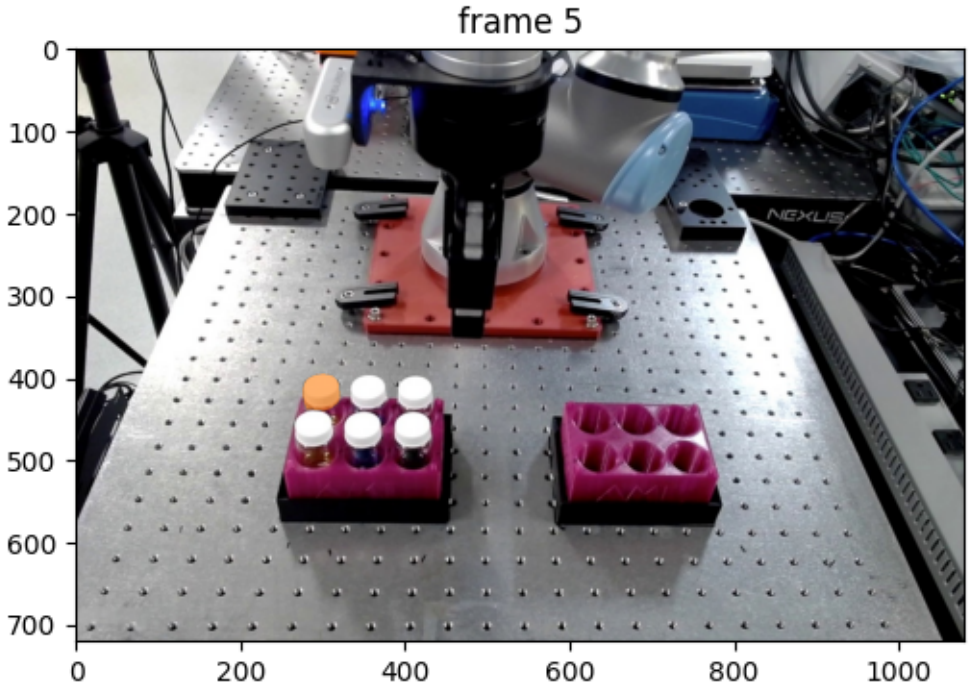

frame 0
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

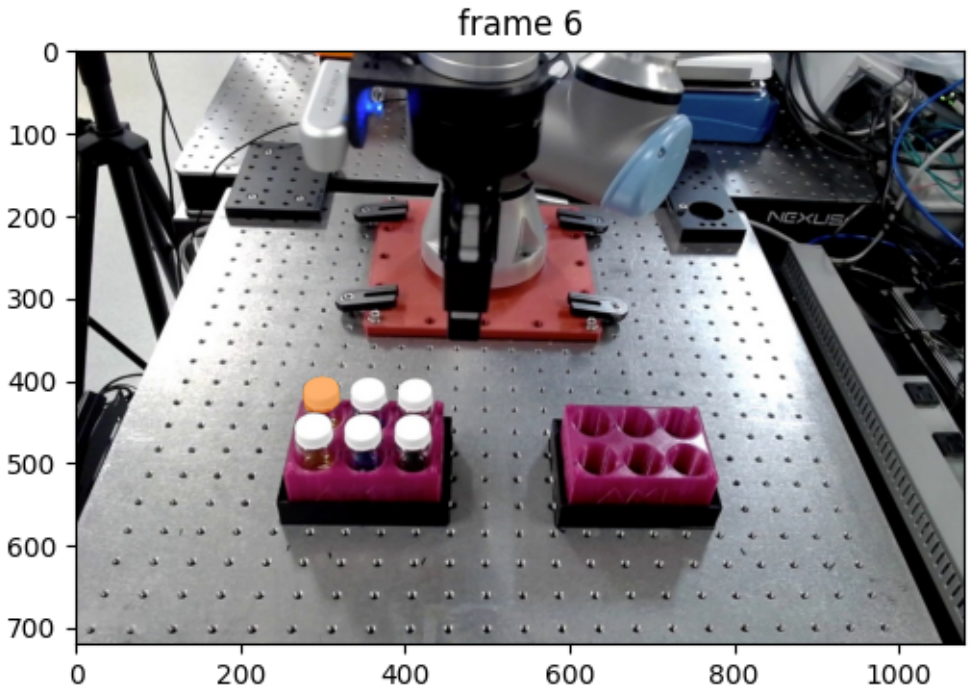

frame 1
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

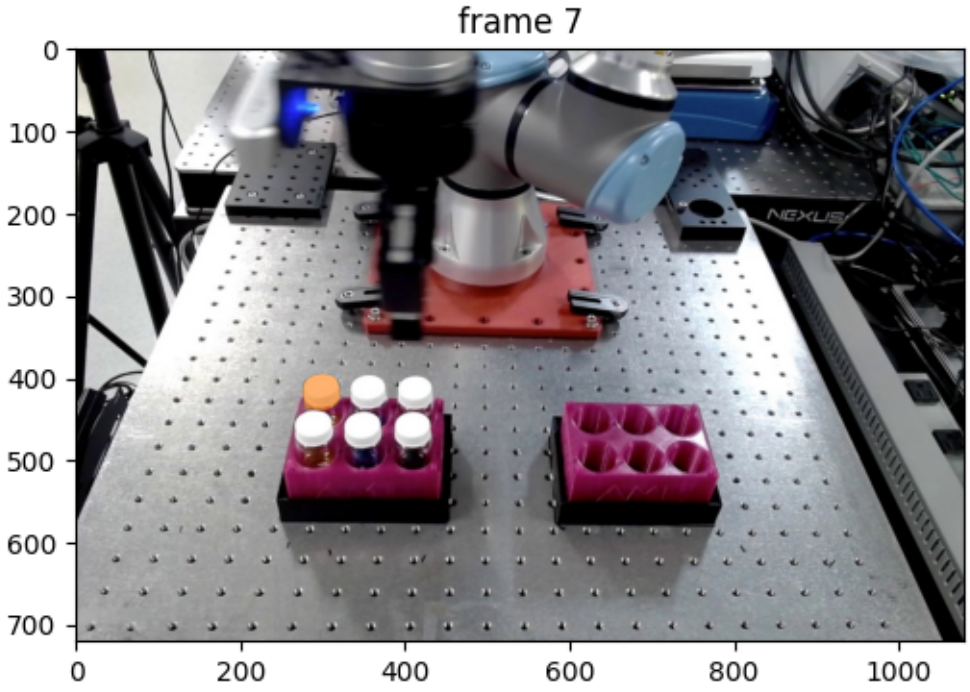

frame 2
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

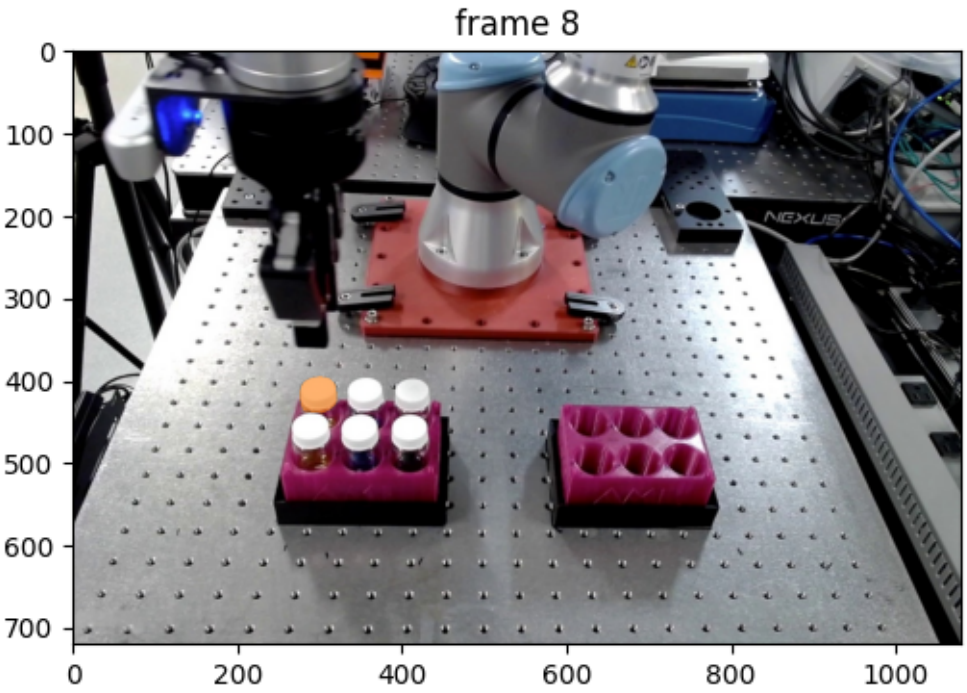

frame 3
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

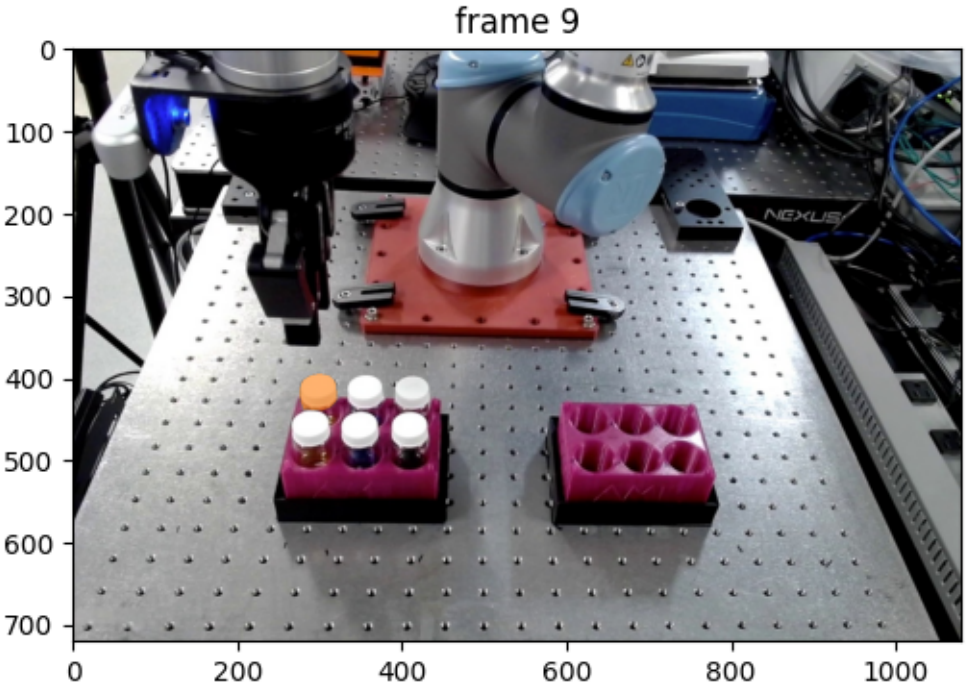

frame 4
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

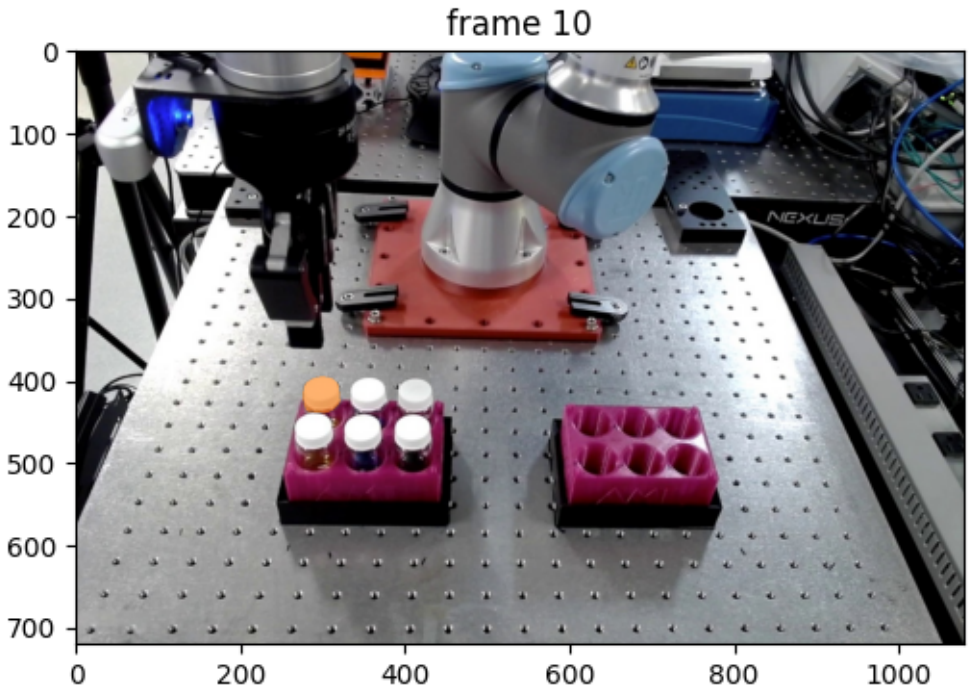

frame 5
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

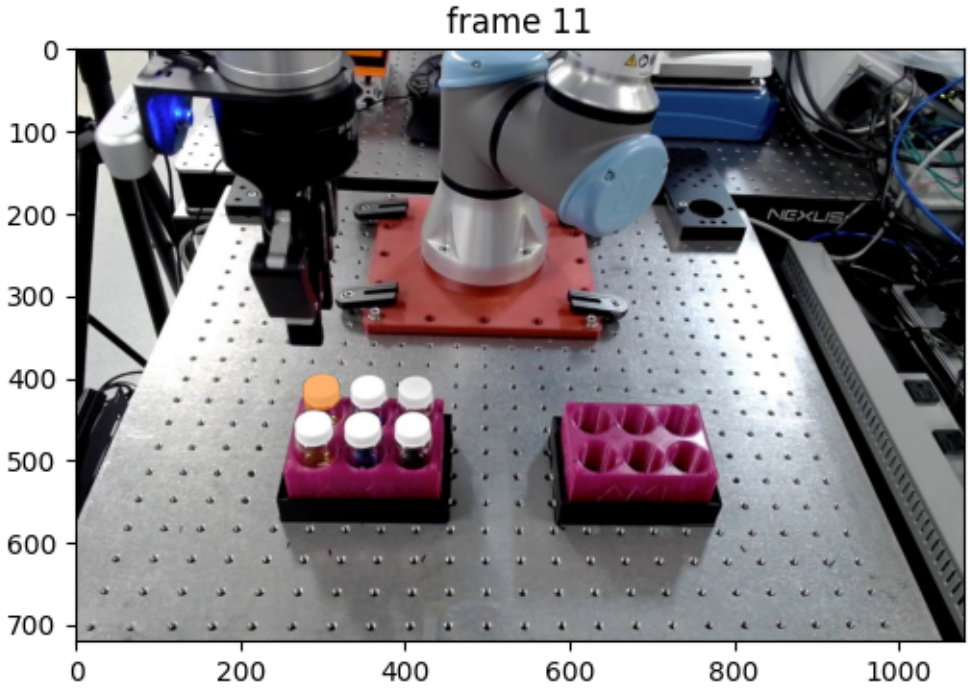

frame 6
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

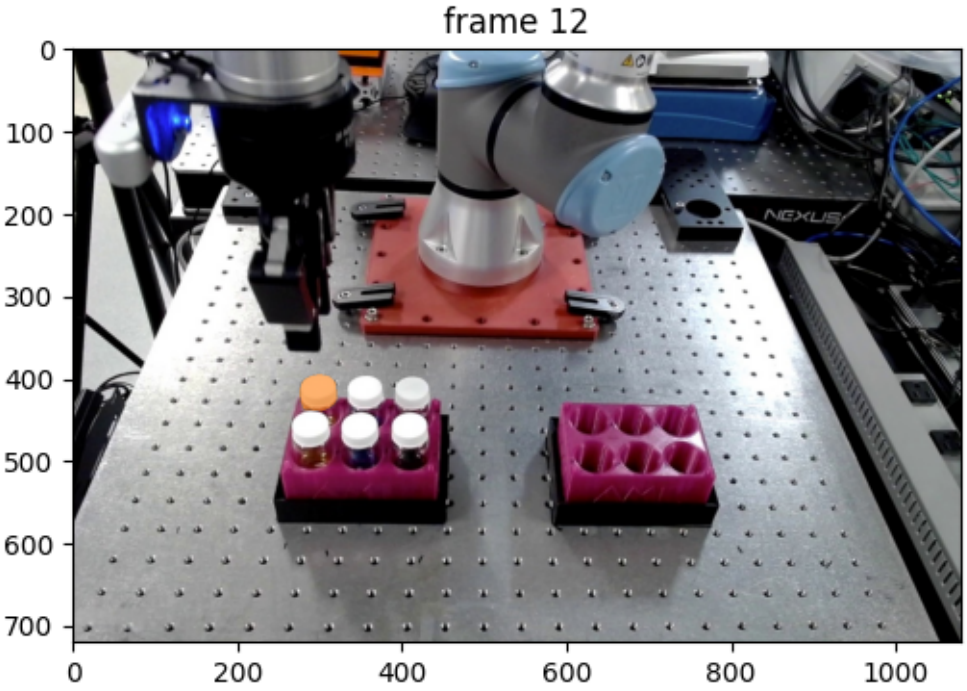

frame 7
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

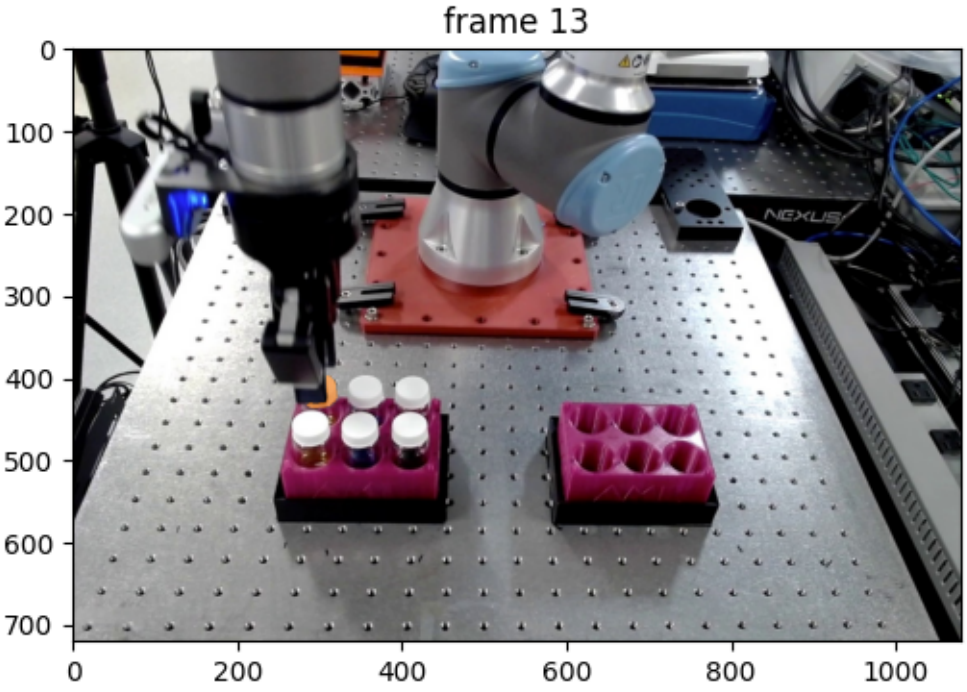

frame 8
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

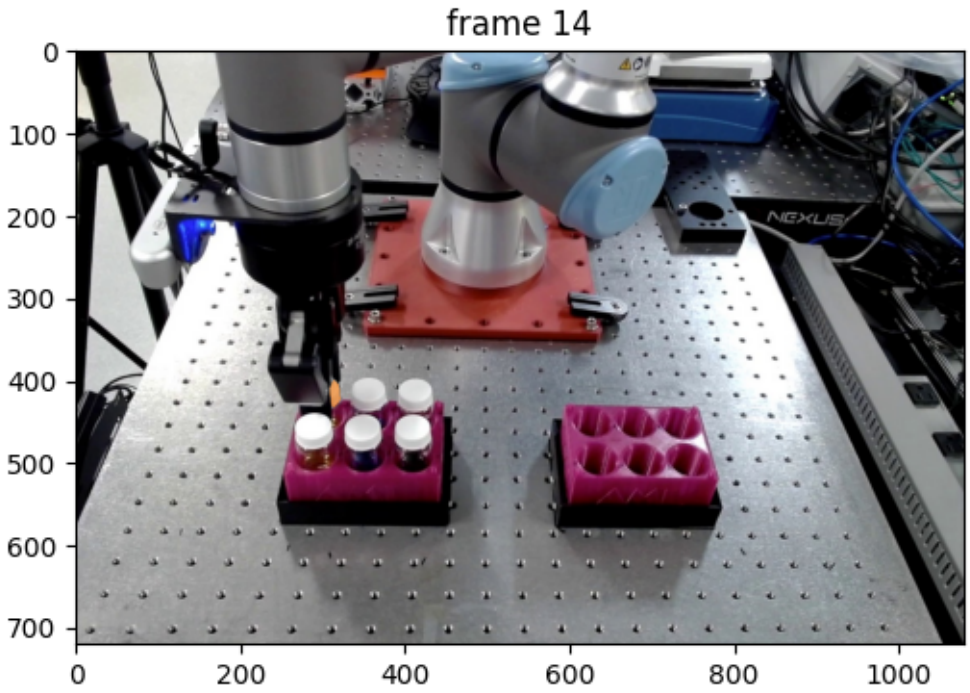

frame 9
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

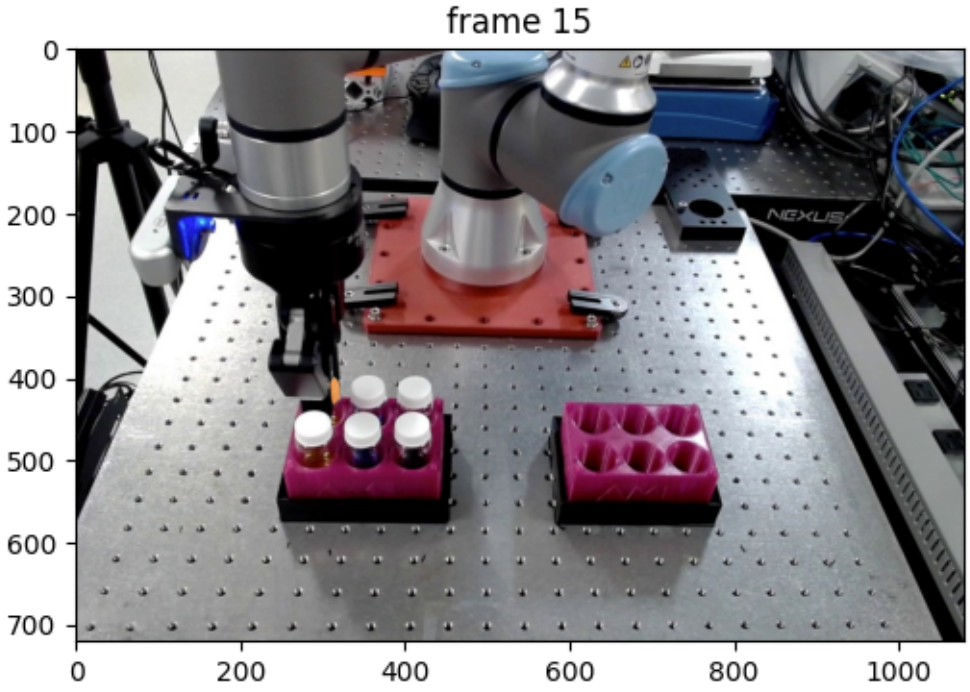

frame 10
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

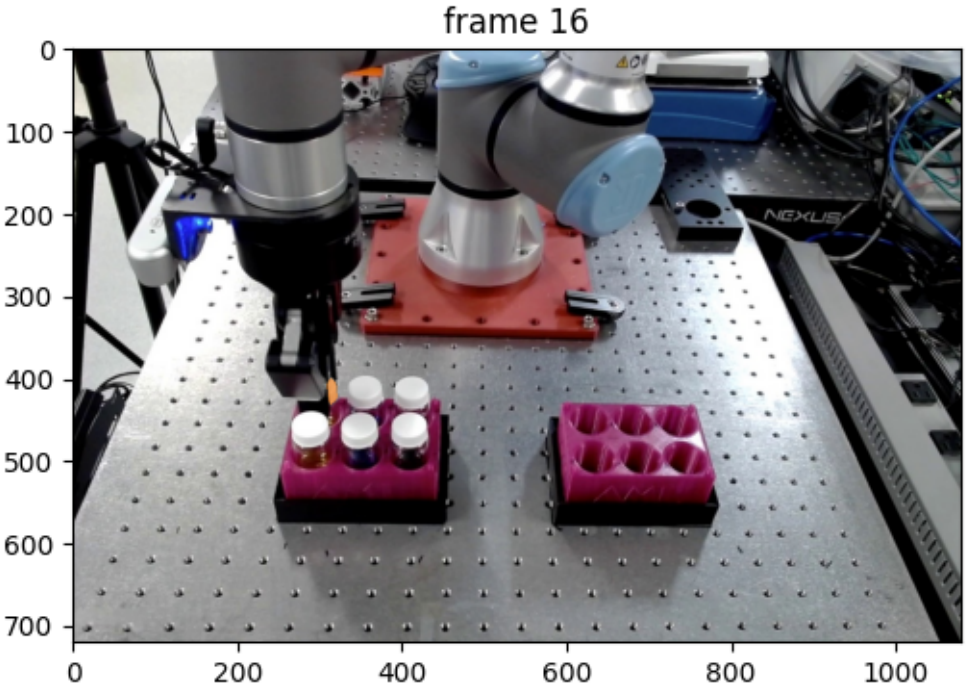

frame 11
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

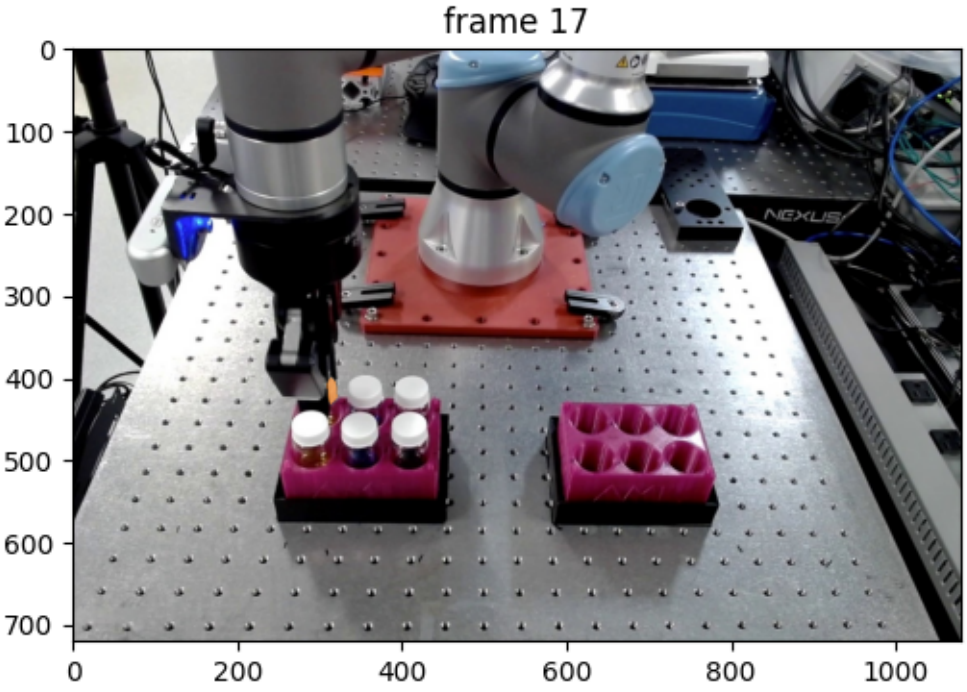

frame 12
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

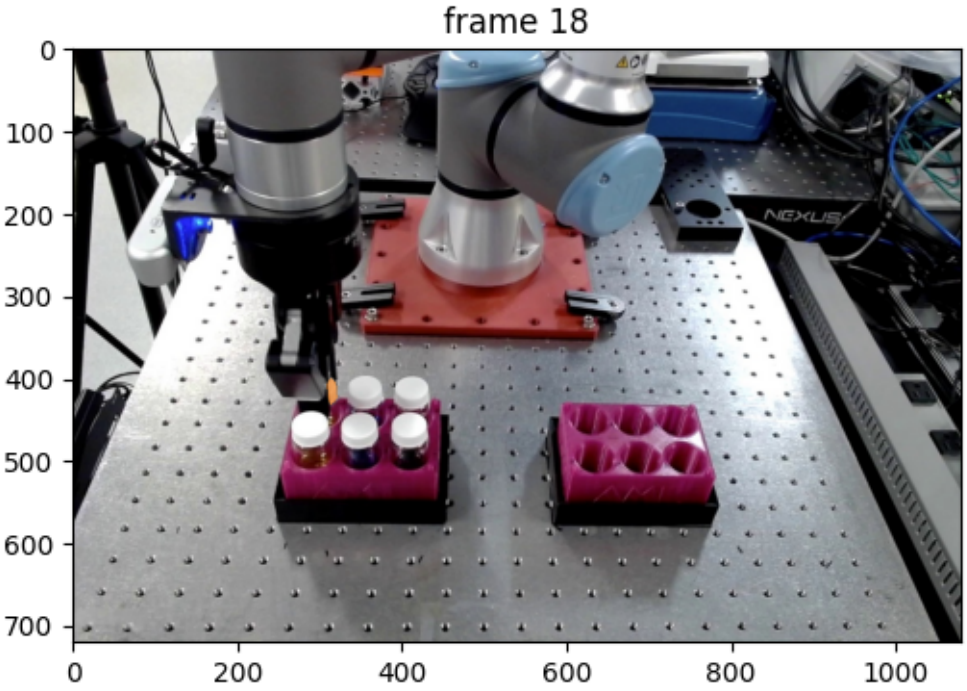

frame 13
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

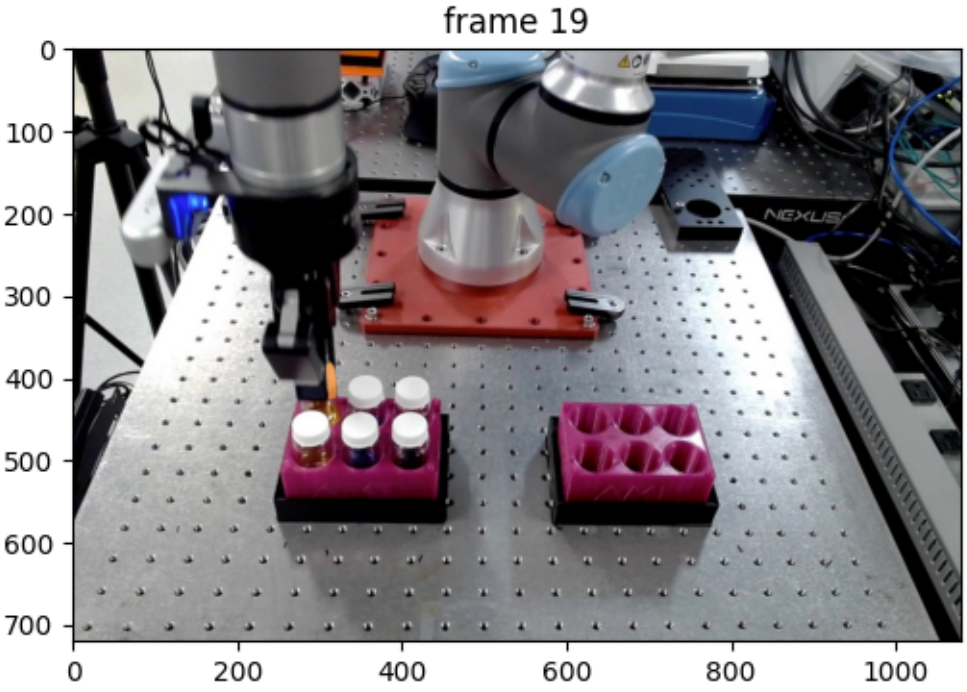

frame 14
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

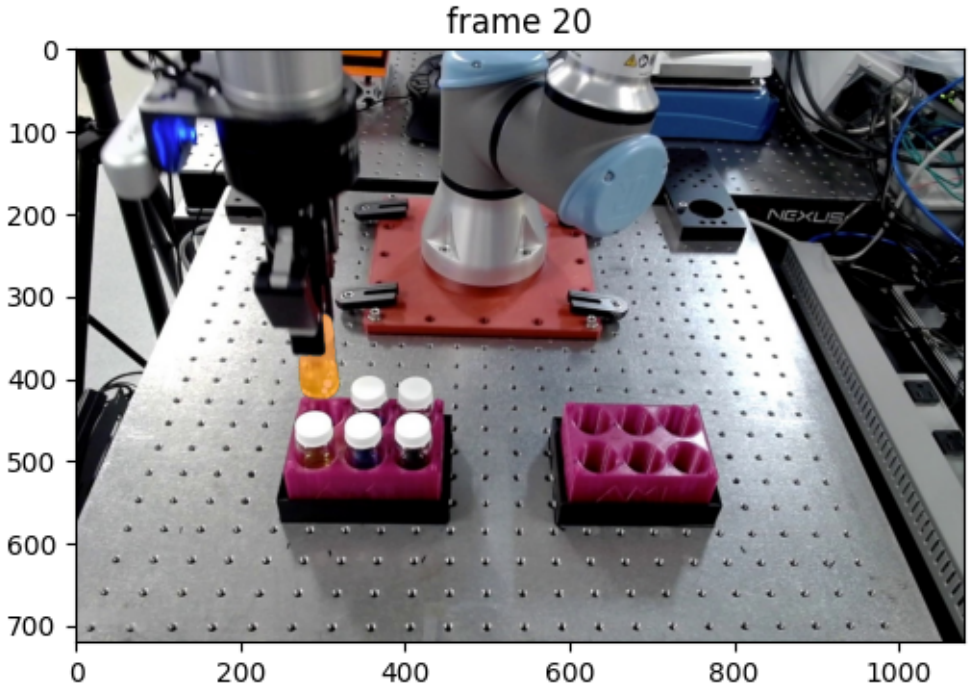

frame 15
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

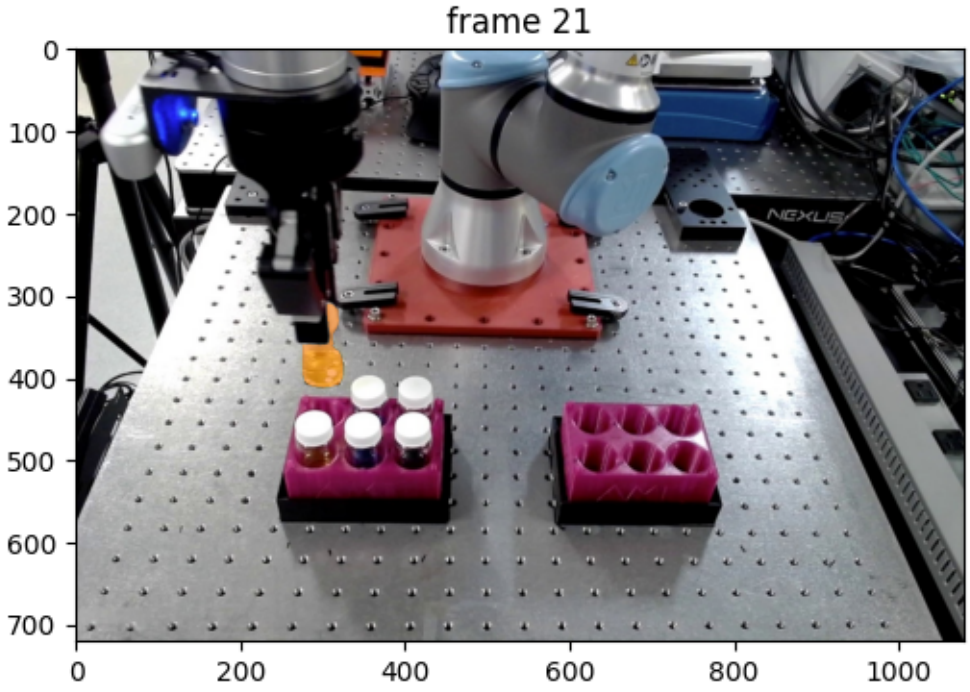

frame 16
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

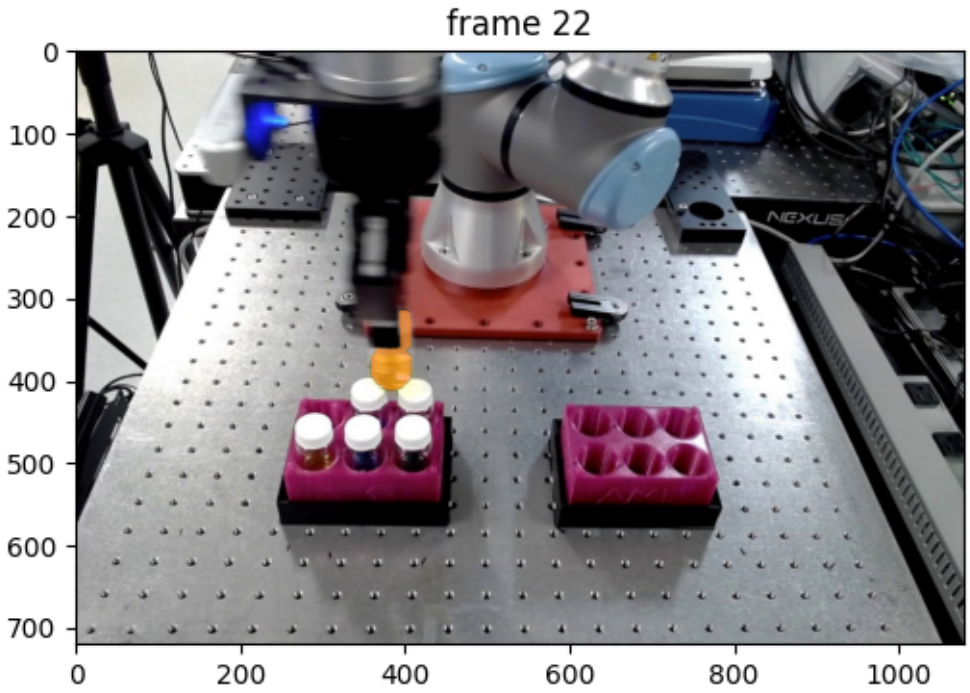

frame 17
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

