RoboTAP Implementation for PiH

Samyak Tiwari

00

Denoising Point Cloud Data

Denoising Point Cloud Data

Ideas:

- SuperQuadric Fitting
- Edge detection
- Unsupervised classification of stray points

Issues:

- Diffusion EDF may have model-inherent limitations — won't be solved even with perfect point cloud data
- Too much point-cloud manipulation can cause offsets in pose estimation.

01

Playing Around w/TAPIR

Summary of TAPIR

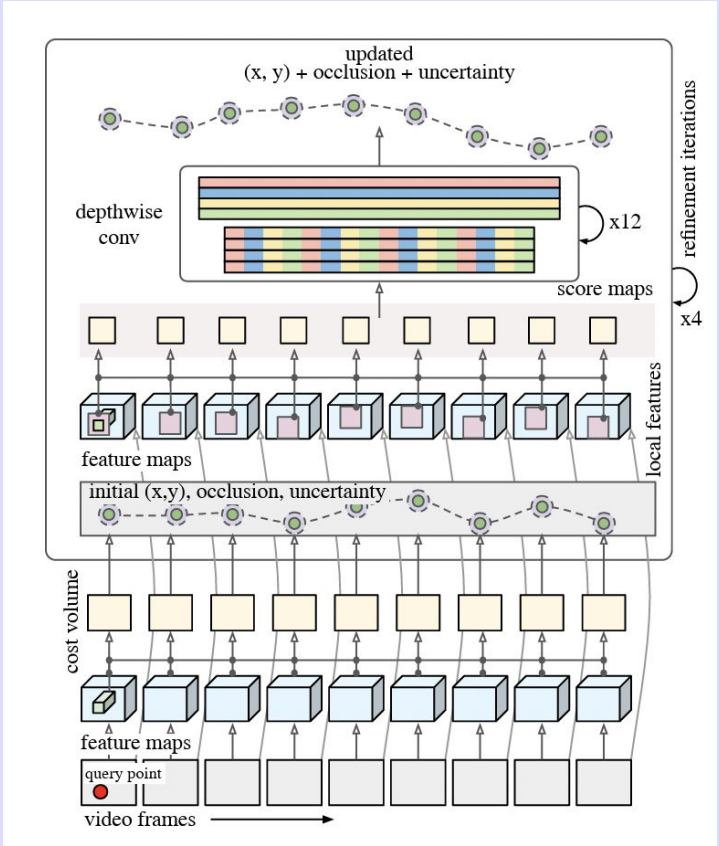
- Coarse-to-fine method ensures efficiency
- Iterative refinement ensures accuracy

Track Initialization:

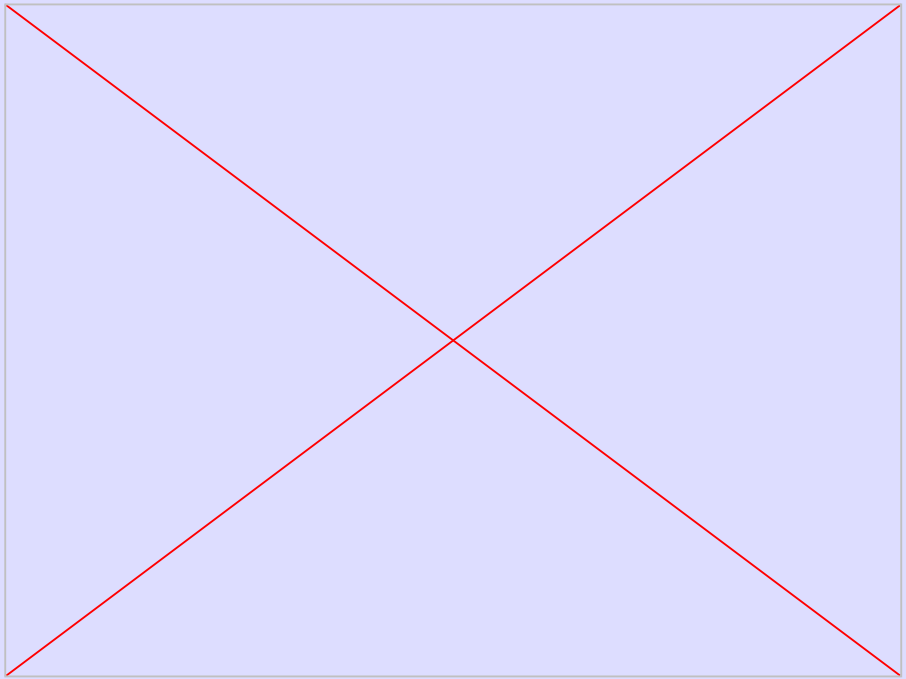
- Focuses on “global” features
- Features extracted via bilinear interpolation
- “Cost Volume” obtained from dot products of F_q with features on course map
- ConvNet produces heat map from cost volume for initial estimates of query positions and occlusion probabilities!

Iterative Refinement

- Looks at more local features (7x7 range) along track to update position and occlusion estimates.
- Integrates information across time (NOT causal).

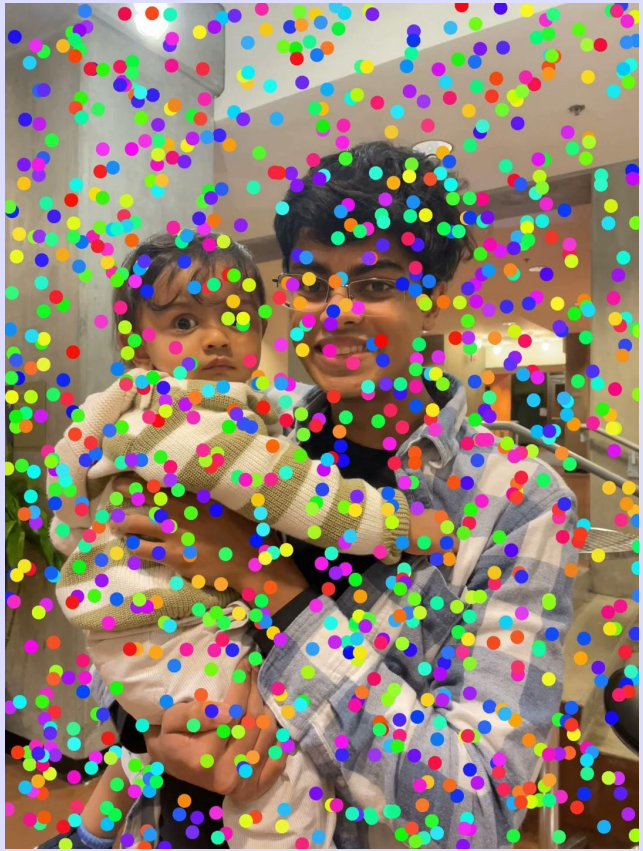