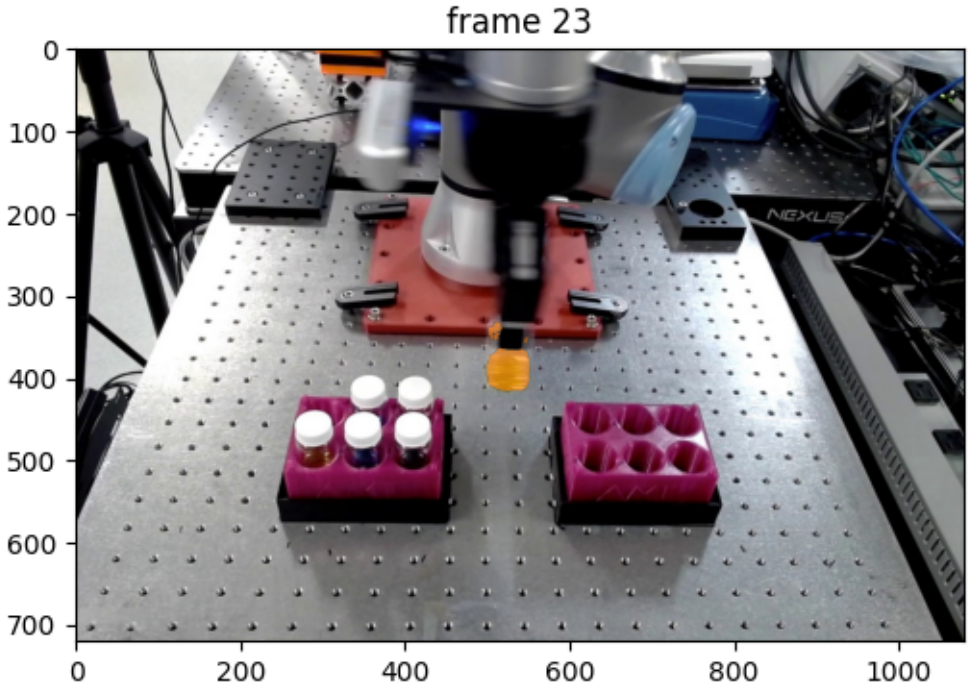

frame 18
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

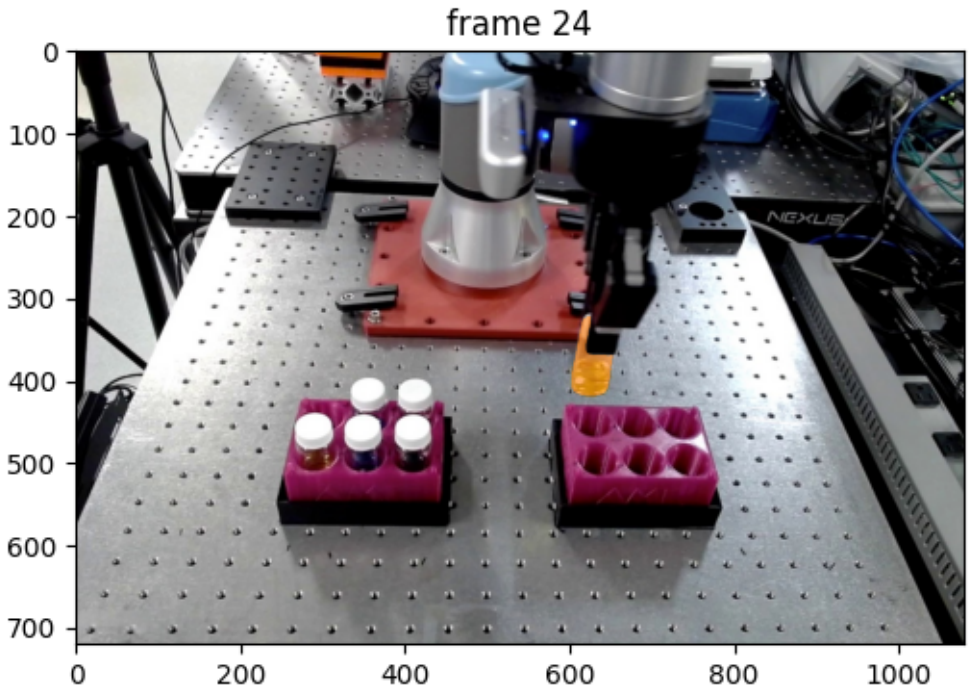

frame 19
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

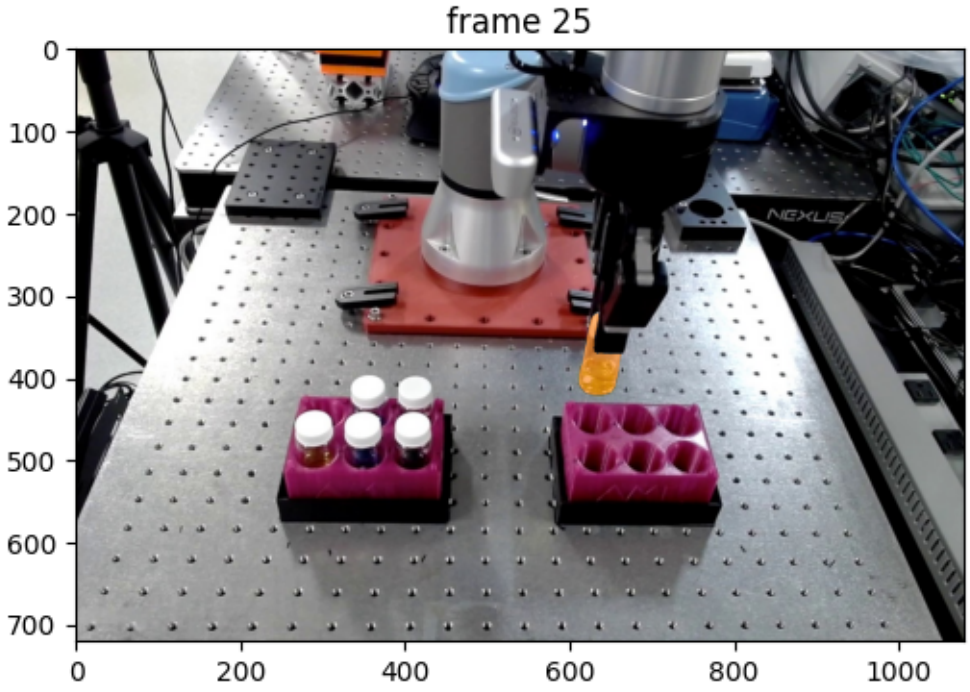

frame 20
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

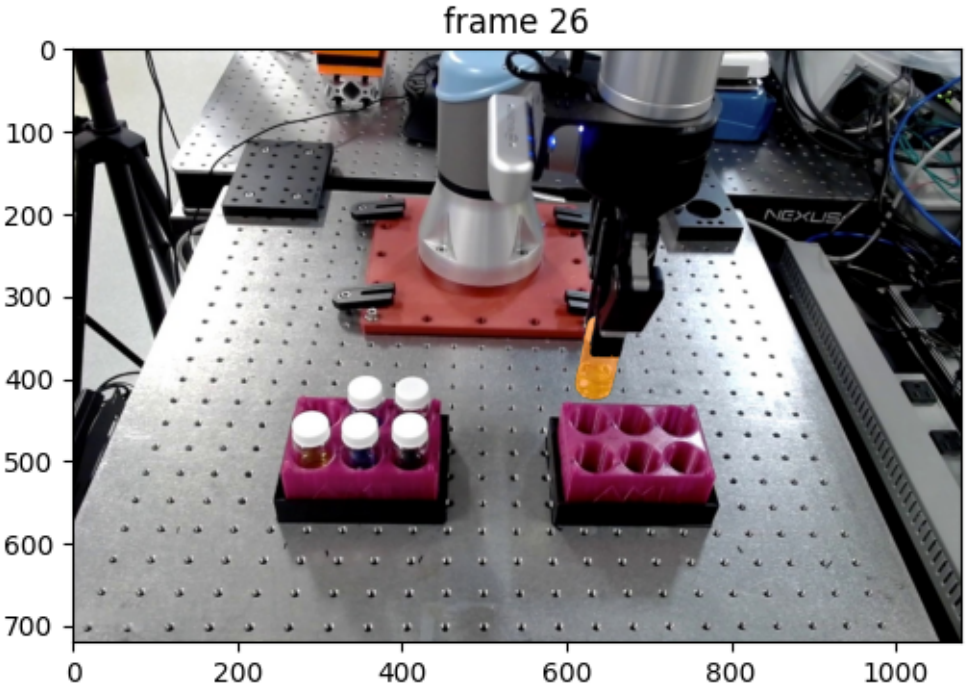

frame 21
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

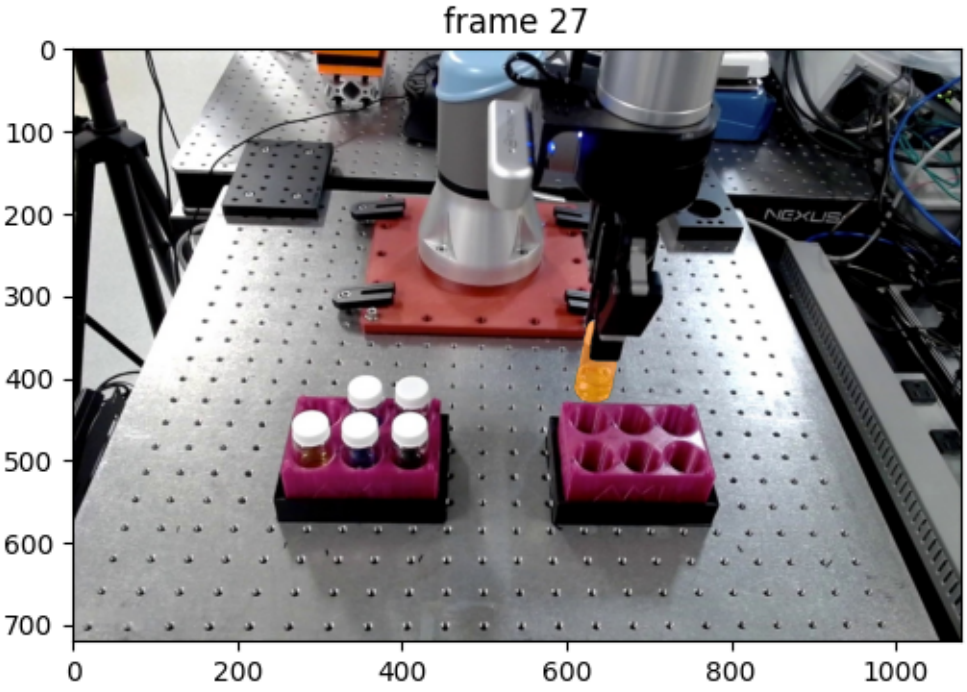

frame 22
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

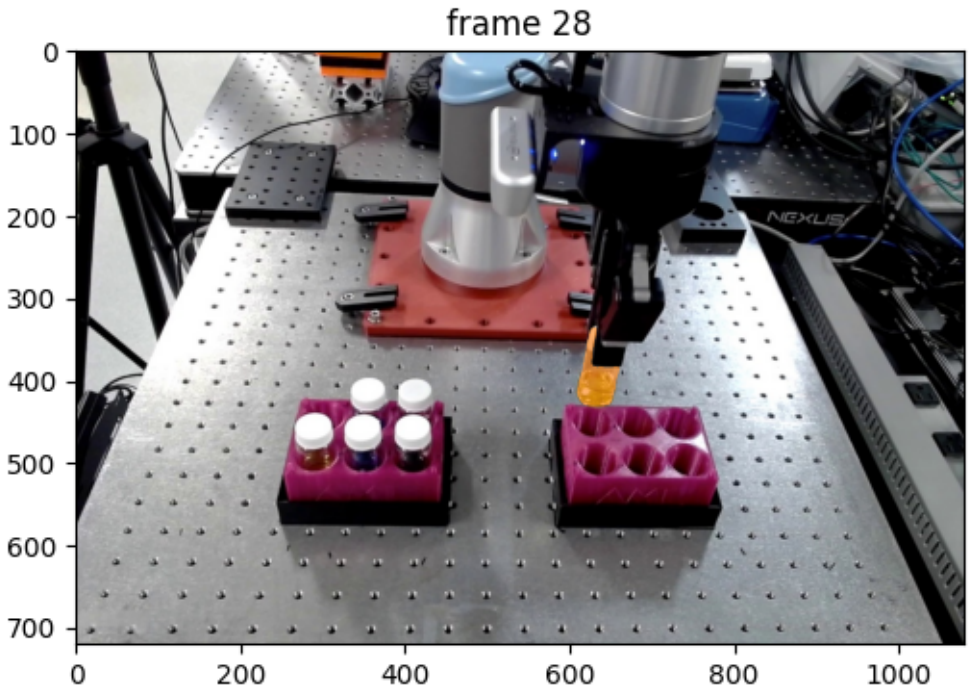

frame 23
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

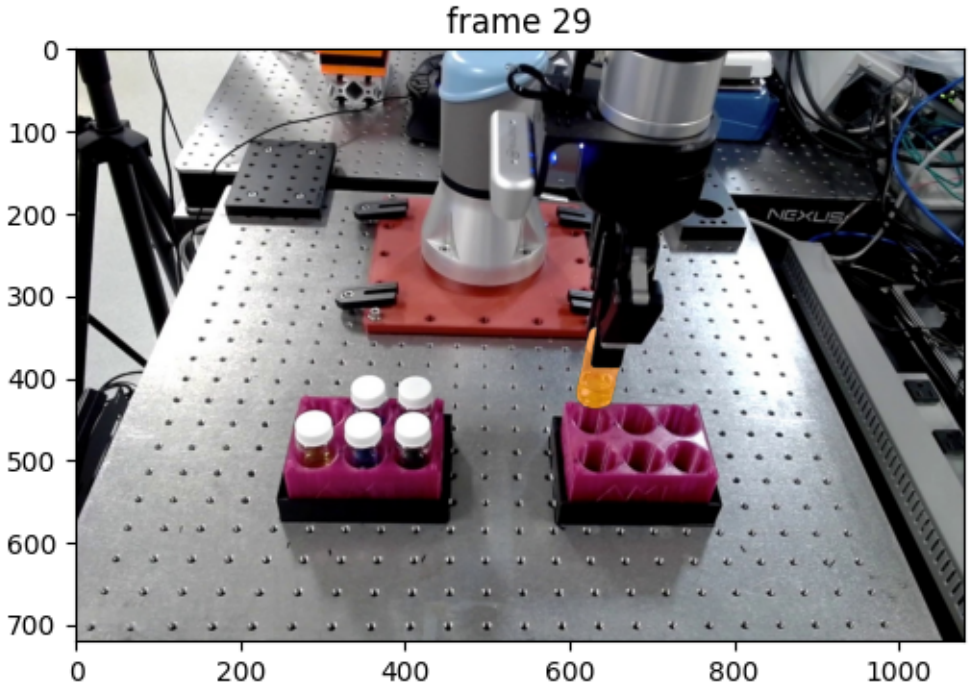

frame 24
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

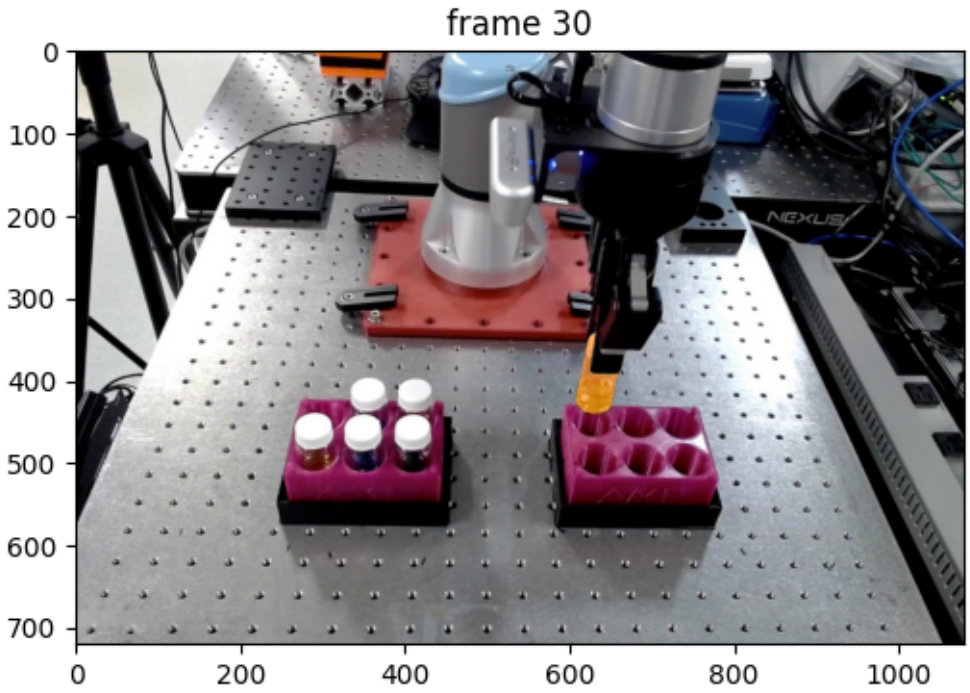

frame 25
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

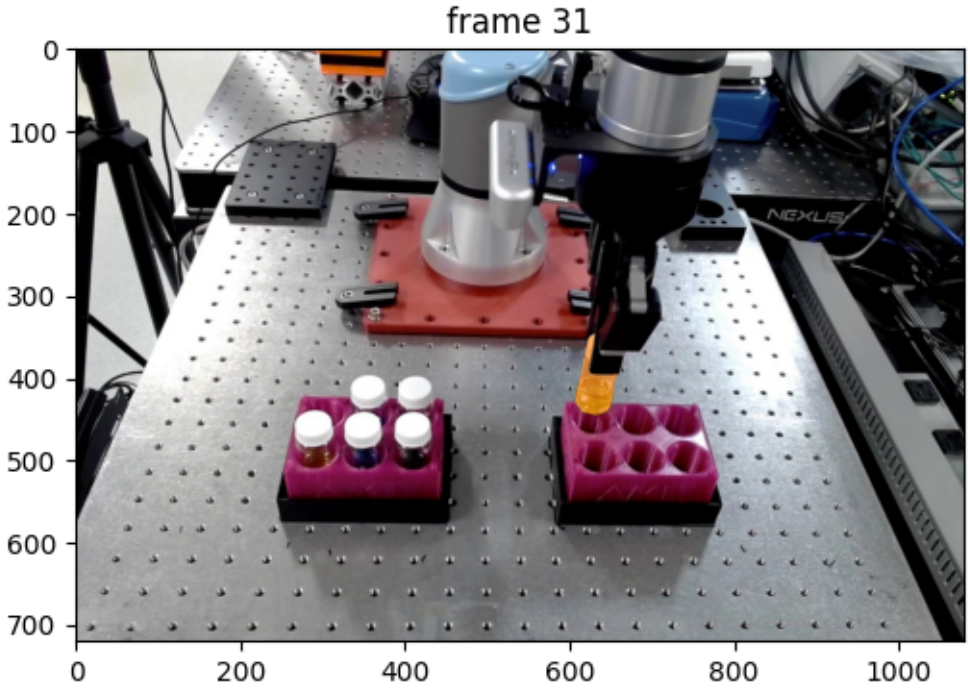

frame 26
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

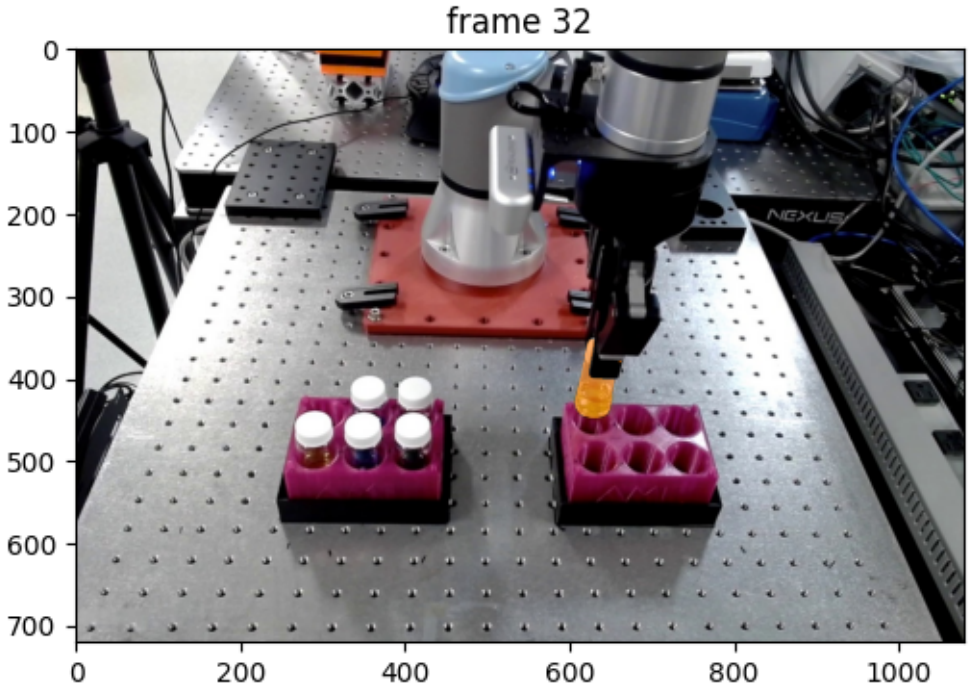

frame 27
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

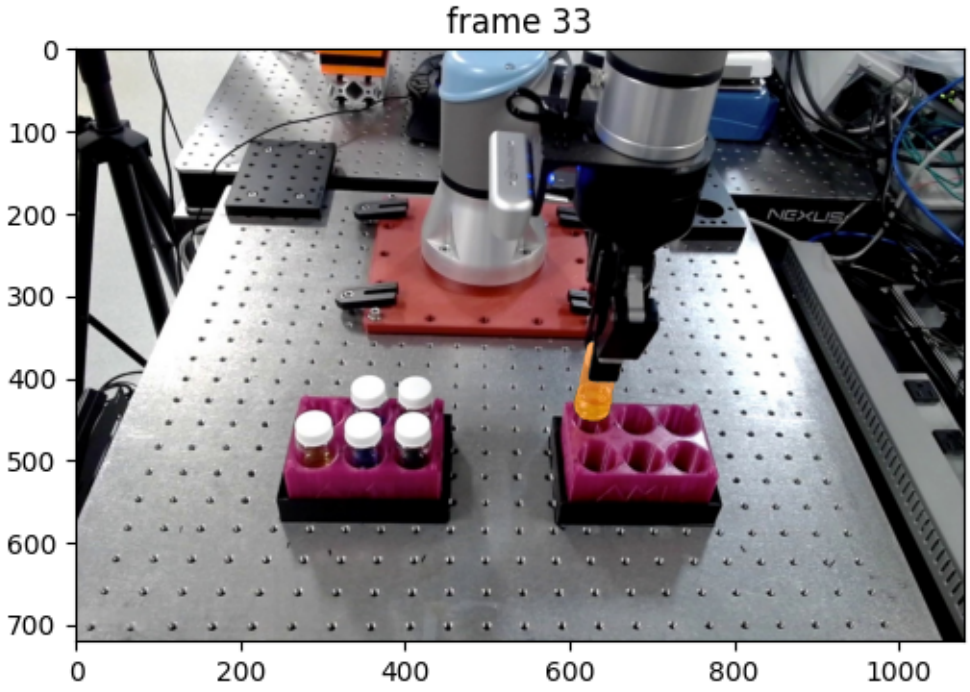

frame 28
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

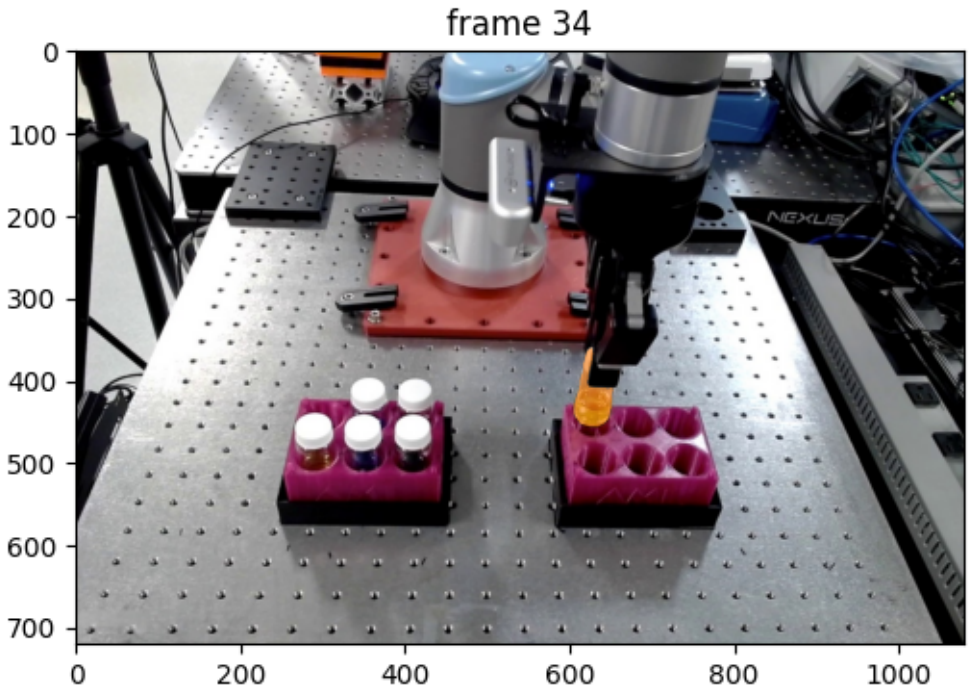

frame 29
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

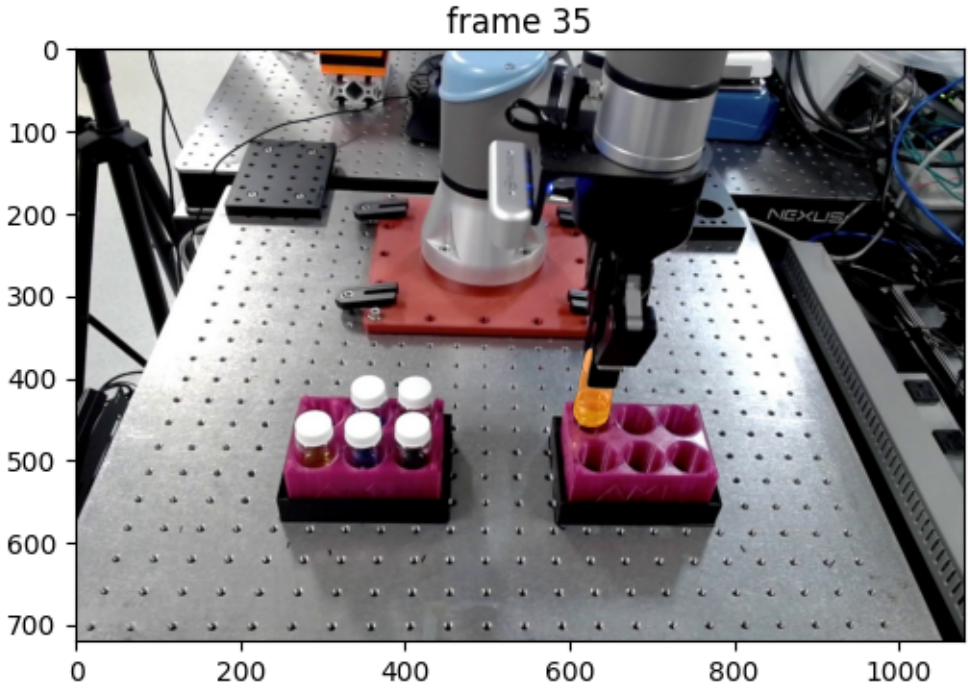

frame 30
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

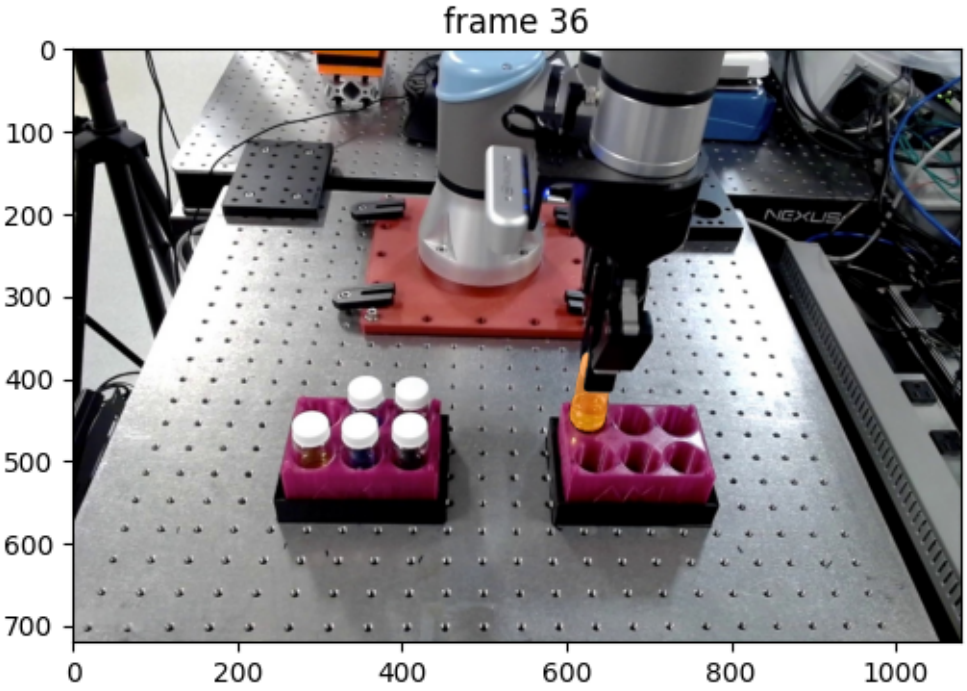

frame 31
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

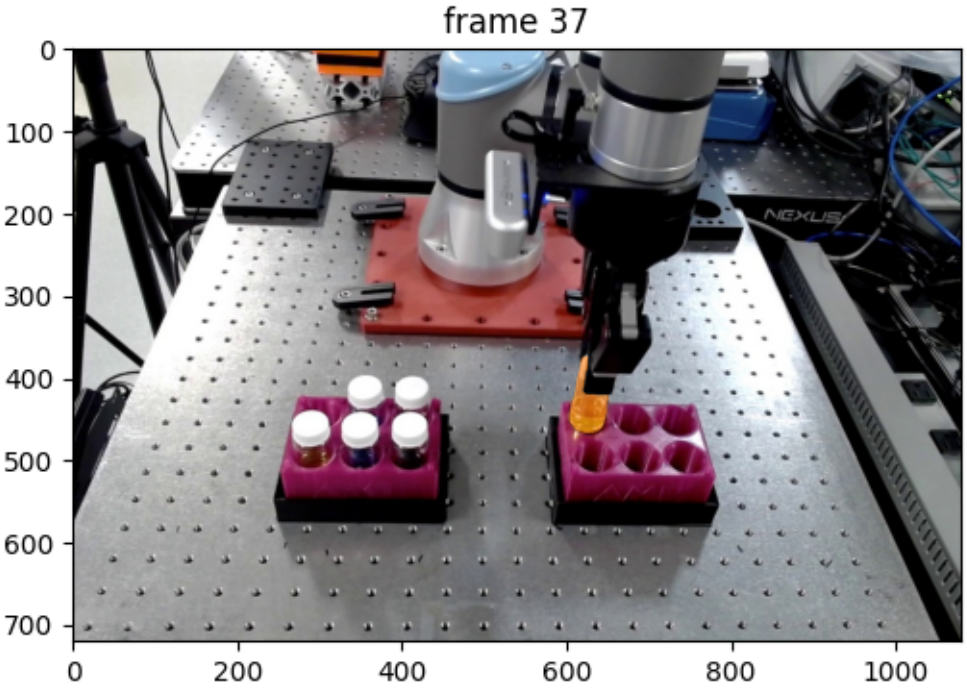

frame 32
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

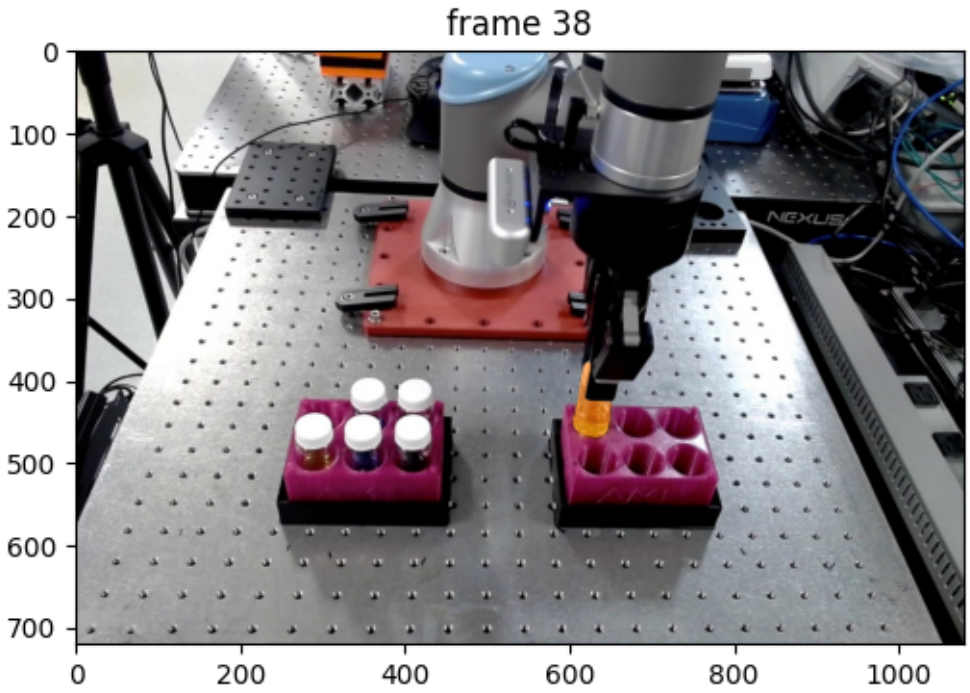

frame 33
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

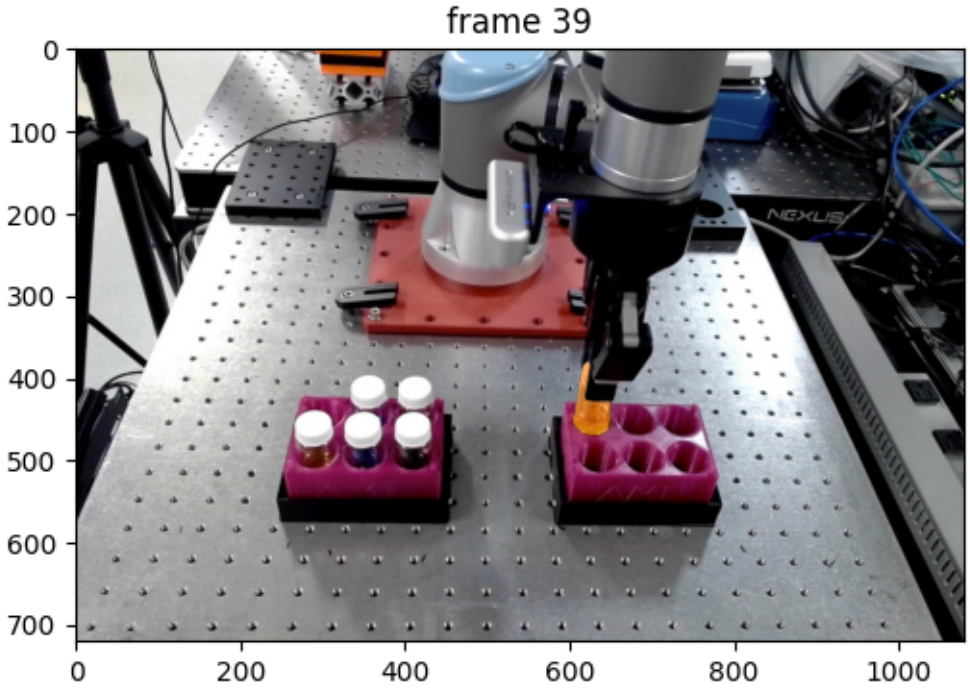

frame 34
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

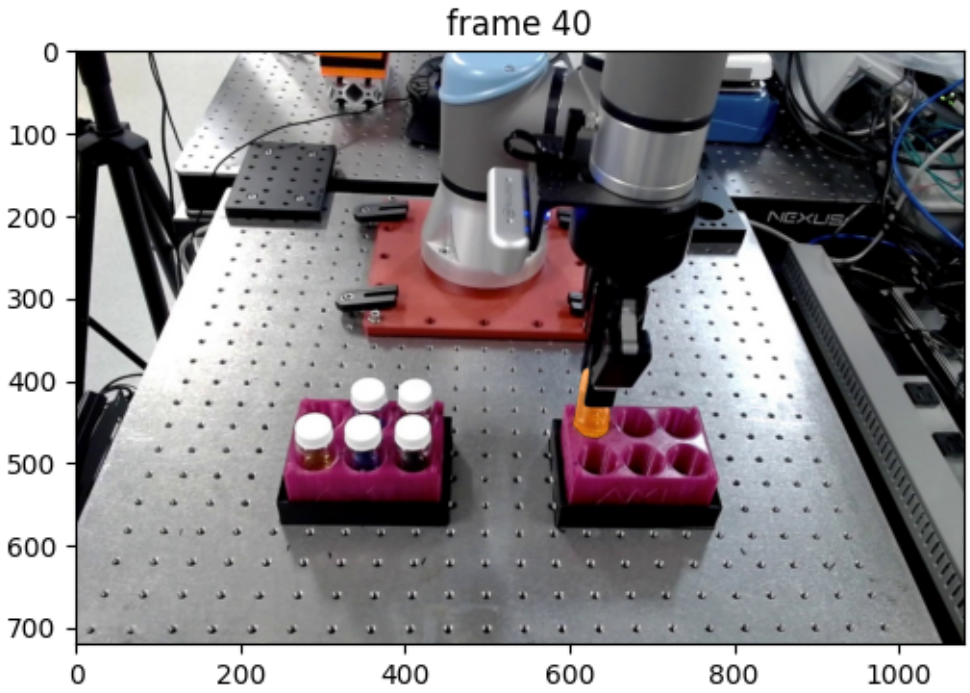

frame 35
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

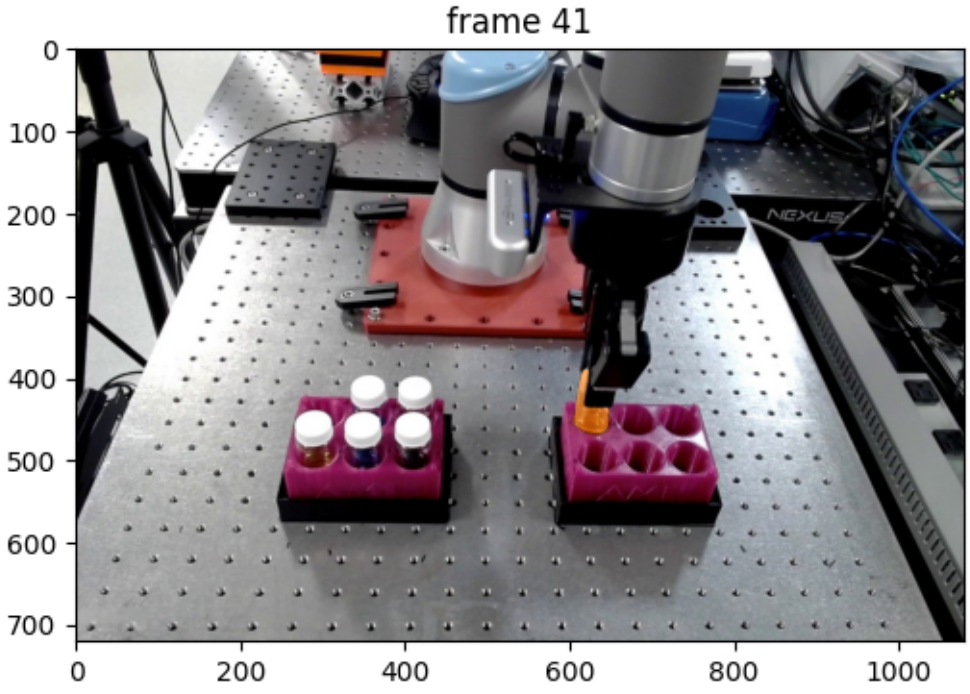

frame 36
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

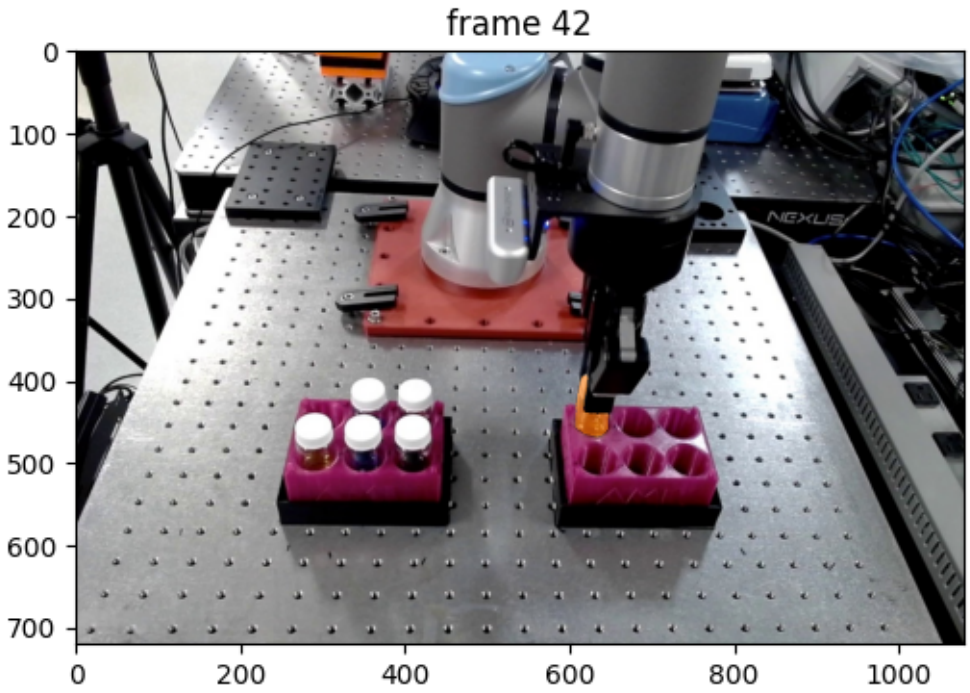

frame 37
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

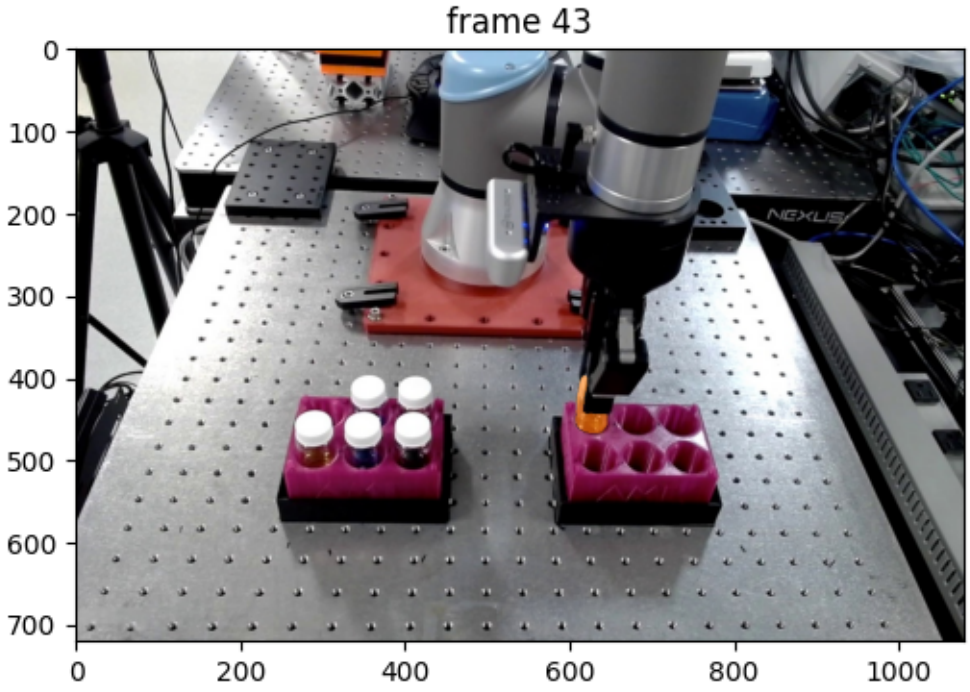

frame 38
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

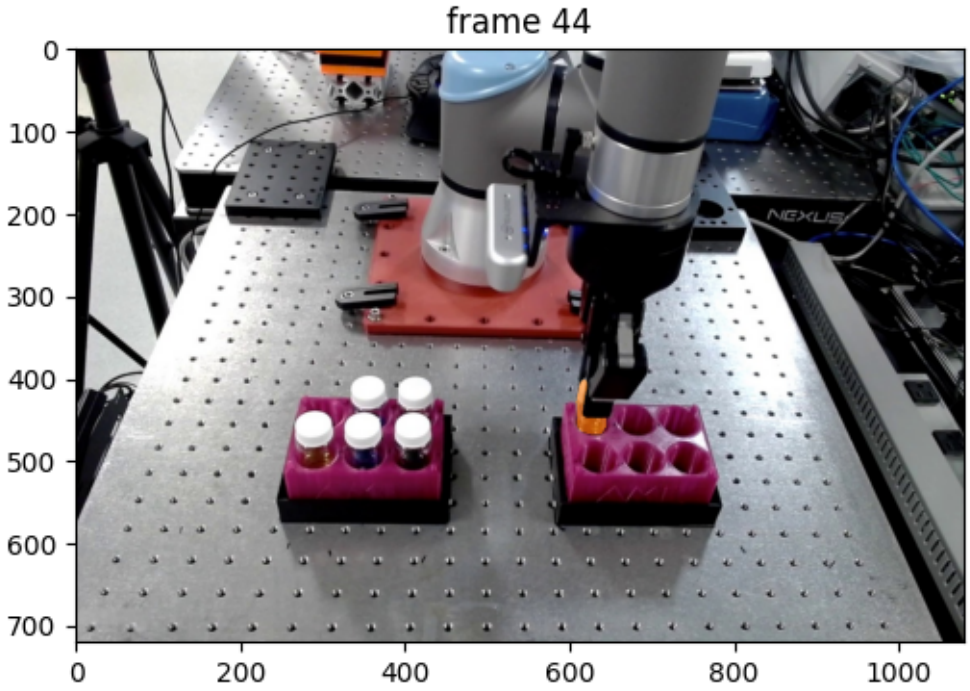

frame 39
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

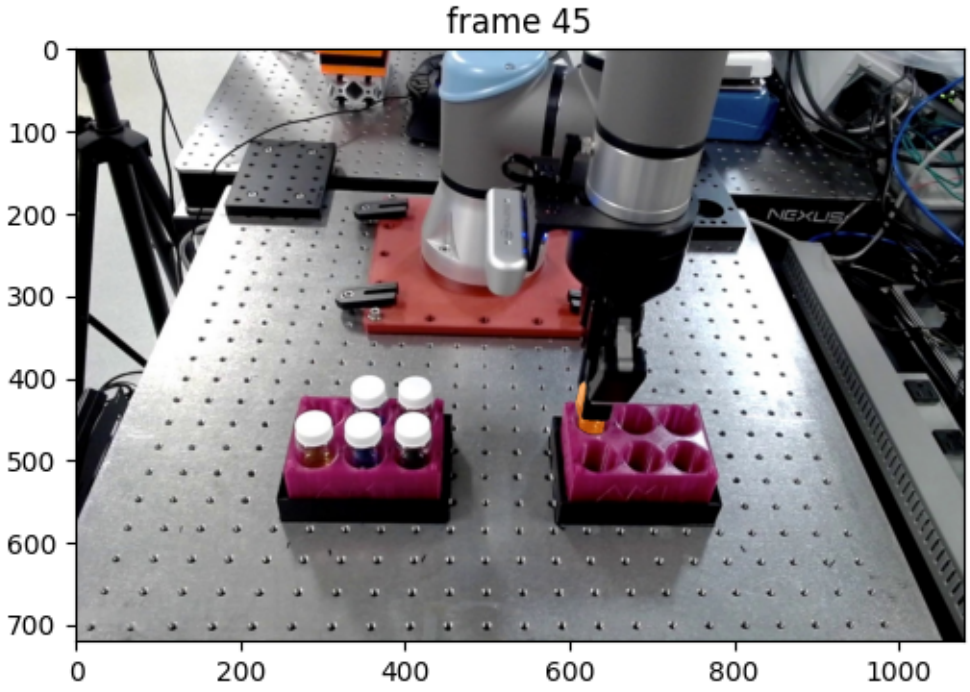

frame 40
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

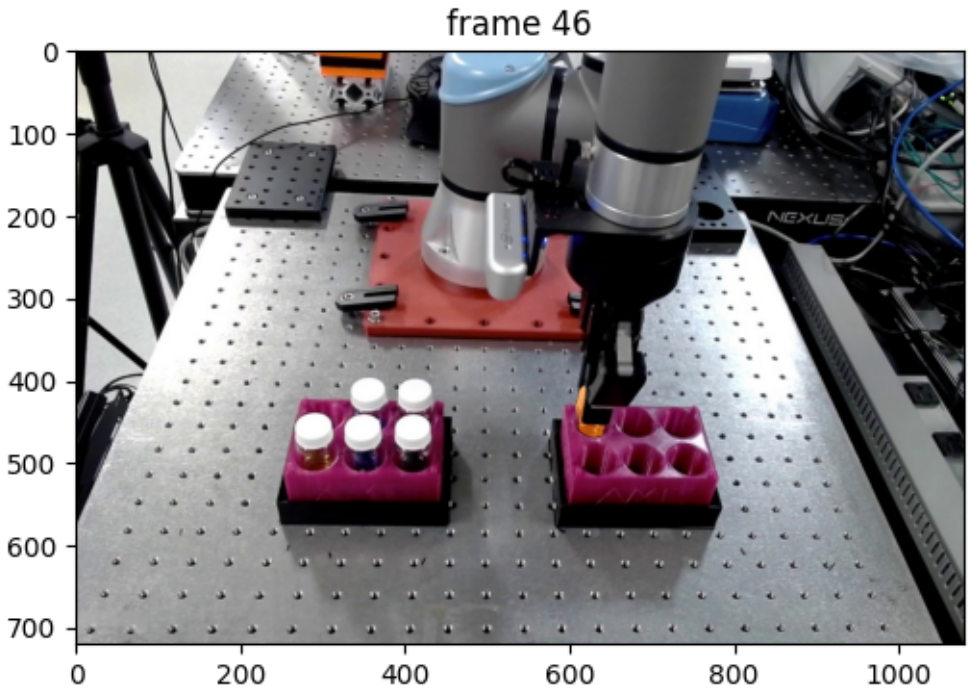

frame 41
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

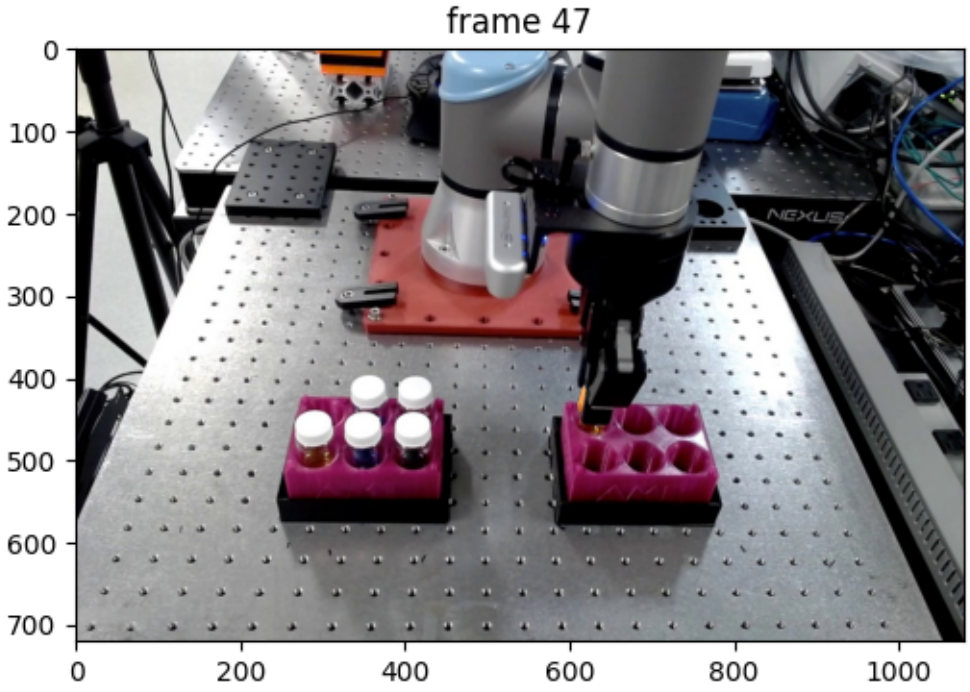

frame 42
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

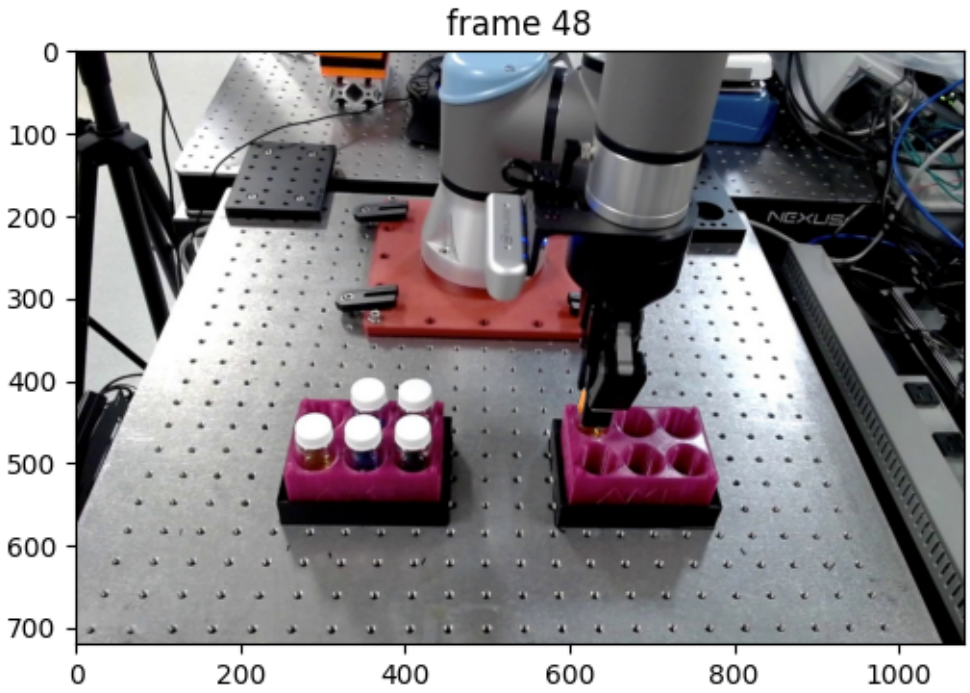

frame 43
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

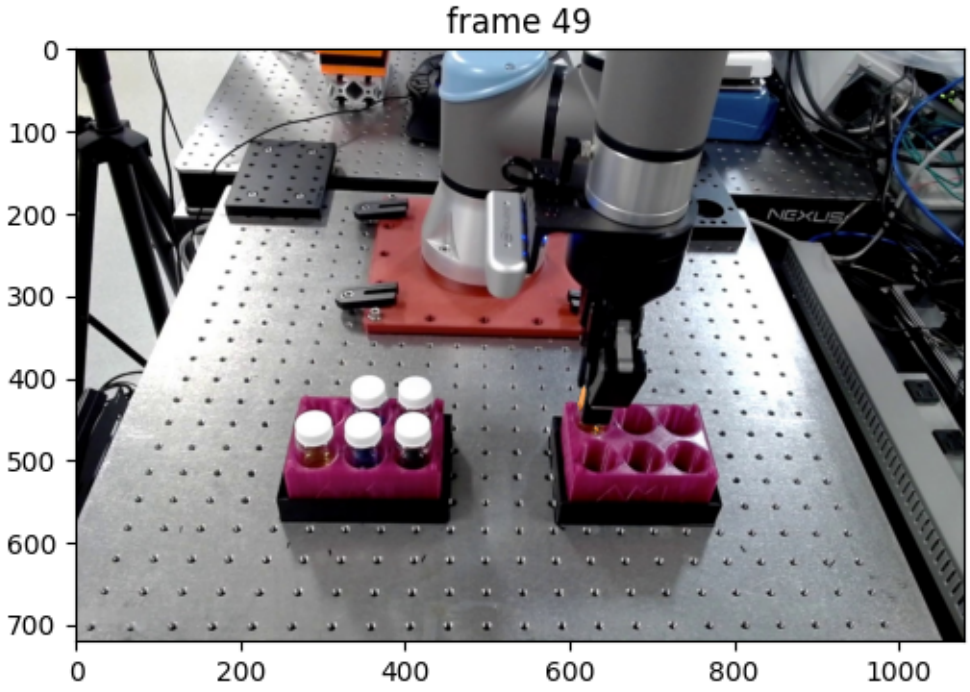

frame 44
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

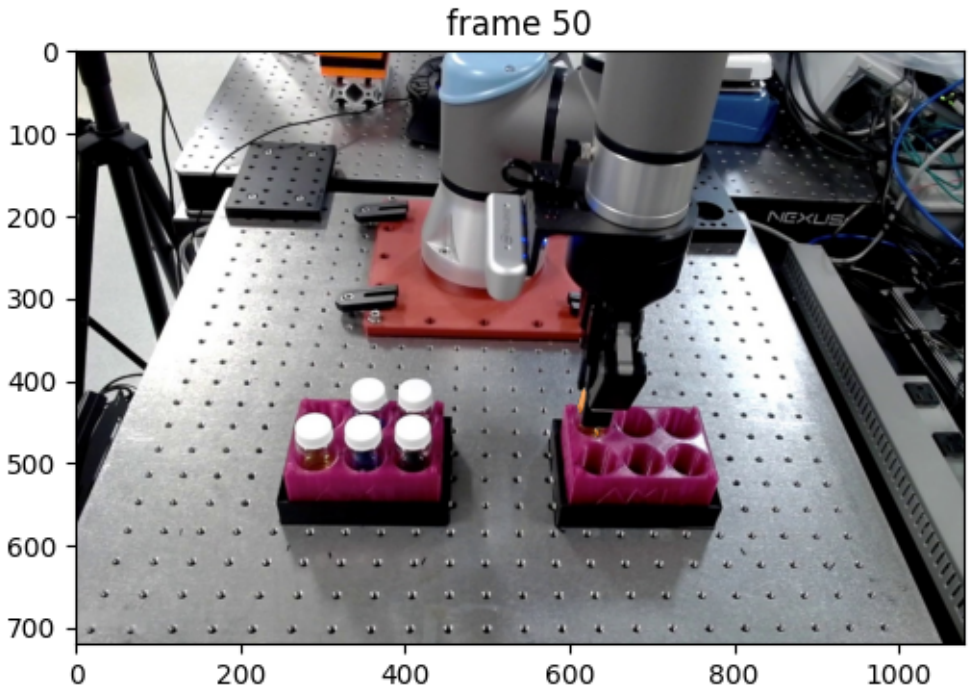

frame 45
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

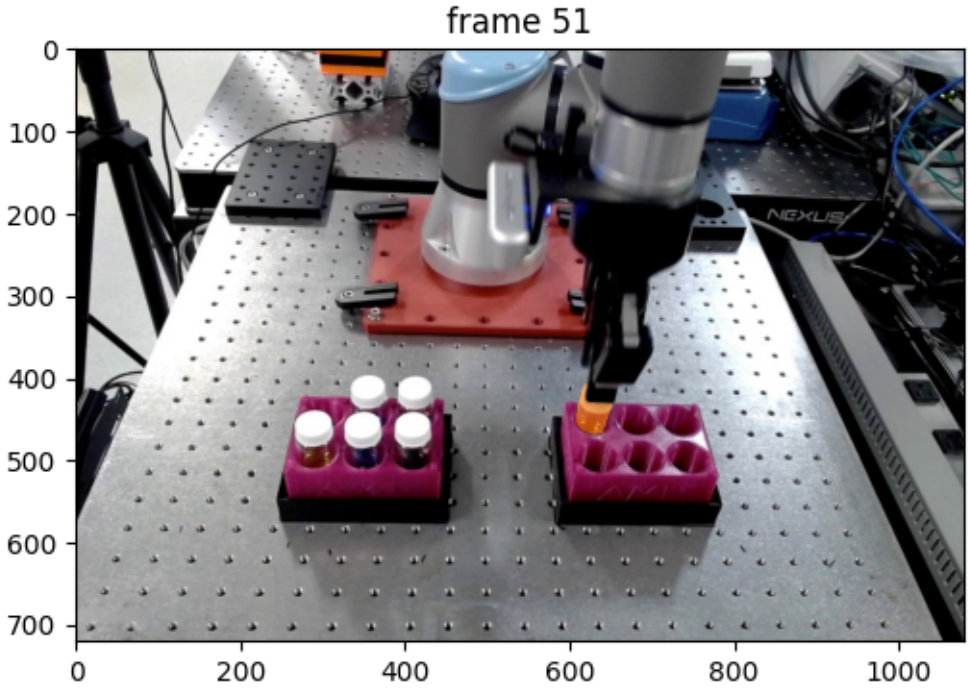

frame 46
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

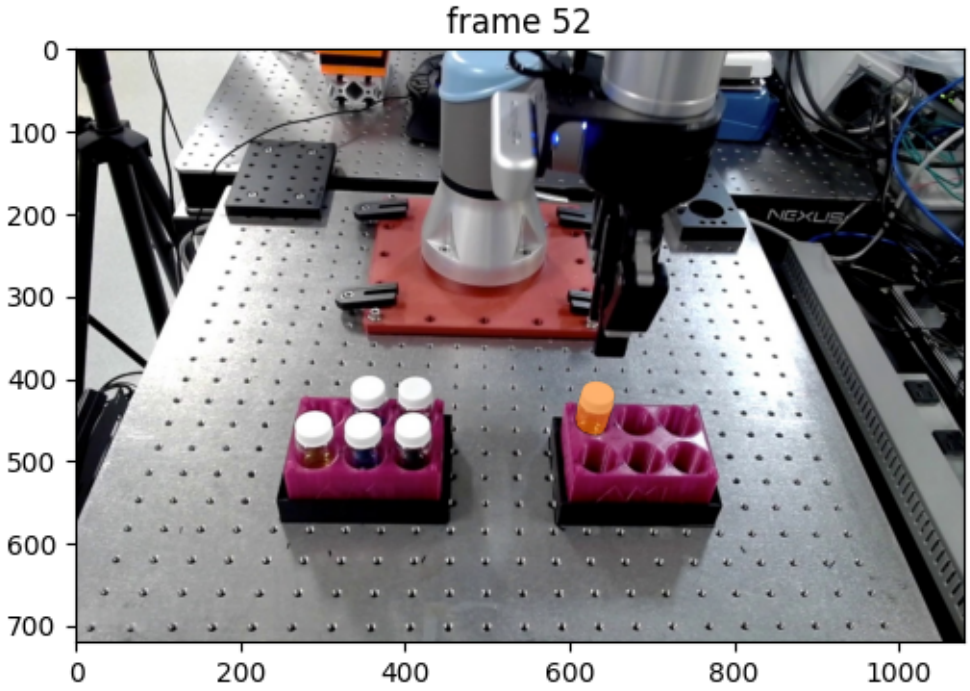

frame 47
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

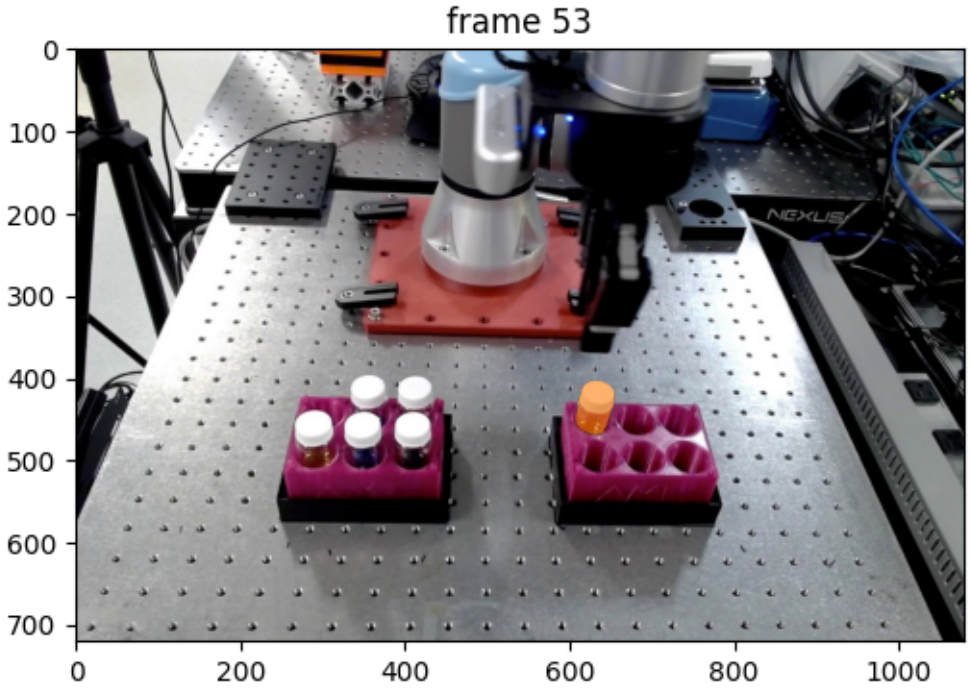

frame 48
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

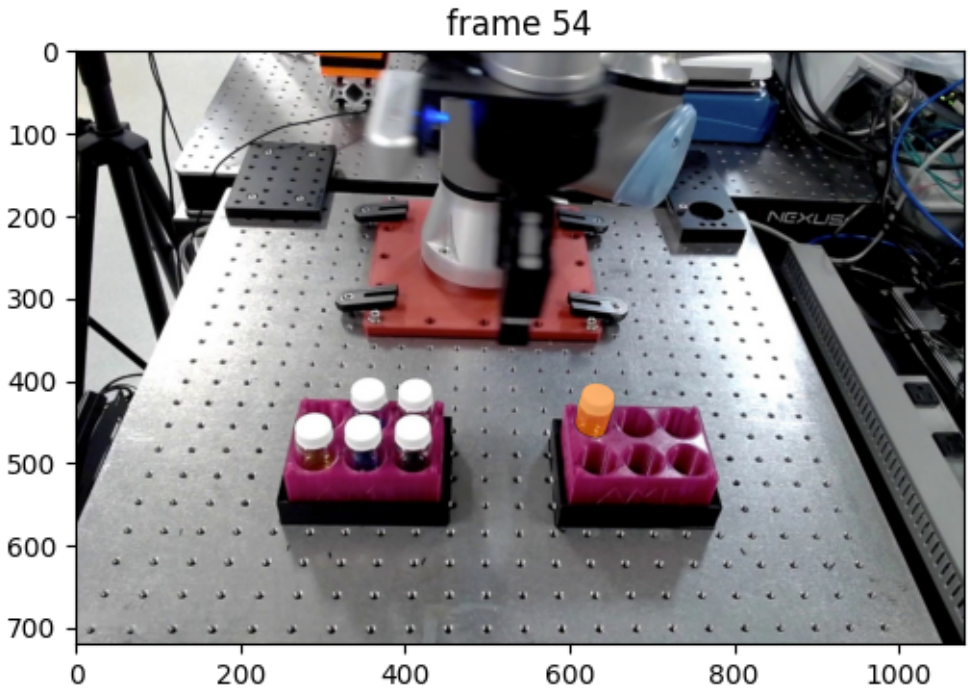

frame 49
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

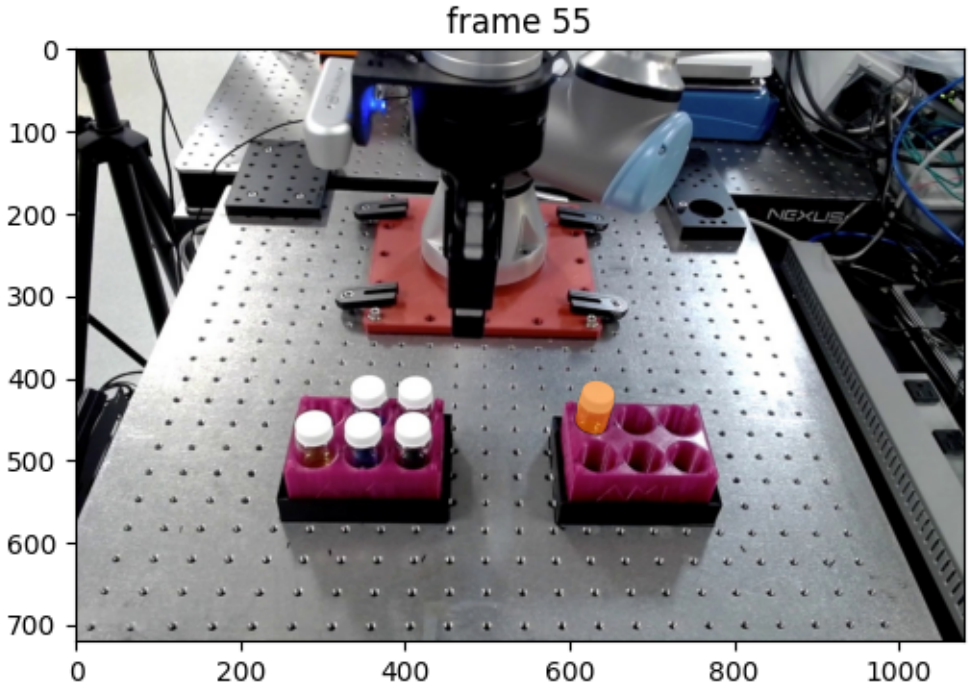

frame 50
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

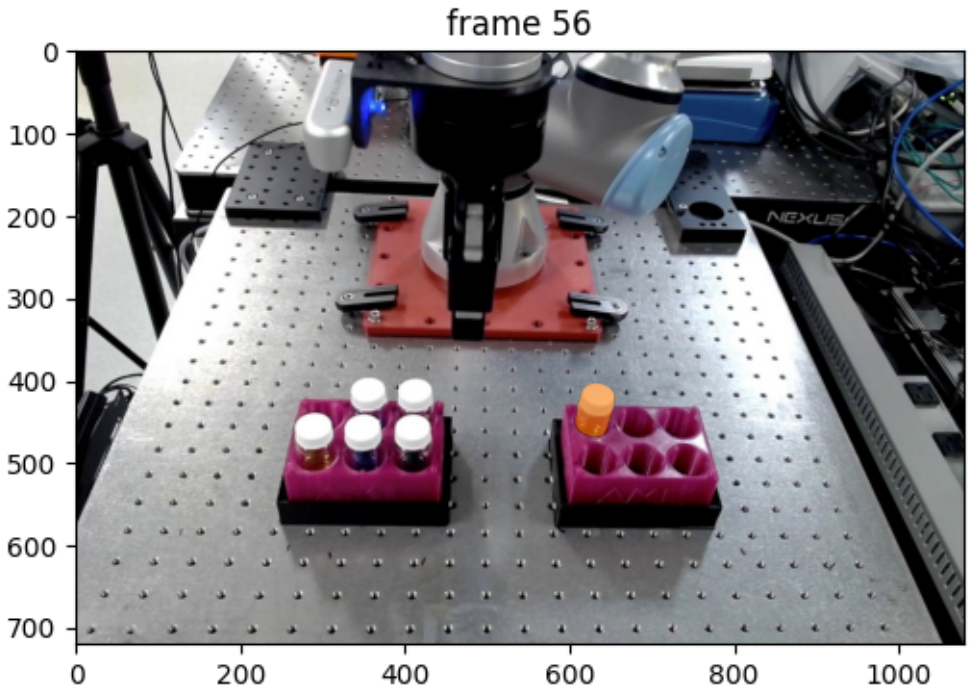

frame 51
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

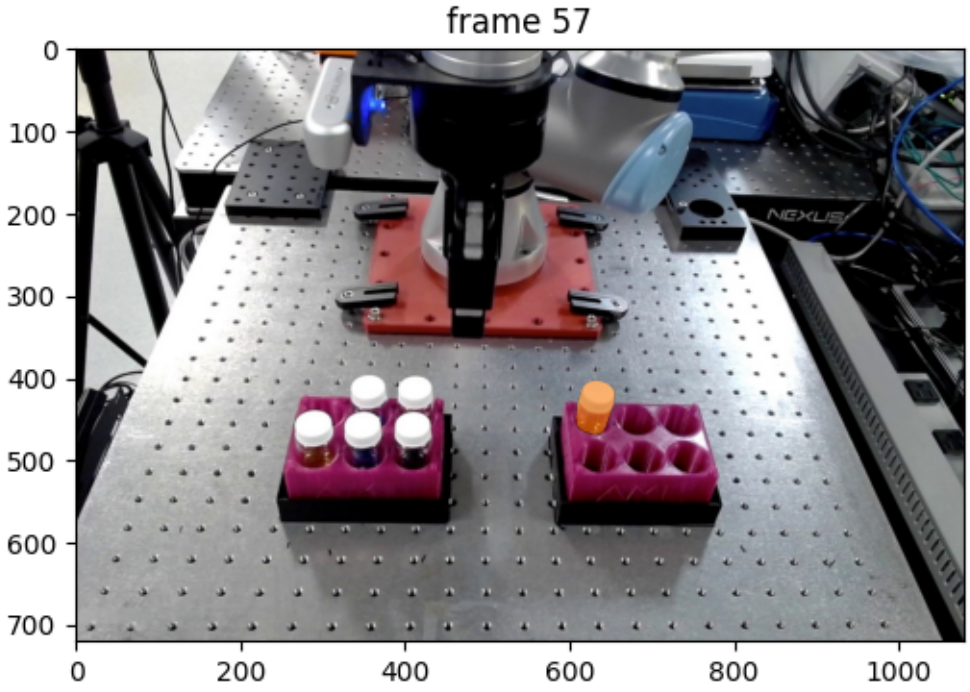

frame 52
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

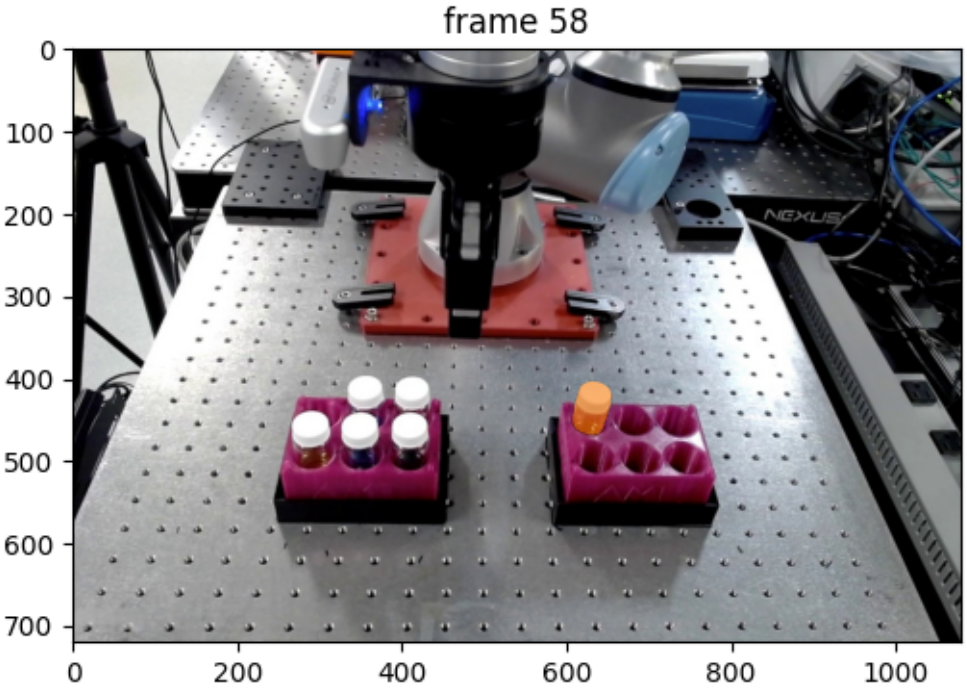

frame 53
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

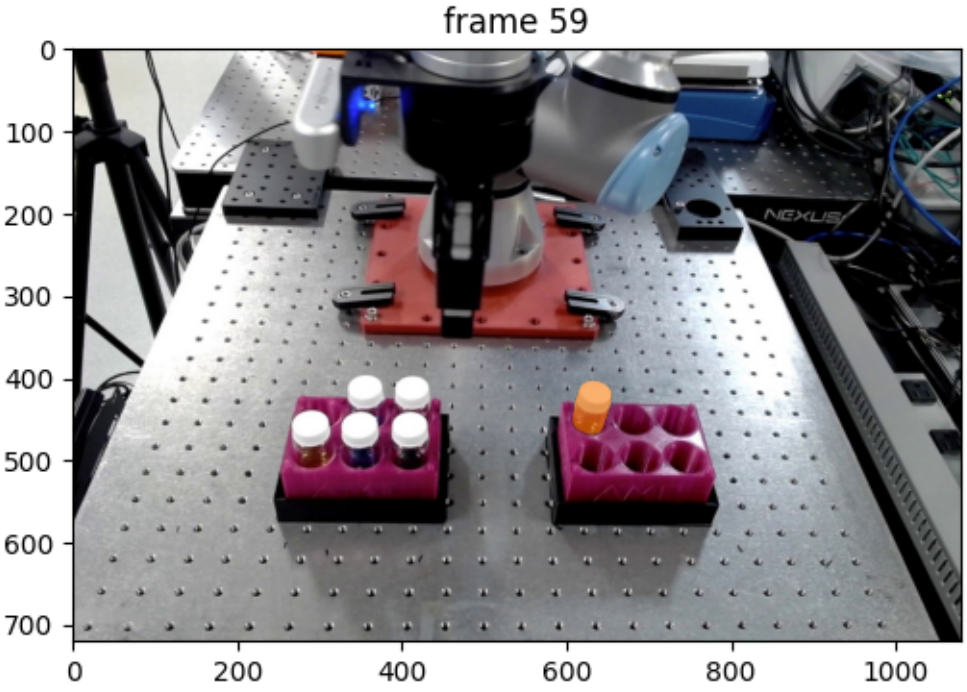

frame 54
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

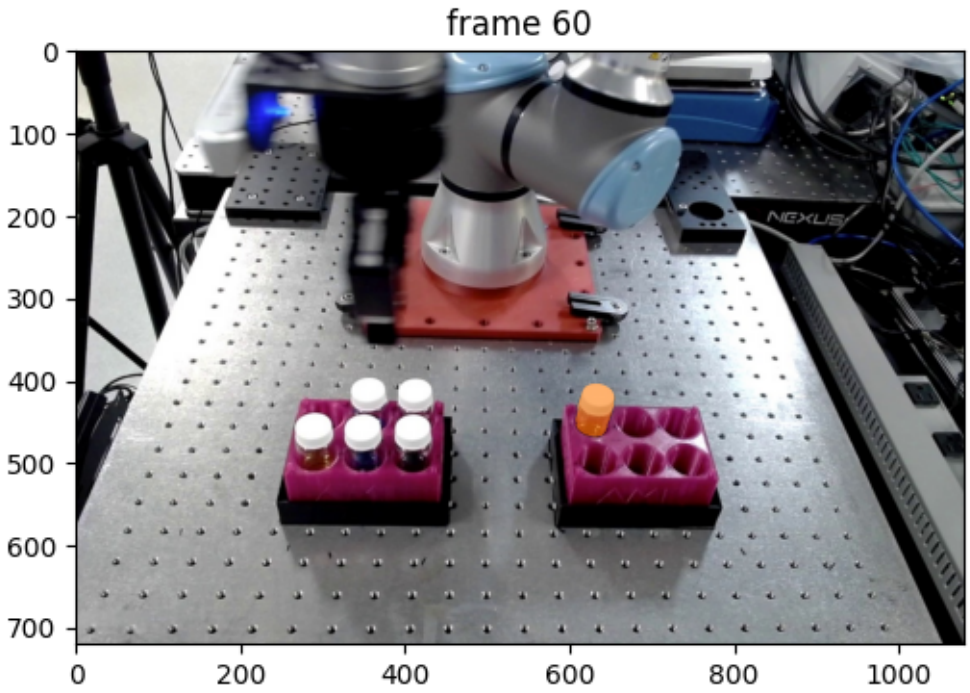

frame 55
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

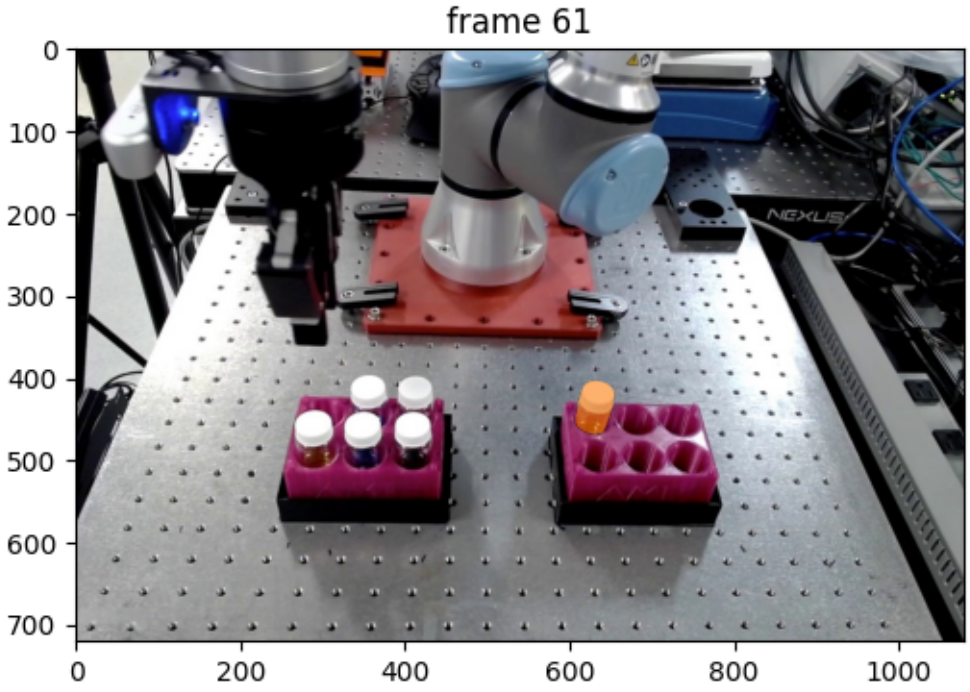

frame 56
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

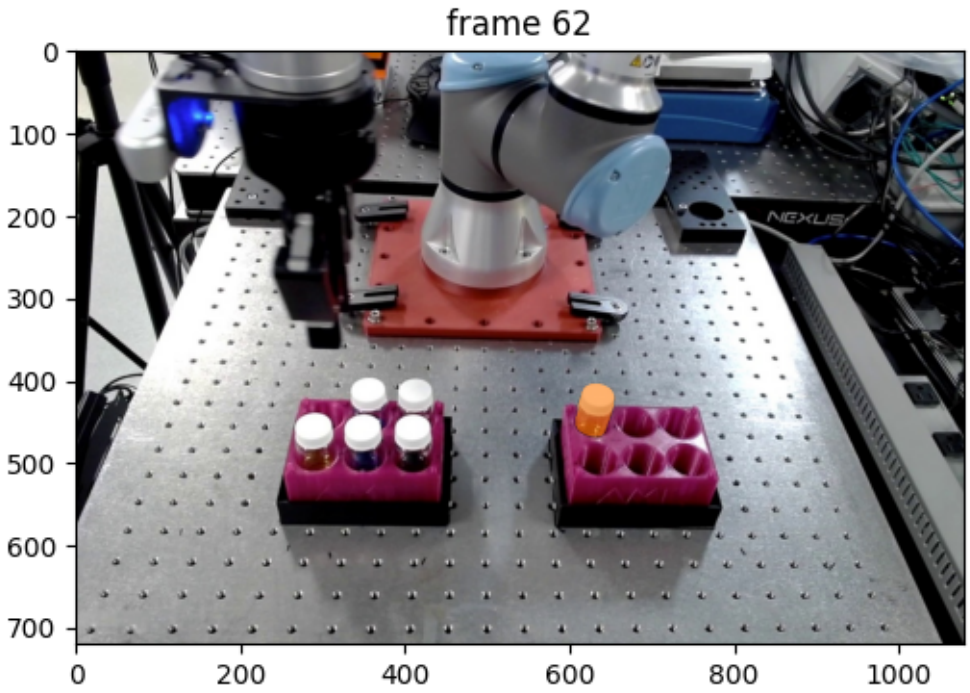

frame 57
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

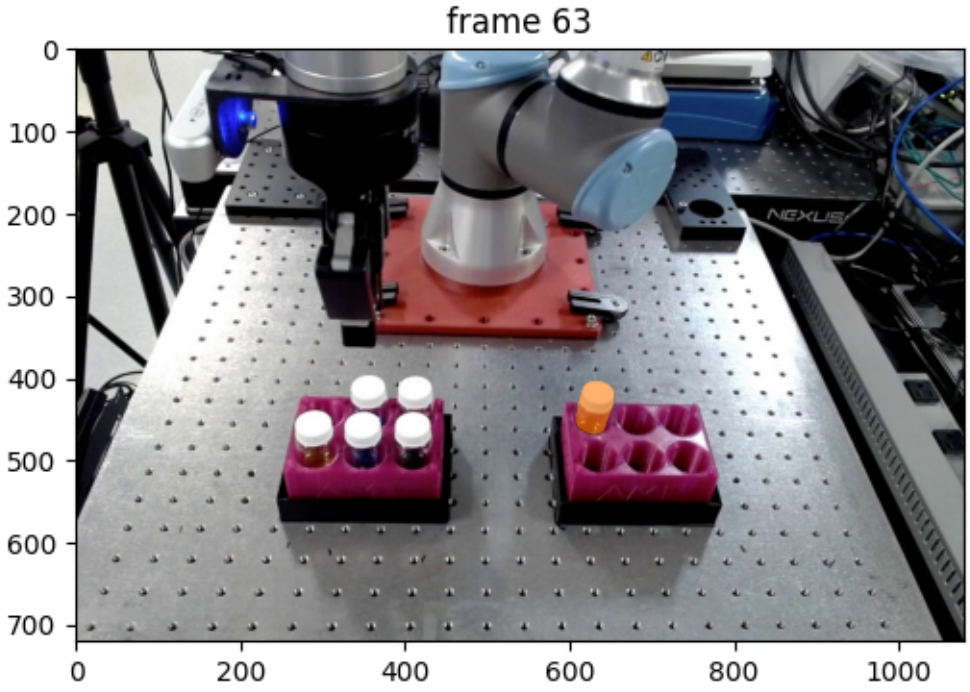

frame 58
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

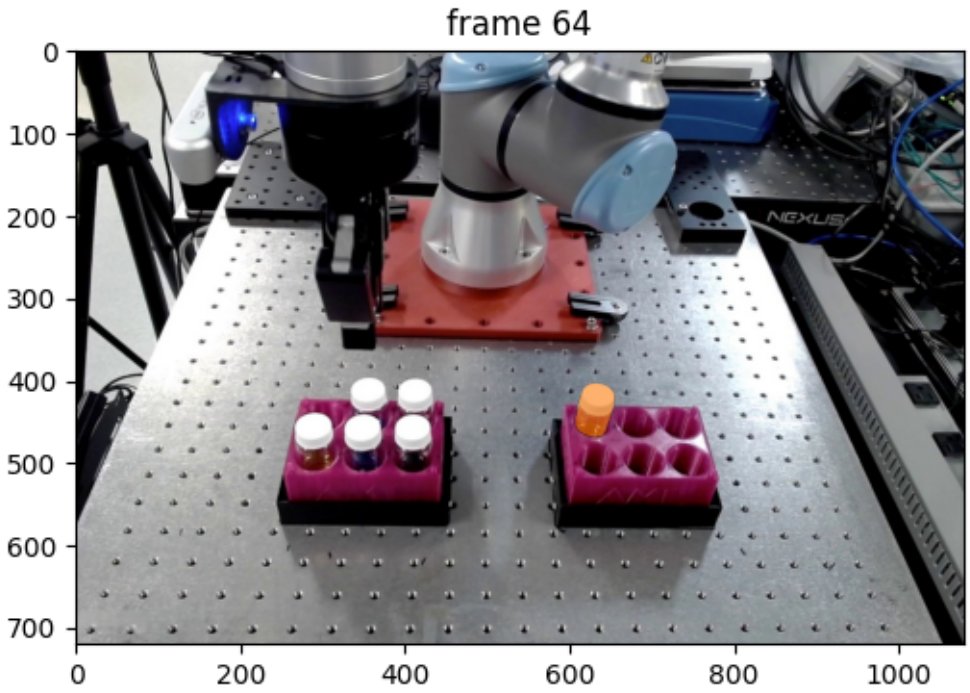

frame 59
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

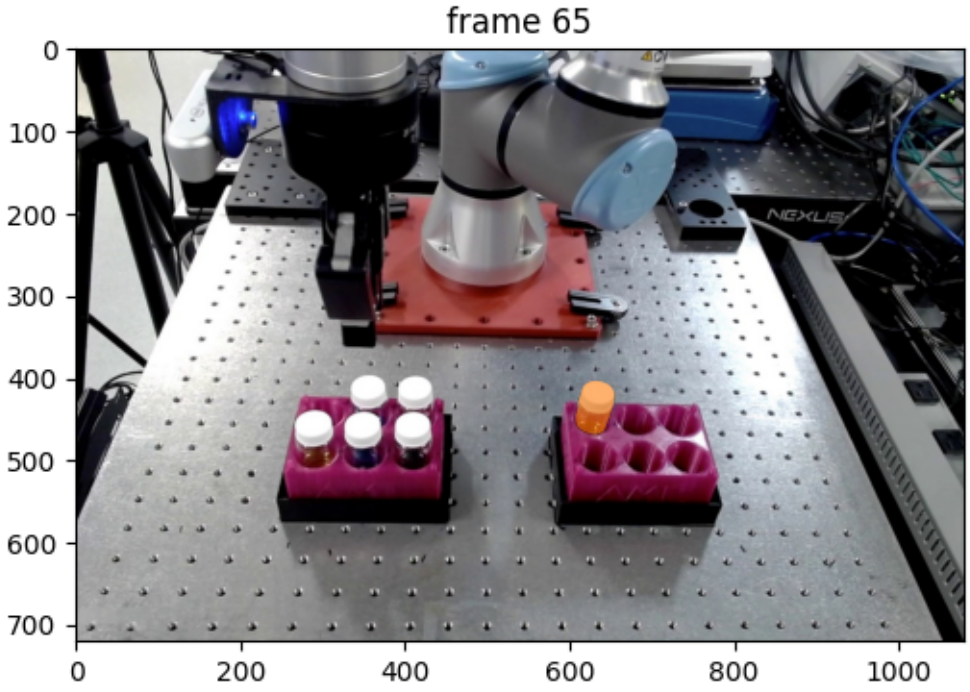

frame 60
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

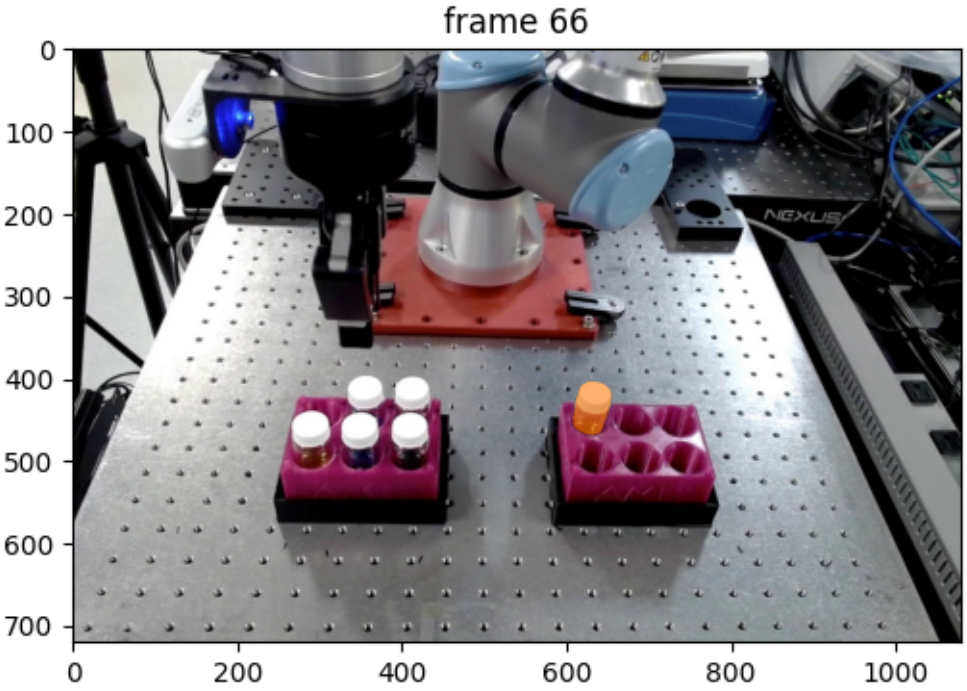

frame 61
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

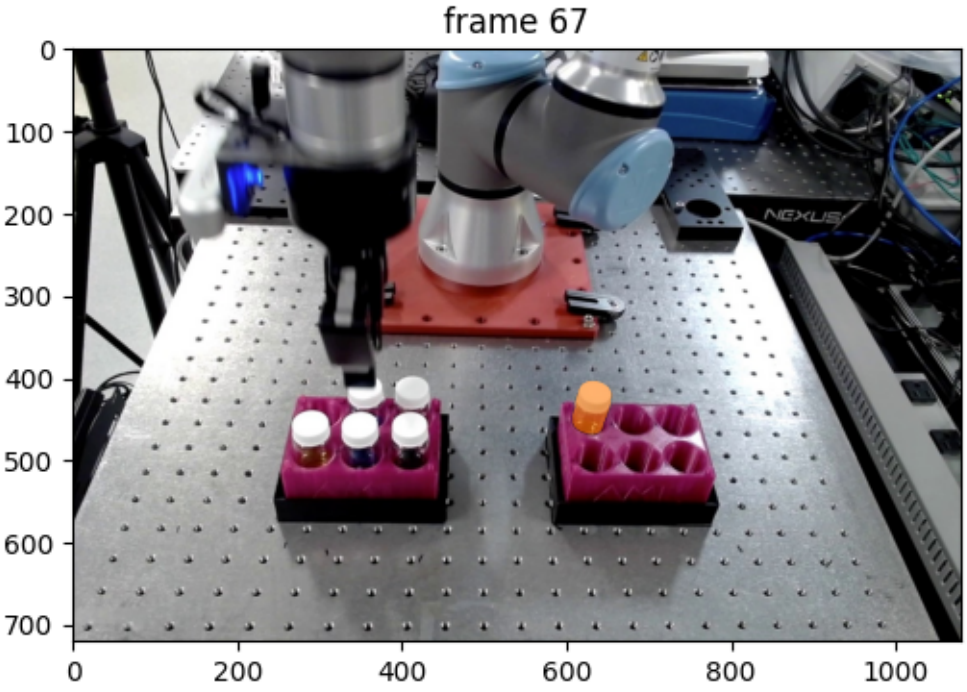

frame 62
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

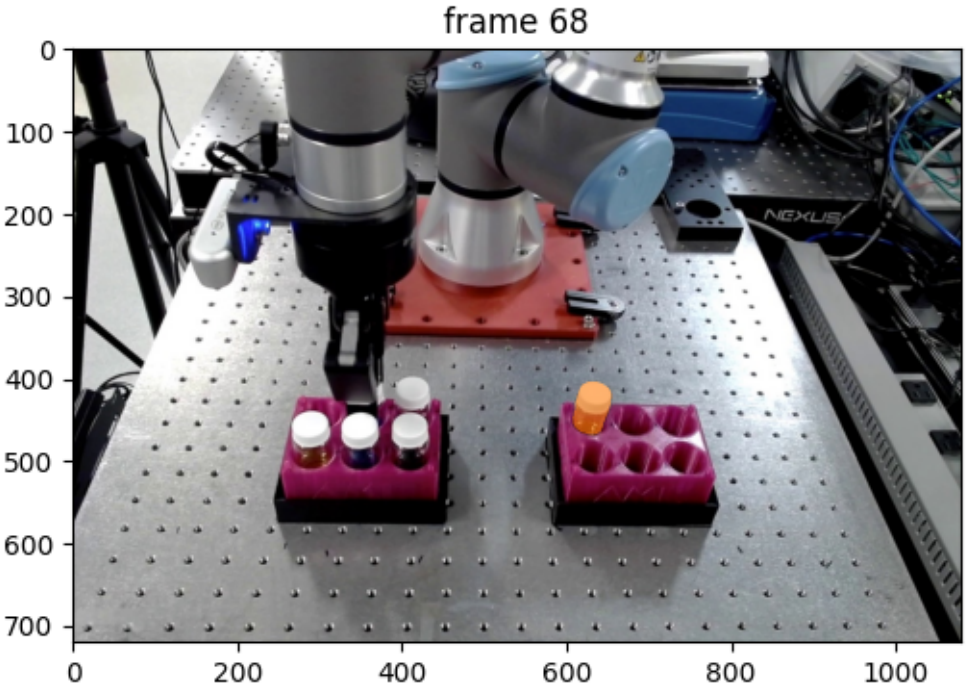

frame 63
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

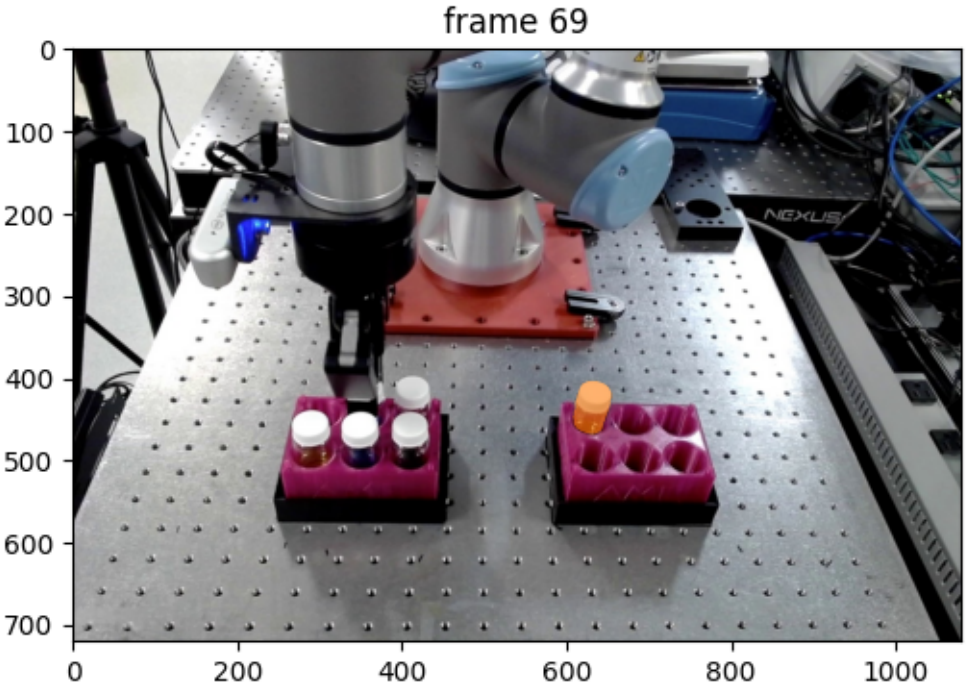

frame 64
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

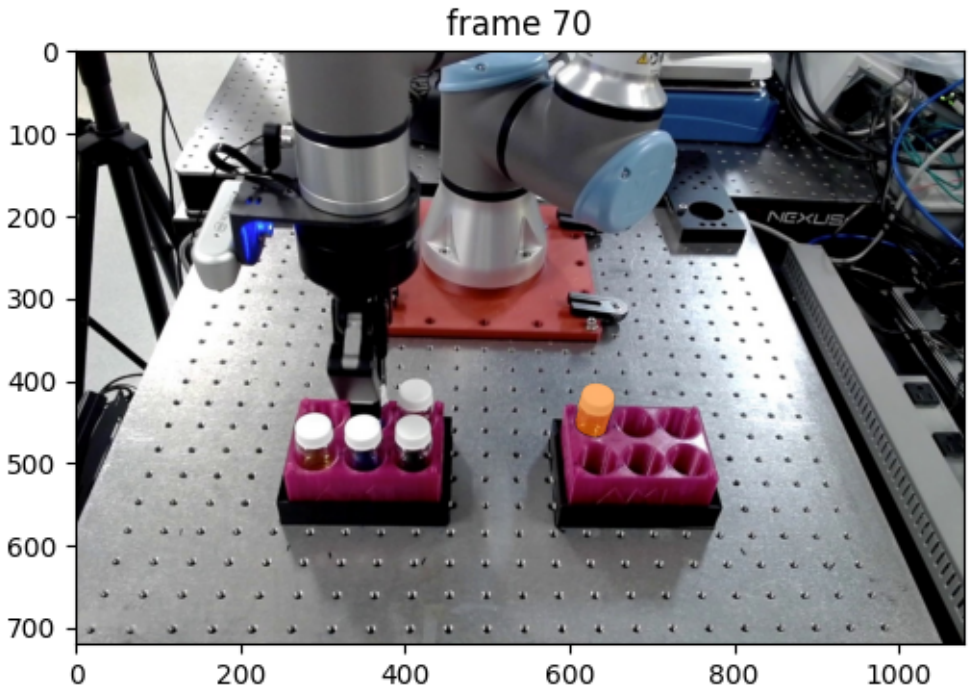

frame 65
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

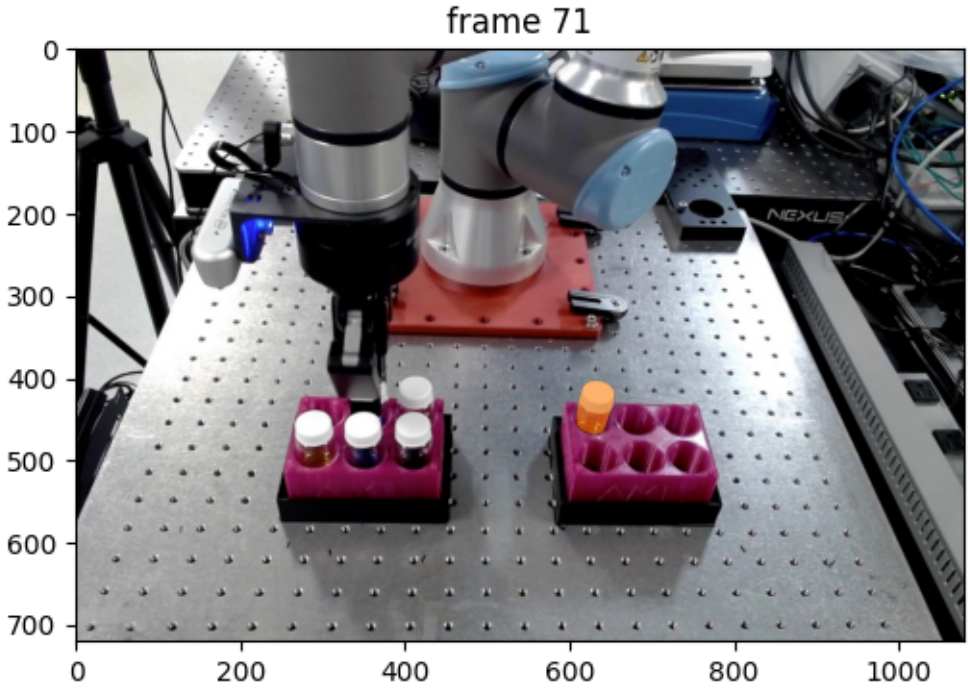

frame 66
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

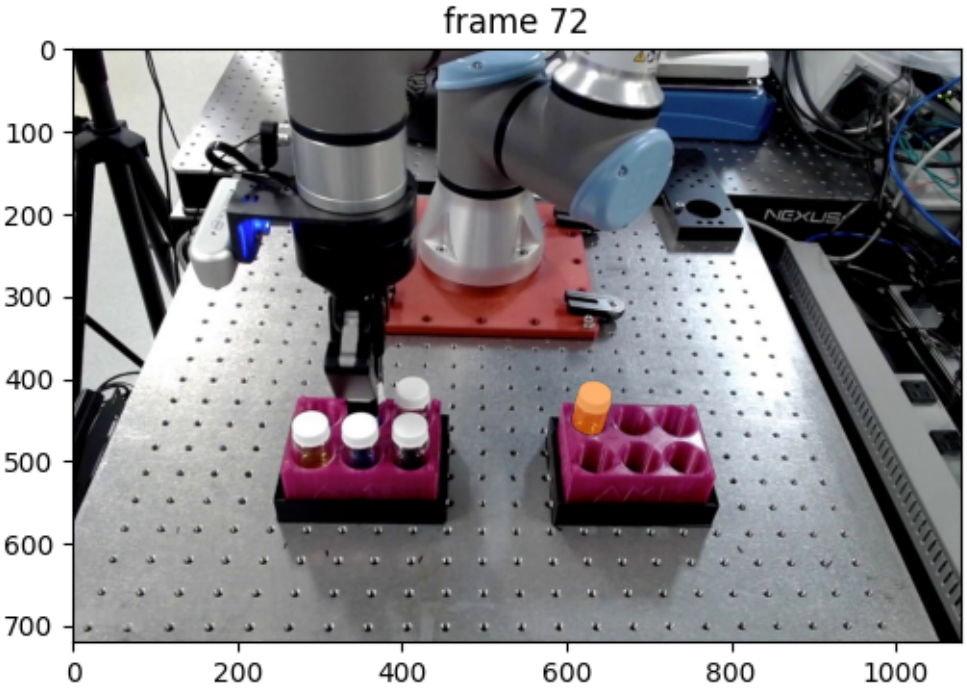

frame 67
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

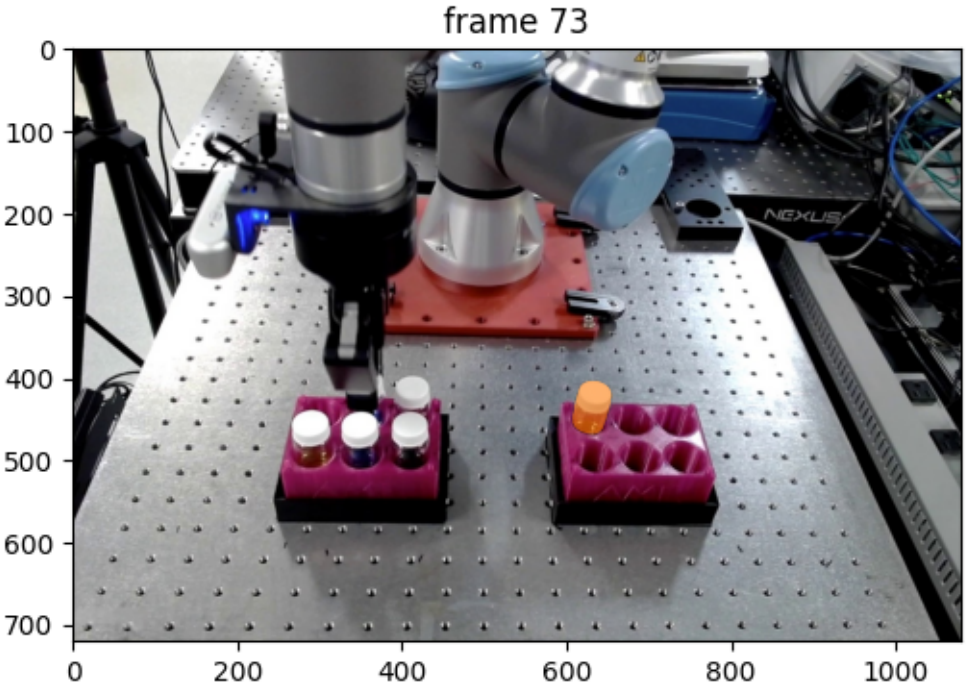

frame 68
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

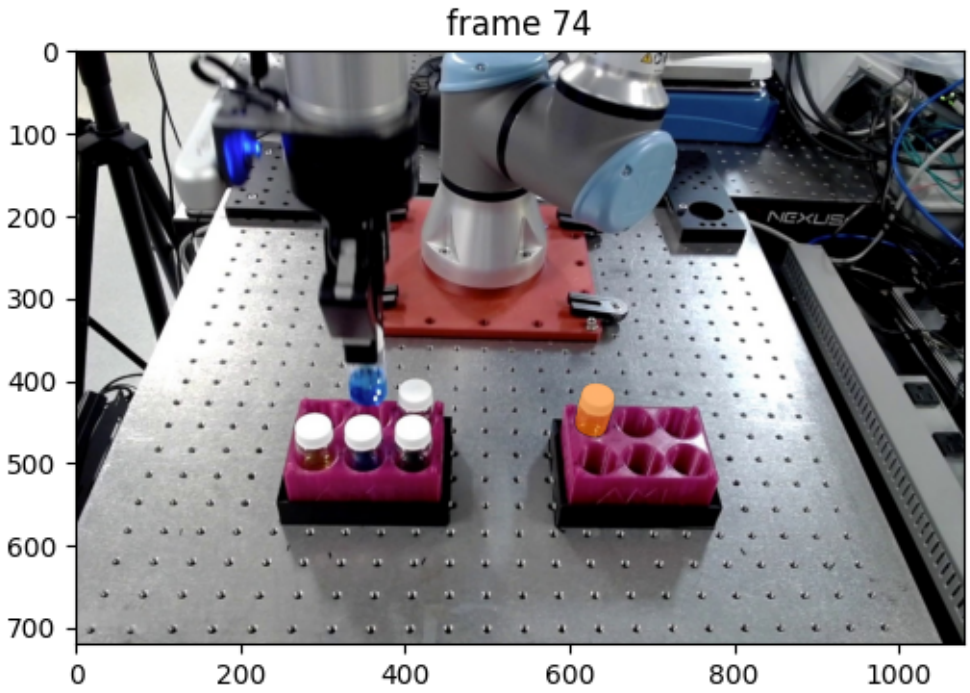

frame 69
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

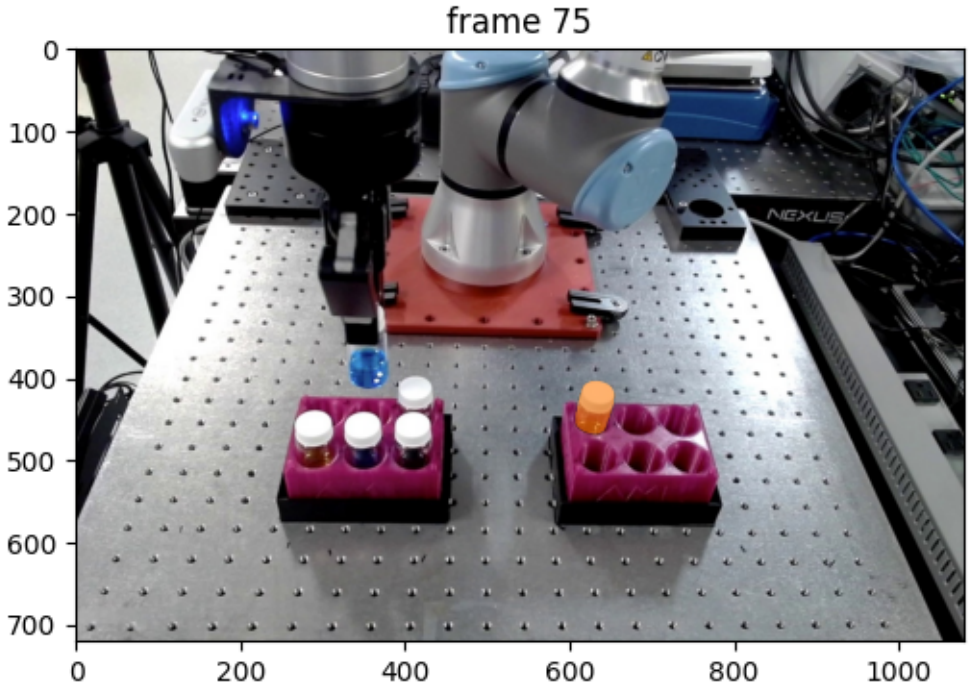

frame 70
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

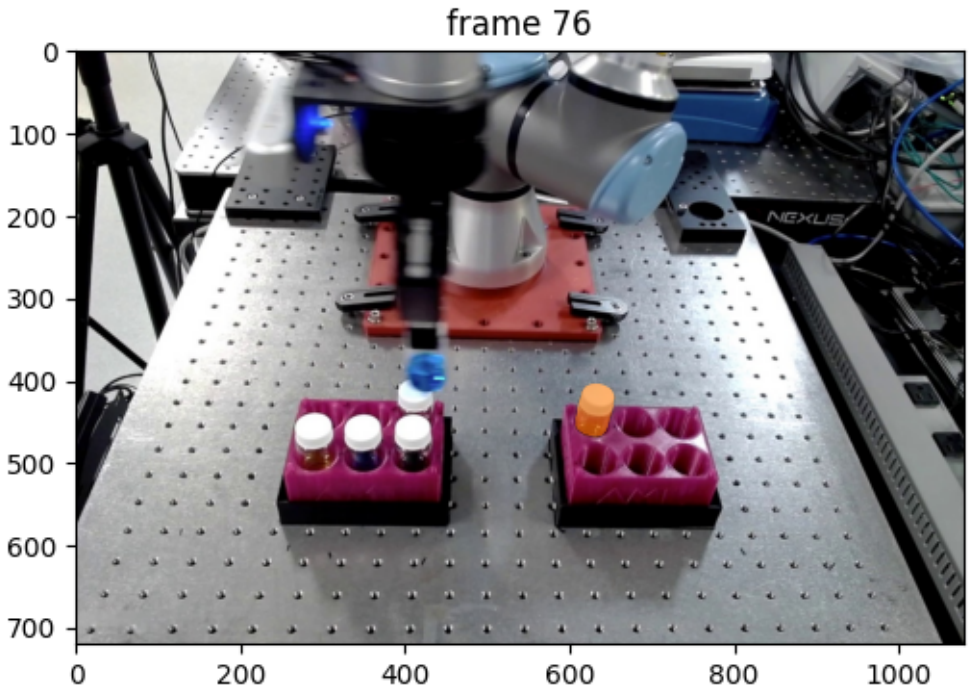

frame 71
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

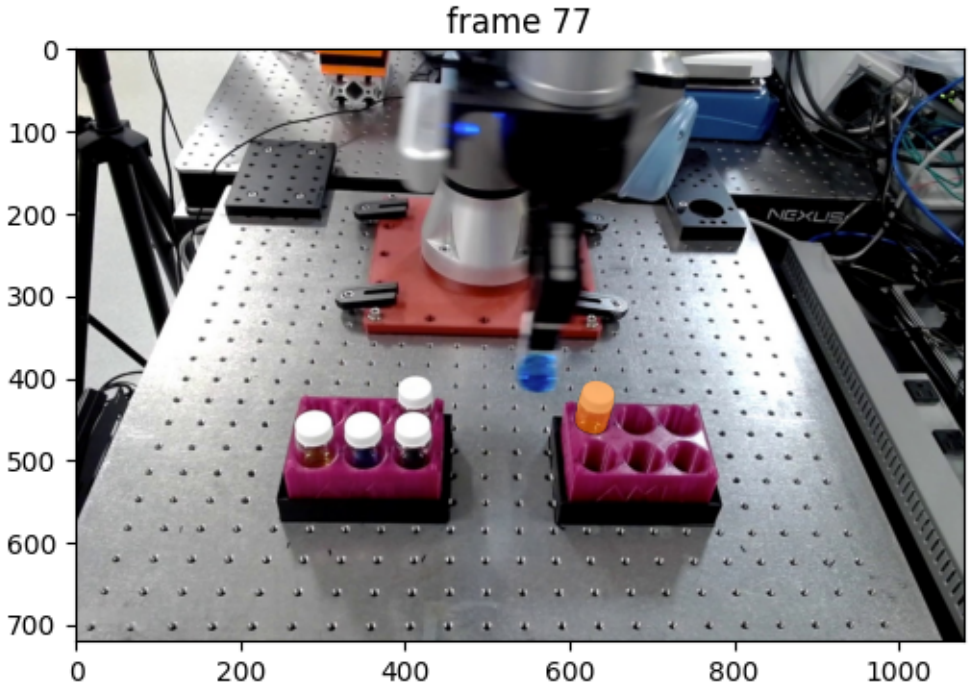

frame 72
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

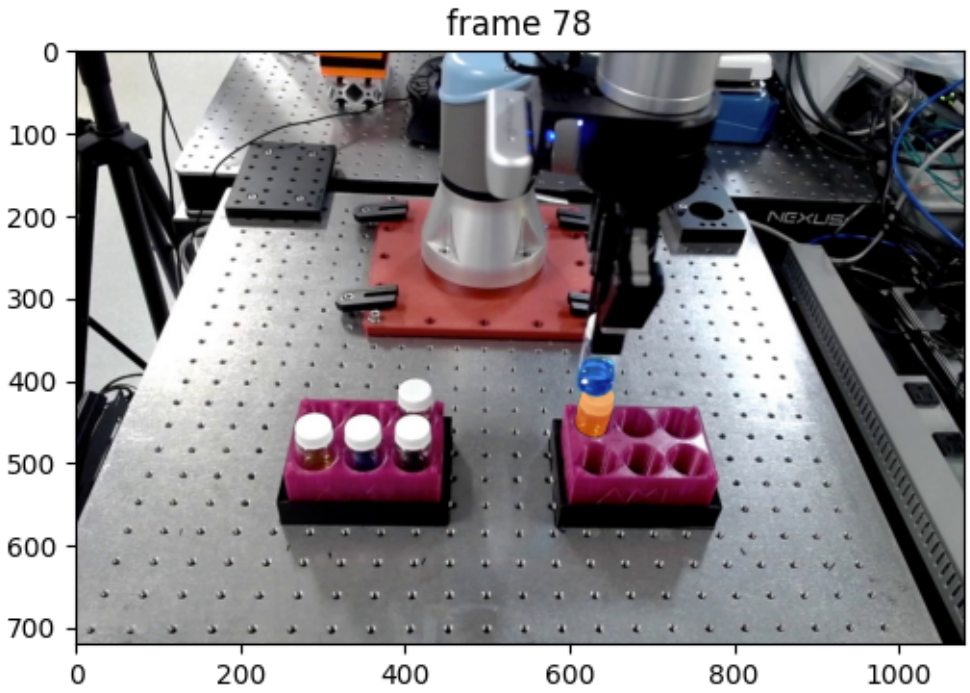

frame 73
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

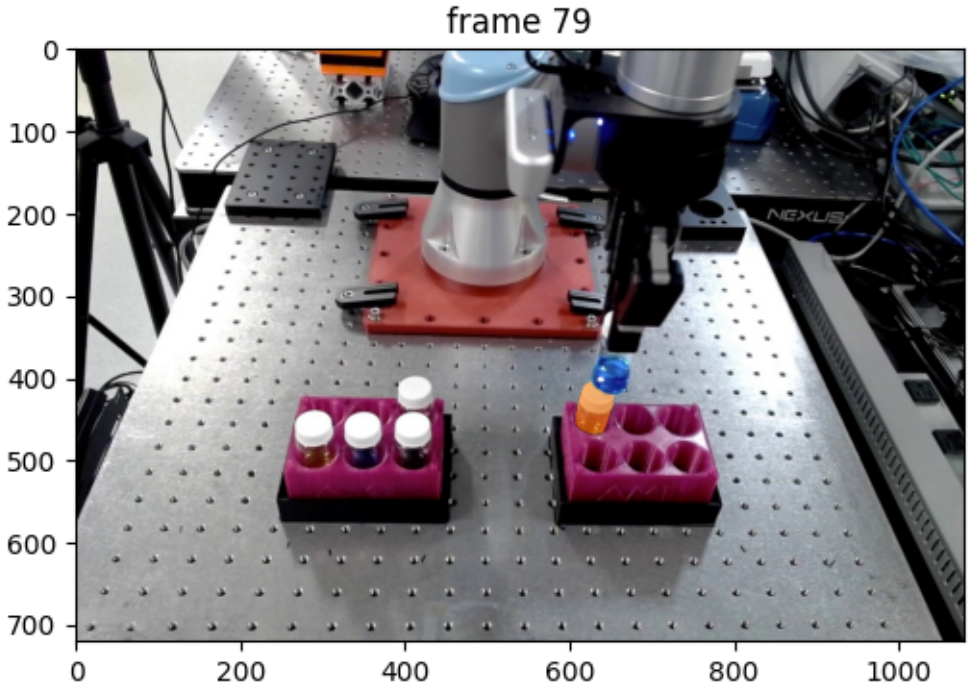

frame 74
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 75
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 76
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 77
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 78
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 79
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

Step 4: Add new prompts to further refine the masklet It appears that in the output masklet above, there are some small imperfections in boundary details on frame 150.

With SAM 2 we can fix the model predictions interactively. We can add a **negative click** at (x, y) = (82, 415) on this frame with label `0` to refine the masklet. Here we call the `add_new_points_or_box` API with a different `frame_idx` argument to indicate the frame index we want to refine.

[ ]:
```
if False: # Mask refinement
    ann_frame_idx = 150  # further refine some details on this frame
    ann_obj_id = 1  # give a unique id to the object we interact with (it can␣
↪be any integers)

    # show the segment before further refinement
    plt.figure(figsize=(9, 6))
    plt.title(f"frame {ann_frame_idx} -- before refinement")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
    show_mask(video_segments[ann_frame_idx][ann_obj_id], plt.gca(),␣
↪obj_id=ann_obj_id)

    # Let's add a negative click on this frame at (x, y) = (82, 415) to refine␣
↪the segment
    points = np.array([[82, 410]], dtype=np.float32)
    # for labels, `1` means positive click and `0` means negative click
    labels = np.array([0], np.int32)
    _, _, out_mask_logits = predictor.add_new_points_or_box(
        inference_state=inference_state,
        frame_idx=ann_frame_idx,
        obj_id=ann_obj_id,
        points=points,
        labels=labels,
    )

    # show the segment after the further refinement
    plt.figure(figsize=(9, 6))
    plt.title(f"frame {ann_frame_idx} -- after refinement")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
    show_points(points, labels, plt.gca())
    show_mask((out_mask_logits > 0.0).cpu().numpy(), plt.gca(),␣
↪obj_id=ann_obj_id)
```

Step 5: Propagate the prompts (again) to get the masklet across the video Let's get an updated masklet for the entire video. Here we call `propagate_in_video` again to propagate all the prompts after adding the new refinement click above.

[ ]:
```
if False: #propegate the refined mask
    # run propagation throughout the video and collect the results in a dict
    video_segments = {}  # video_segments contains the per-frame segmentation␣
↪results
```

```
    for out_frame_idx, out_obj_ids, out_mask_logits in predictor.
↪propagate_in_video(inference_state):
        video_segments[out_frame_idx] = {
            out_obj_id: (out_mask_logits[i] > 0.0).cpu().numpy()
            for i, out_obj_id in enumerate(out_obj_ids)
        }

    # render the segmentation results every few frames
    vis_frame_stride = 30
    plt.close("all")
    for out_frame_idx in range(0, len(frame_names), vis_frame_stride):
        plt.figure(figsize=(6, 4))
        plt.title(f"frame {out_frame_idx}")
        plt.imshow(Image.open(os.path.join(video_dir,␣
↪frame_names[out_frame_idx])))
        for out_obj_id, out_mask in video_segments[out_frame_idx].items():
            show_mask(out_mask, plt.gca(), obj_id=out_obj_id)
```

The segments now look good on all frames.

1.2.3 Example 2: Segment an object using box prompt

Note: if you have run any previous tracking using this `inference_state`, please reset it first via `reset_state`.

[ ]:
```
predictor.reset_state(inference_state)
```

In addition to using clicks as inputs, SAM 2 also supports segmenting and tracking objects in a video via **bounding boxes**.

In the example below, we segment the child on the right using a **box prompt** of (x_min, y_min, x_max, y_max) = (300, 0, 500, 400) on frame 0 as input into the `add_new_points_or_box` API.

[ ]:
```
if False: # use bounding boxes
    ann_frame_idx = 0  # the frame index we interact with
    ann_obj_id = 4  # give a unique id to each object we interact with (it can␣
↪be any integers)

    # Let's add a box at (x_min, y_min, x_max, y_max) = (300, 0, 500, 400) to␣
↪get started
    box = np.array([300, 0, 500, 400], dtype=np.float32)
    _, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
        inference_state=inference_state,
        frame_idx=ann_frame_idx,
        obj_id=ann_obj_id,
        box=box,
    )

    # show the results on the current (interacted) frame
```

```
    plt.figure(figsize=(9, 6))
    plt.title(f"frame {ann_frame_idx}")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
    show_box(box, plt.gca())
    show_mask((out_mask_logits[0] > 0.0).cpu().numpy(), plt.gca(),␣
↪obj_id=out_obj_ids[0])
```

Here, SAM 2 gets a pretty good segmentation mask of the entire child, even though the input bounding box is not perfectly tight around the object.

Similar to the previous example, if the returned mask from is not perfect when using a box prompt, we can also further **refine** the output using positive or negative clicks. To illustrate this, here we make a **positive click** at (x, y) = (460, 60) with label `1` to expand the segment around the child's hair.

Note: to refine the segmentation mask from a box prompt, we need to send **both the original box input and all subsequent refinement clicks and their labels** when calling `add_new_points_or_box`.

[ ]:
```
if False: # using bounding boxes
    ann_frame_idx = 0  # the frame index we interact with
    ann_obj_id = 4  # give a unique id to each object we interact with (it can␣
↪be any integers)

    # Let's add a positive click at (x, y) = (460, 60) to refine the mask
    points = np.array([[460, 60]], dtype=np.float32)
    # for labels, `1` means positive click and `0` means negative click
    labels = np.array([1], np.int32)
    # note that we also need to send the original box input along with
    # the new refinement click together into `add_new_points_or_box`
    box = np.array([300, 0, 500, 400], dtype=np.float32)
    _, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
        inference_state=inference_state,
        frame_idx=ann_frame_idx,
        obj_id=ann_obj_id,
        points=points,
        labels=labels,
        box=box,
    )

    # show the results on the current (interacted) frame
    plt.figure(figsize=(9, 6))
    plt.title(f"frame {ann_frame_idx}")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
    show_box(box, plt.gca())
    show_points(points, labels, plt.gca())
    show_mask((out_mask_logits[0] > 0.0).cpu().numpy(), plt.gca(),␣
↪obj_id=out_obj_ids[0])
```

Then, to get the masklet throughout the entire video, we propagate the prompts using the `propagate_in_video` API.

[ ]:
```
if False:
    # run propagation throughout the video and collect the results in a dict
    video_segments = {}  # video_segments contains the per-frame segmentation␣
 ↪results
    for out_frame_idx, out_obj_ids, out_mask_logits in predictor.
 ↪propagate_in_video(inference_state):
        video_segments[out_frame_idx] = {
            out_obj_id: (out_mask_logits[i] > 0.0).cpu().numpy()
            for i, out_obj_id in enumerate(out_obj_ids)
        }

    # render the segmentation results every few frames
    vis_frame_stride = 30
    plt.close("all")
    for out_frame_idx in range(0, len(frame_names), vis_frame_stride):
        plt.figure(figsize=(6, 4))
        plt.title(f"frame {out_frame_idx}")
        plt.imshow(Image.open(os.path.join(video_dir,␣
 ↪frame_names[out_frame_idx])))
        for out_obj_id, out_mask in video_segments[out_frame_idx].items():
            show_mask(out_mask, plt.gca(), obj_id=out_obj_id)
```

Note that in addition to clicks or boxes, SAM 2 also supports directly using a **mask prompt** as input via the `add_new_mask` method in the `SAM2VideoPredictor` class. This can be helpful in e.g. semi-supervised VOS evaluations (see tools/vos_inference.py for an example).

1.2.4 Example 3: Segment multiple objects simultaneously

Note: if you have run any previous tracking using this `inference_state`, please reset it first via `reset_state`.

[21]:
```
predictor.reset_state(inference_state)
```

Step 1: Add two objects on a frame SAM 2 can also segment and track two or more objects at the same time. One way, of course, is to do them one by one. However, it would be more efficient to batch them together (e.g. so that we can share the image features between objects to reduce computation costs).

This time, let's focus on object parts and segment **the shirts of both childen** in this video. Here we add prompts for these two objects and assign each of them a unique object id.

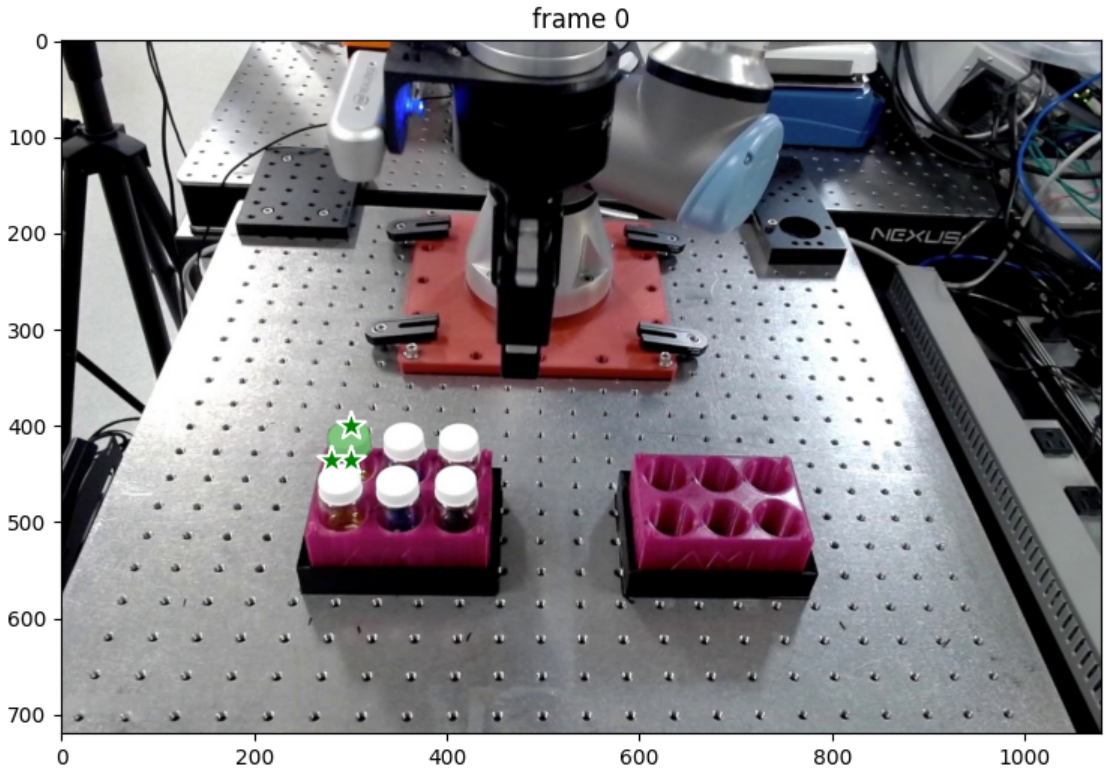
[22]:
```
prompts = {}  # hold all the clicks we add for visualization
```

Add the first object (the left child's shirt) with a **positive click** at (x, y) = (200, 300) on frame 0.

We assign it to object id 2 (it can be arbitrary integers, and only needs to be unique for each object to track), which is passed to the `add_new_points_or_box` API to distinguish the object we are

clicking upon.

[23]:
```
ann_frame_idx = 0  # the frame index we interact with
ann_obj_id = 2  # give a unique id to each object we interact with (it can be␣
 ↪any integers)

#
points = np.array([[300, 400], [300,435], [280,435]], dtype=np.float32)

# for labels, `1` means positive click and `0` means negative click
labels = np.array([1,1,1], np.int32)

prompts[ann_obj_id] = points, labels
_, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
    inference_state=inference_state,
    frame_idx=ann_frame_idx,
    obj_id=ann_obj_id,
    points=points,
    labels=labels,
)

# show the results on the current (interacted) frame
plt.figure(figsize=(9, 6))
plt.title(f"frame {ann_frame_idx}")
plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
show_points(points, labels, plt.gca())
for i, out_obj_id in enumerate(out_obj_ids):
    show_points(*prompts[out_obj_id], plt.gca())
    show_mask((out_mask_logits[i] > 0.0).cpu().numpy(), plt.gca(),␣
 ↪obj_id=out_obj_id)
```



Hmm, this time we just want to select the child's shirt, but the model predicts the mask for the entire child. Let's refine the prediction with a **negative click** at (x, y) = (275, 175).

[24]:
```
if False: # refine ann_obj_id=2 with a negative click
    # add the first object
    ann_frame_idx = 0  # the frame index we interact with
    ann_obj_id = 2  # give a unique id to each object we interact with (it can␣
↪be any integers)

    # Let's add a 2nd negative click at (x, y) = (275, 175) to refine the first␣
↪object
    # sending all clicks (and their labels) to `add_new_points_or_box`
    points = np.array([[200, 300], [275, 175]], dtype=np.float32)
    # for labels, `1` means positive click and `0` means negative click
    labels = np.array([1, 0], np.int32)
    prompts[ann_obj_id] = points, labels
    _, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
        inference_state=inference_state,
        frame_idx=ann_frame_idx,
        obj_id=ann_obj_id,
        points=points,
        labels=labels,
```

```
    )

    # show the results on the current (interacted) frame
    plt.figure(figsize=(9, 6))
    plt.title(f"frame {ann_frame_idx}")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
    show_points(points, labels, plt.gca())
    for i, out_obj_id in enumerate(out_obj_ids):
        show_points(*prompts[out_obj_id], plt.gca())
        show_mask((out_mask_logits[i] > 0.0).cpu().numpy(), plt.gca(),␣
↪obj_id=out_obj_id)
```

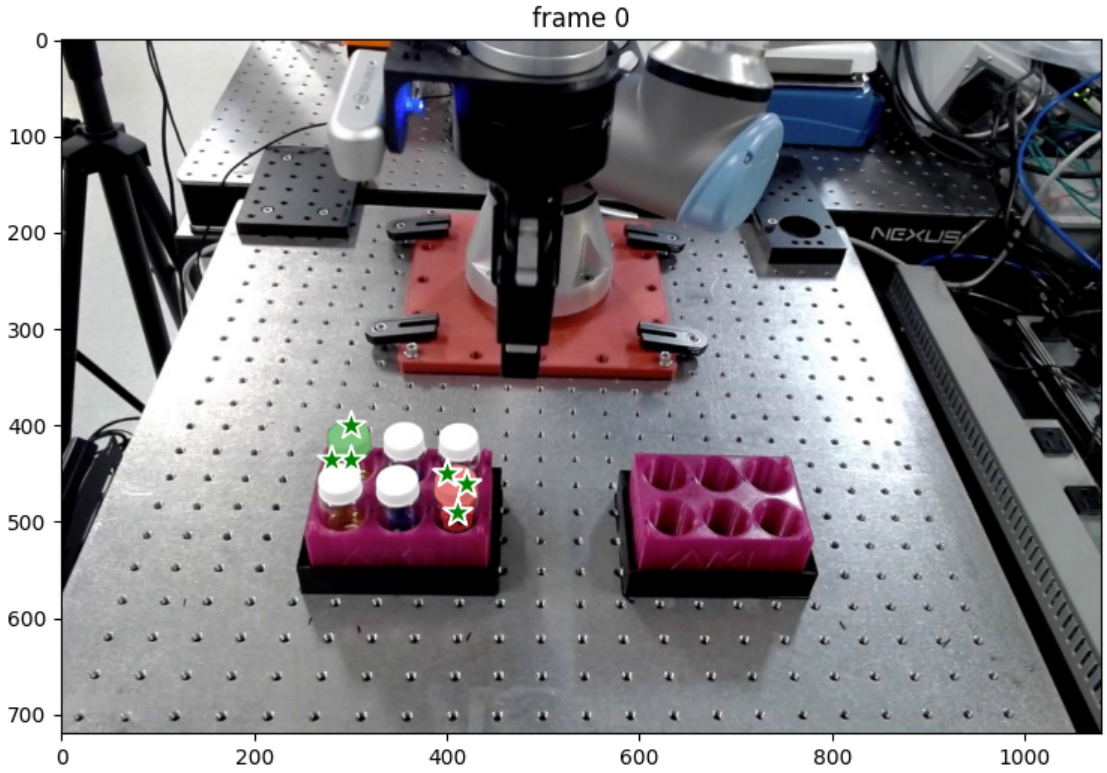
After the 2nd negative click, now we get the left child's shirt as our first object.

Let's move on to the second object (the right child's shirt) with a positive click at (x, y) = (400, 150) on frame 0. Here we assign object id `3` to this second object (it can be arbitrary integers, and only needs to be unique for each object to track).

Note: when there are multiple objects, the `add_new_points_or_box` API will return a list of masks for each object.

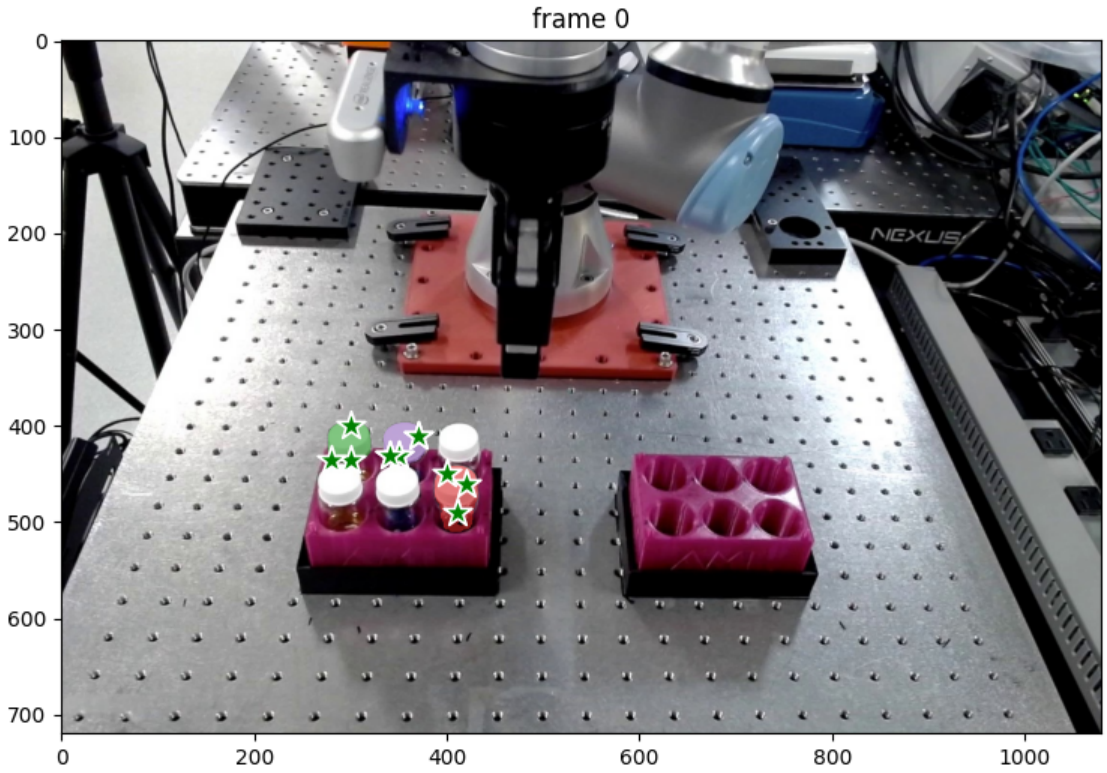
[25]:
```
# Add another object - id3
ann_frame_idx = 0  # the frame index we interact with
ann_obj_id = 3  # give a unique id to each object we interact with (it can be␣
↪any integers)

# Let's now move on to the second object we want to track (giving it object id␣
↪`3`)
# with a positive click at (x, y) = (400, 150)
points = np.array([[400, 450], [420,460], [410,490]], dtype=np.float32)
# for labels, `1` means positive click and `0` means negative click
labels = np.array([1,1,1], np.int32)
prompts[ann_obj_id] = points, labels

# `add_new_points_or_box` returns masks for all objects added so far on this␣
↪interacted frame
_, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
    inference_state=inference_state,
    frame_idx=ann_frame_idx,
    obj_id=ann_obj_id,
    points=points,
    labels=labels,
)

# show the results on the current (interacted) frame on all objects
plt.figure(figsize=(9, 6))
plt.title(f"frame {ann_frame_idx}")
plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
```

```
show_points(points, labels, plt.gca())
for i, out_obj_id in enumerate(out_obj_ids):
    show_points(*prompts[out_obj_id], plt.gca())
    show_mask((out_mask_logits[i] > 0.0).cpu().numpy(), plt.gca(),␣
 ↪obj_id=out_obj_id)
```



[26]:
```
# Add another object - id3
ann_frame_idx = 0  # the frame index we interact with
ann_obj_id = 4  # give a unique id to each object we interact with (it can be␣
 ↪any integers)

# Let's now move on to the second object we want to track (giving it object id␣
 ↪`3`)
# with a positive click at (x, y) = (400, 150)
points = np.array([[370, 410], [350,430], [340,430]], dtype=np.float32)
# for labels, `1` means positive click and `0` means negative click
labels = np.array([1,1,1], np.int32)
prompts[ann_obj_id] = points, labels

# `add_new_points_or_box` returns masks for all objects added so far on this␣
 ↪interacted frame
```

```
_, out_obj_ids, out_mask_logits = predictor.add_new_points_or_box(
    inference_state=inference_state,
    frame_idx=ann_frame_idx,
    obj_id=ann_obj_id,
    points=points,
    labels=labels,
)

# show the results on the current (interacted) frame on all objects
plt.figure(figsize=(9, 6))
plt.title(f"frame {ann_frame_idx}")
plt.imshow(Image.open(os.path.join(video_dir, frame_names[ann_frame_idx])))
show_points(points, labels, plt.gca())
for i, out_obj_id in enumerate(out_obj_ids):
    show_points(*prompts[out_obj_id], plt.gca())
    show_mask((out_mask_logits[i] > 0.0).cpu().numpy(), plt.gca(),␣
 ↪obj_id=out_obj_id)
```



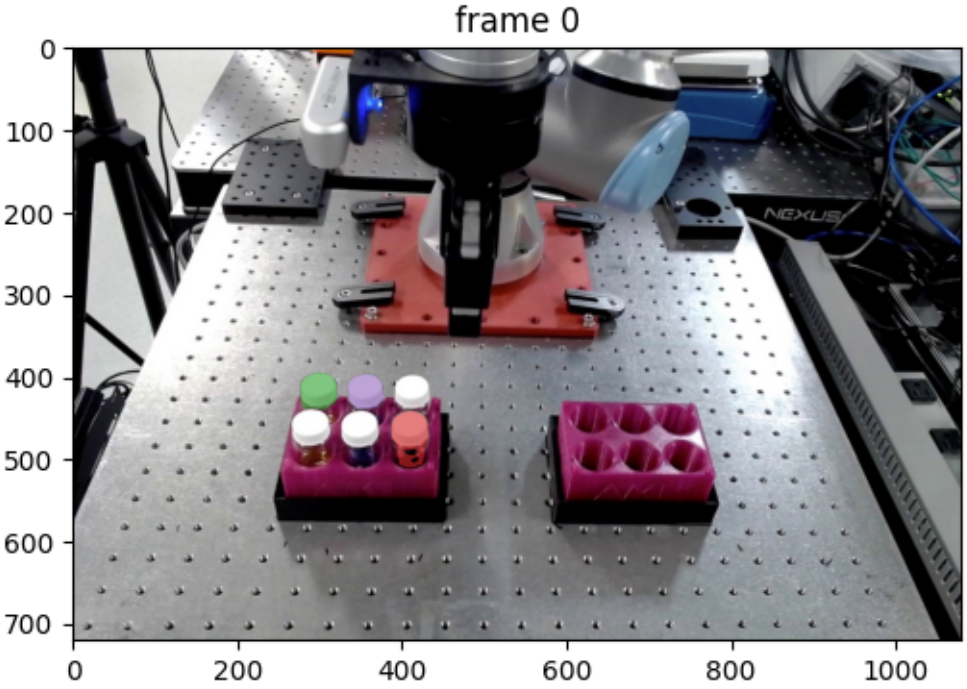
This time the model predicts the mask of the shirt we want to track in just one click. Nice!

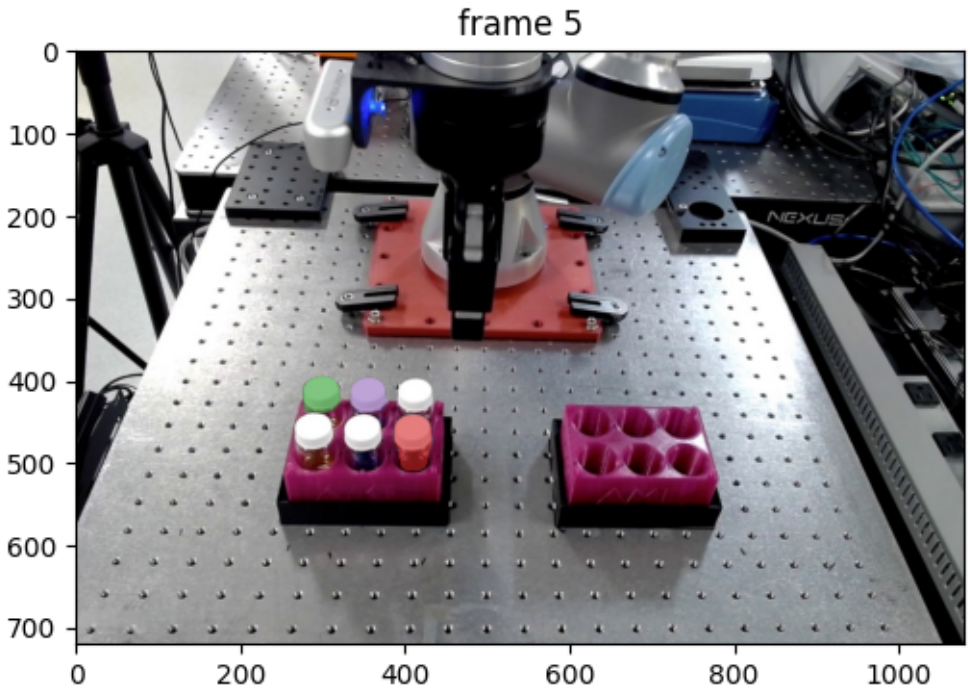
Step 2: Propagate the prompts to get masklets across the video Now, we propagate the prompts for both objects to get their masklets throughout the video.

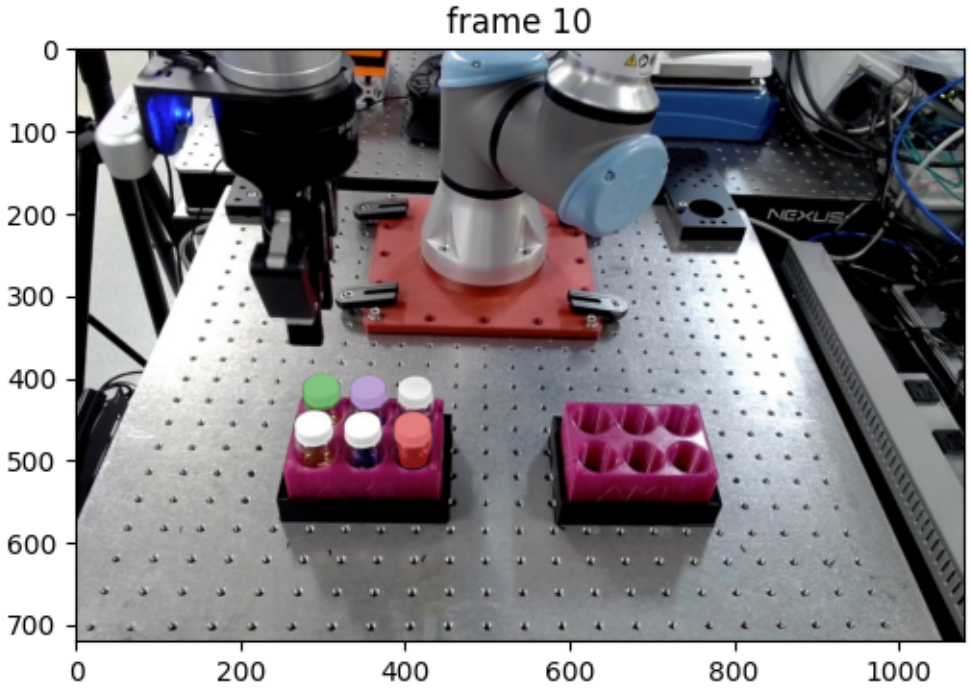
Note: when there are multiple objects, the `propagate_in_video` API will return a list of masks for each object.

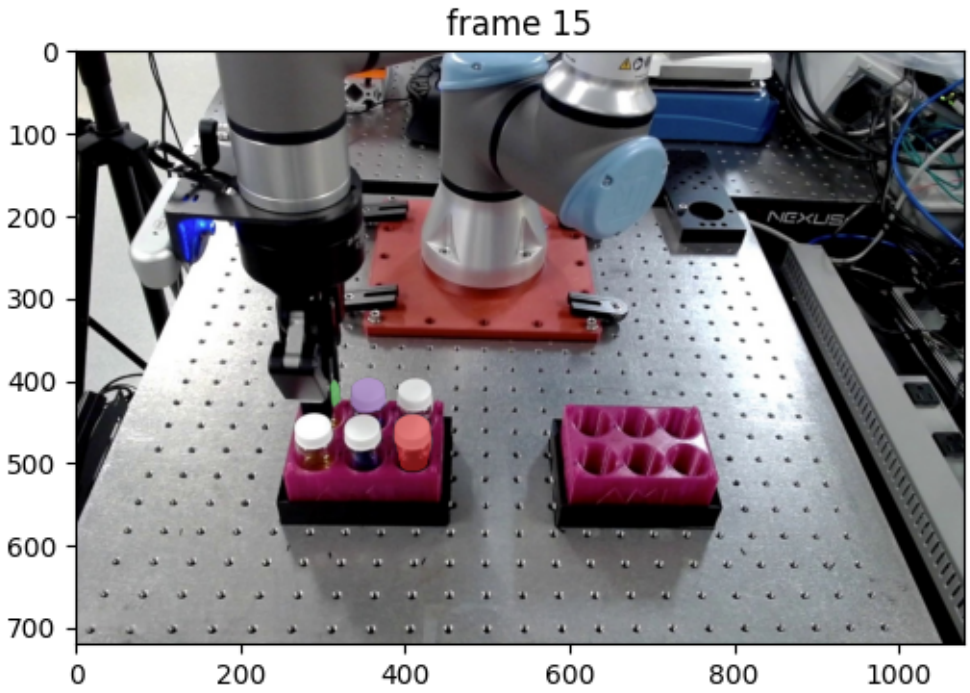
[27]:
```
# run propagation throughout the video and collect the results in a dict
video_segments = {}  # video_segments contains the per-frame segmentation
 ↪results
for out_frame_idx, out_obj_ids, out_mask_logits in predictor.
 ↪propagate_in_video(inference_state):
    video_segments[out_frame_idx] = {
        out_obj_id: (out_mask_logits[i] > 0.0).cpu().numpy()
        for i, out_obj_id in enumerate(out_obj_ids)
    }

# render the segmentation results every few frames
vis_frame_stride = 5
plt.close("all")
for out_frame_idx in range(0, len(frame_names), vis_frame_stride):
    plt.figure(figsize=(6, 4))
    plt.title(f"frame {out_frame_idx}")
    plt.imshow(Image.open(os.path.join(video_dir, frame_names[out_frame_idx])))
    for out_obj_id, out_mask in video_segments[out_frame_idx].items():
        show_mask(out_mask, plt.gca(), obj_id=out_obj_id)
```

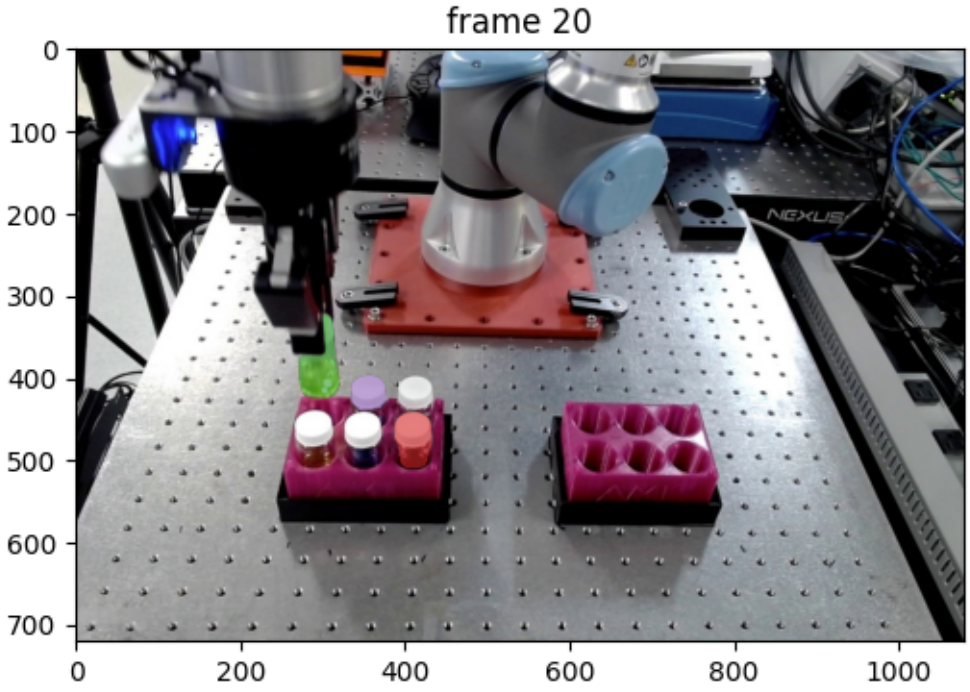
```
propagate in video: 100%|      | 80/80 [00:05<00:00, 15.64it/s]
```

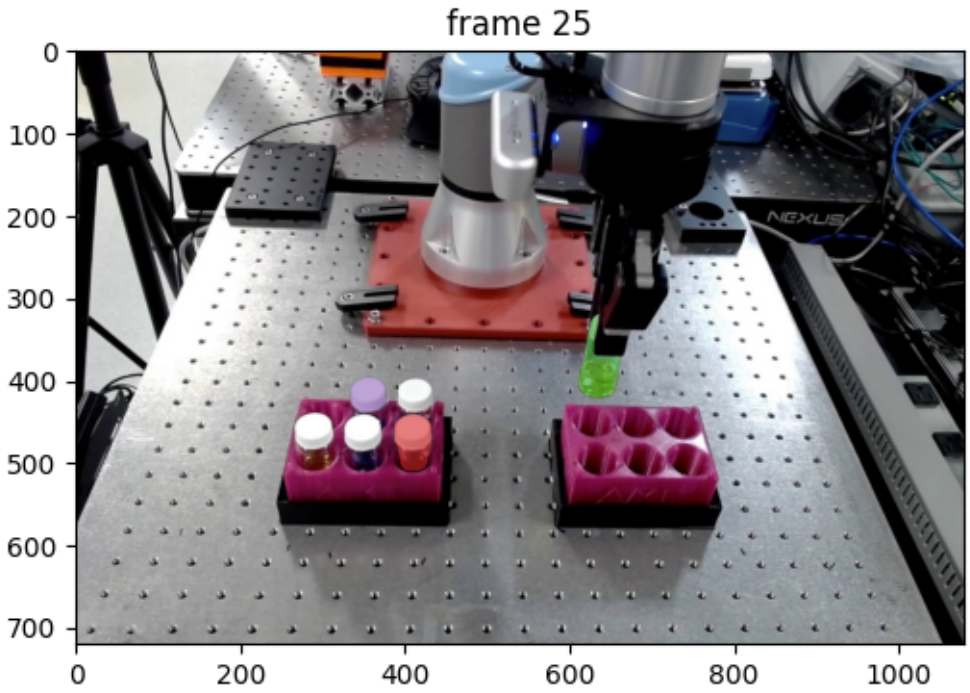


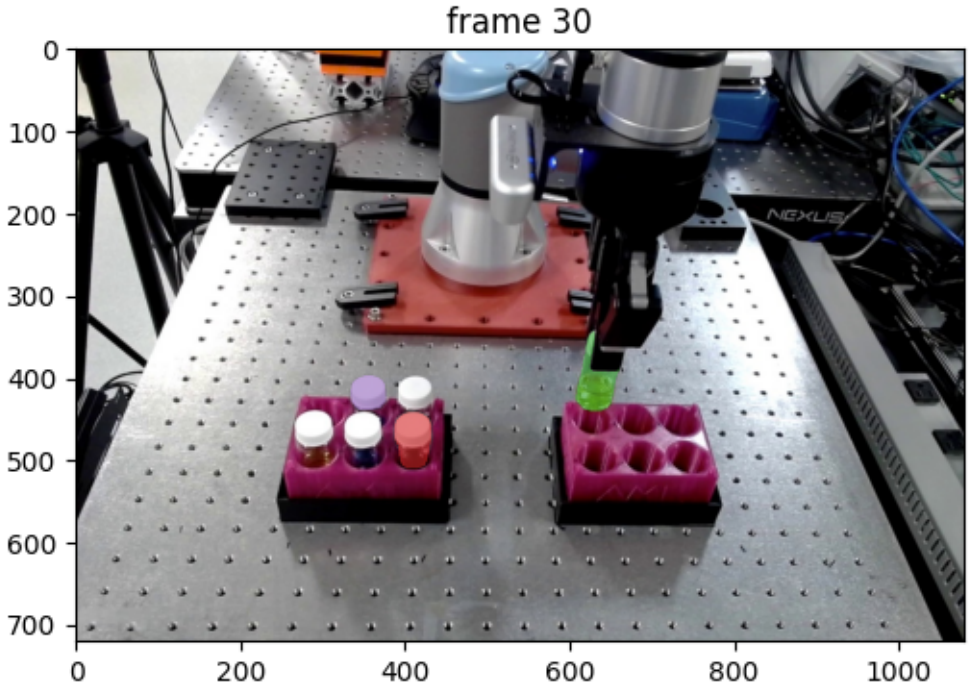

frame 5
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

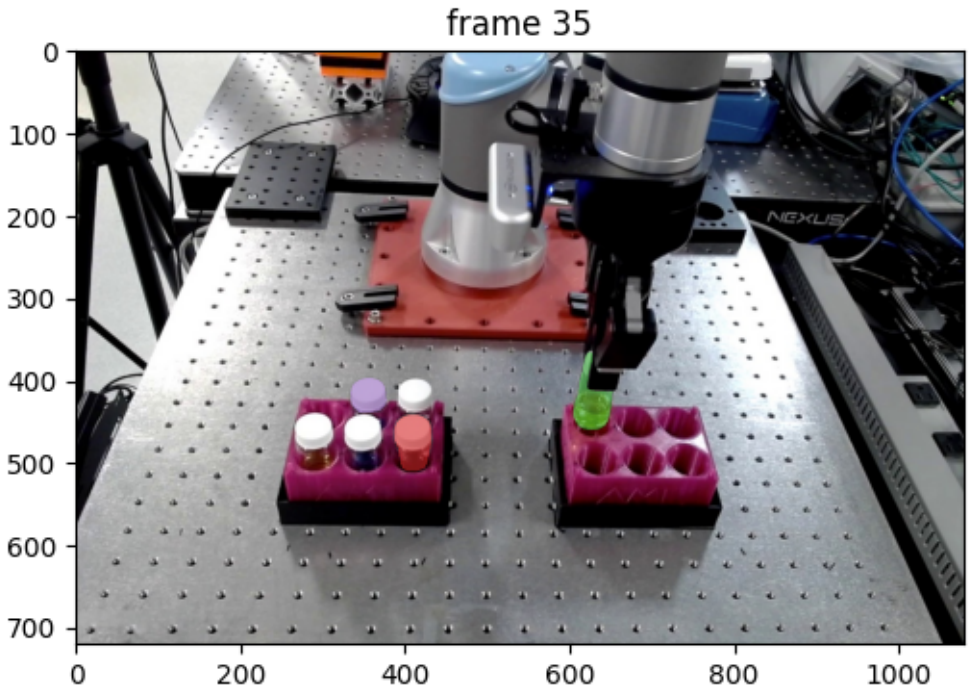

frame 10
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

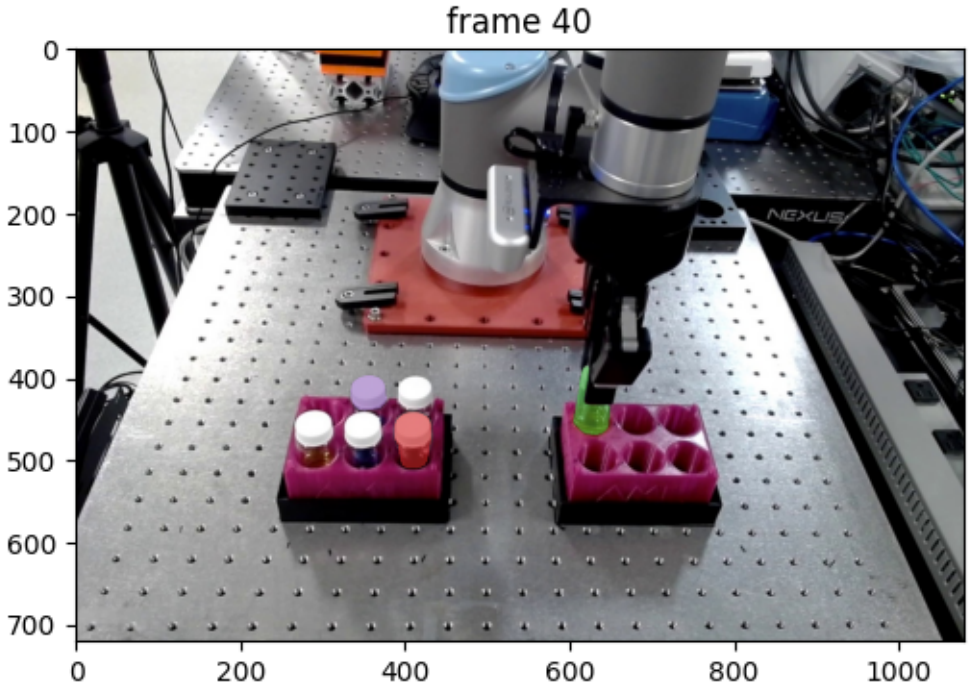

frame 15
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

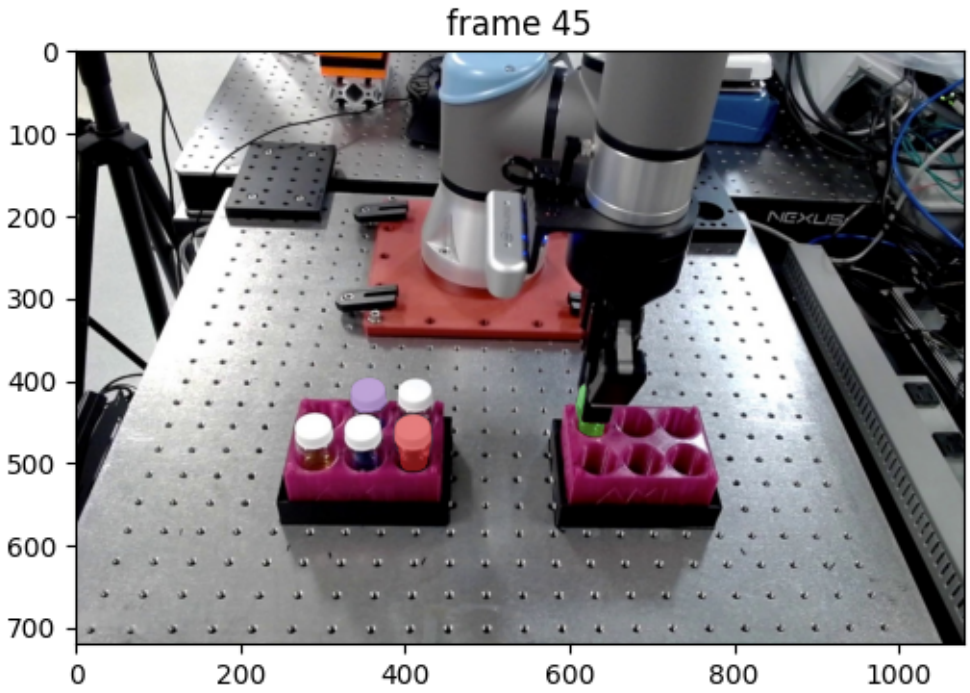

frame 20
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

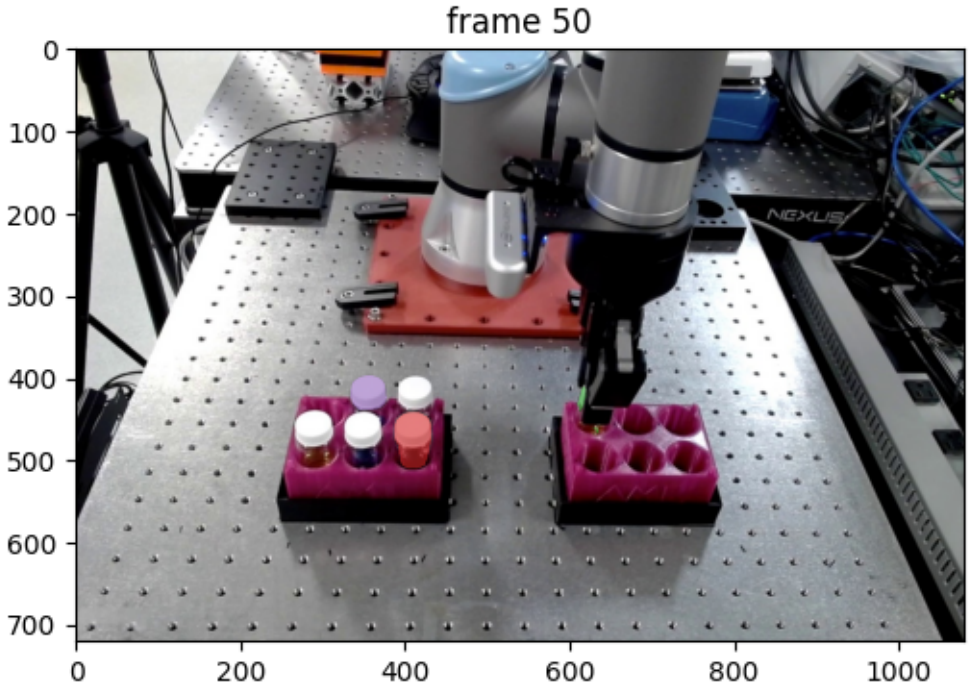

frame 25
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

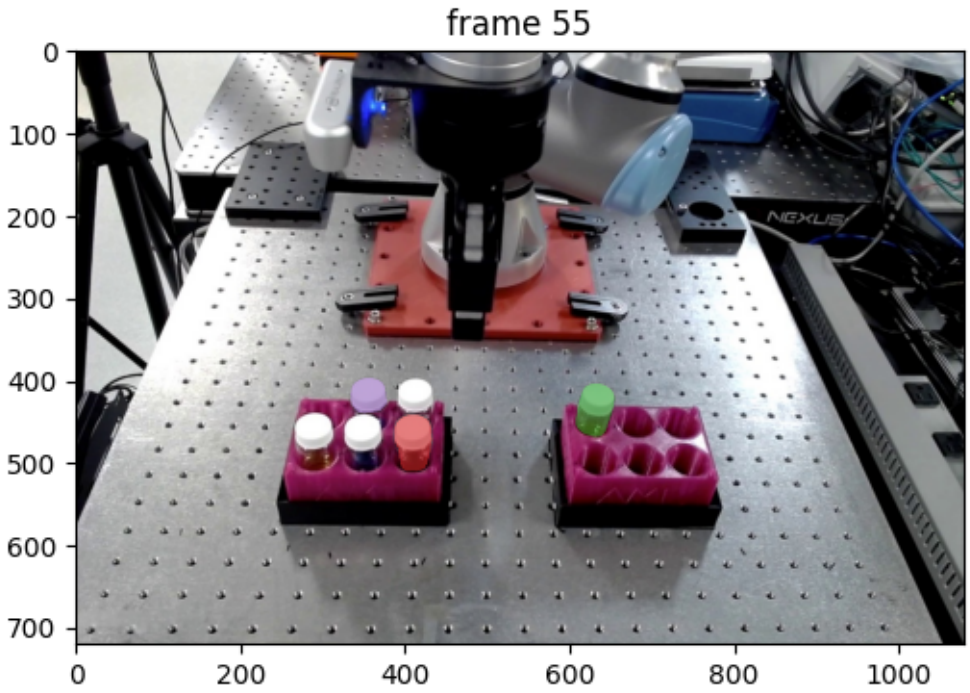

frame 30
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

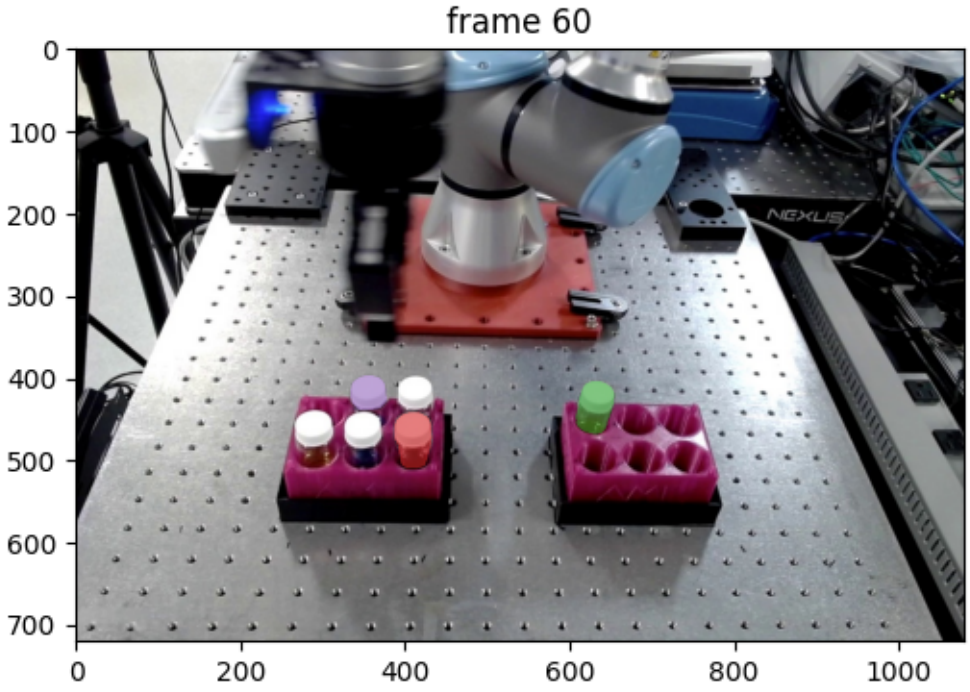

frame 35
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

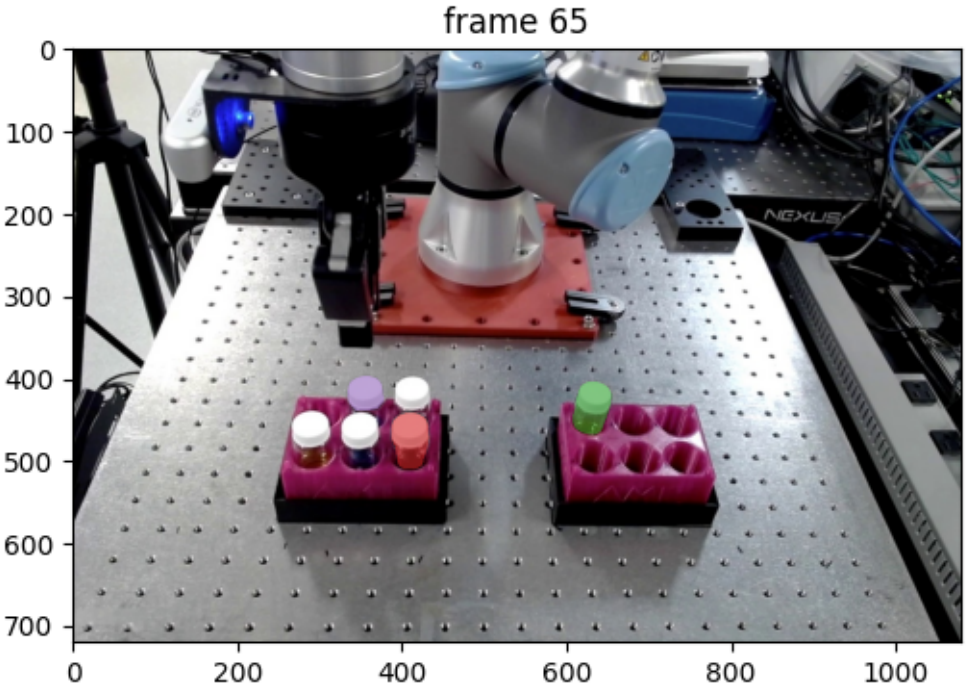

frame 40
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

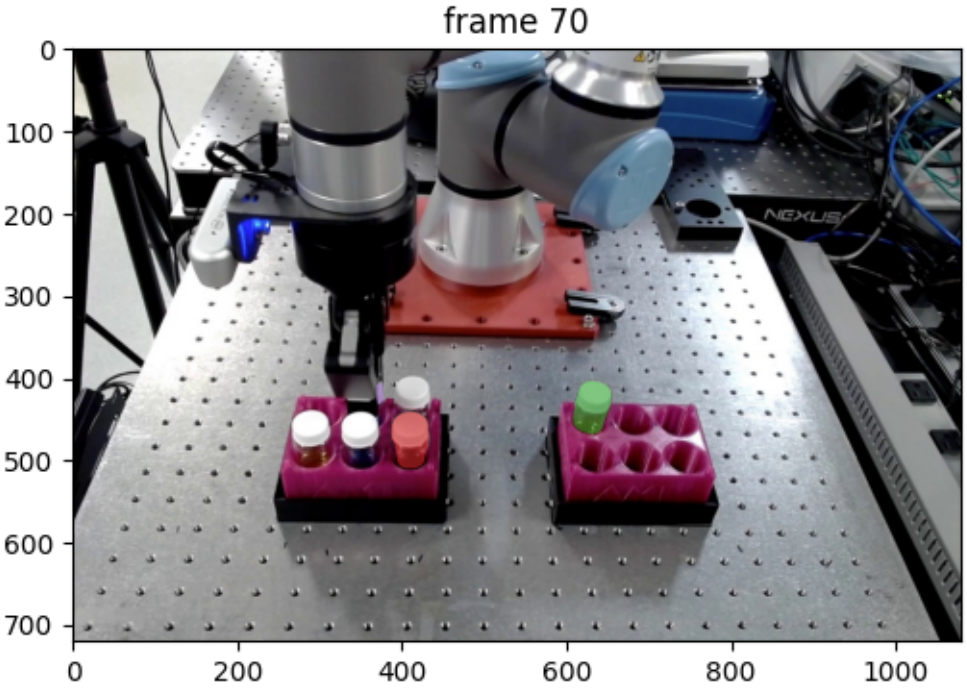

frame 45
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

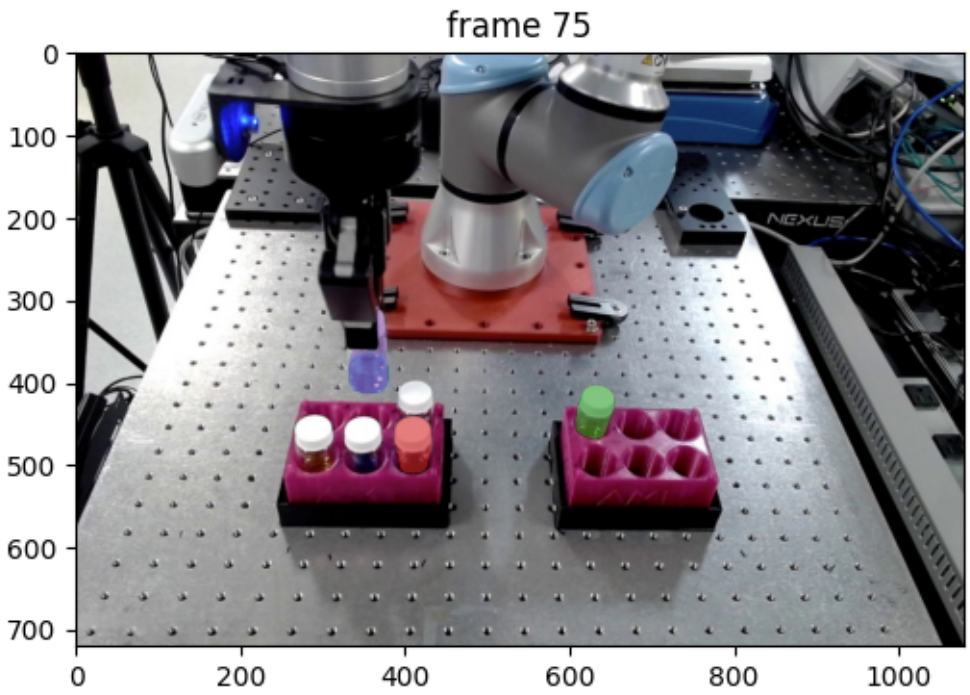

frame 50
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 55
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 60
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 65
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


frame 70
0
100
200
300
400
500
600
700
0
200
400
600
800
1000



Looks like both children's shirts are well segmented in this video.

Now you can try SAM 2 on your own videos and use cases!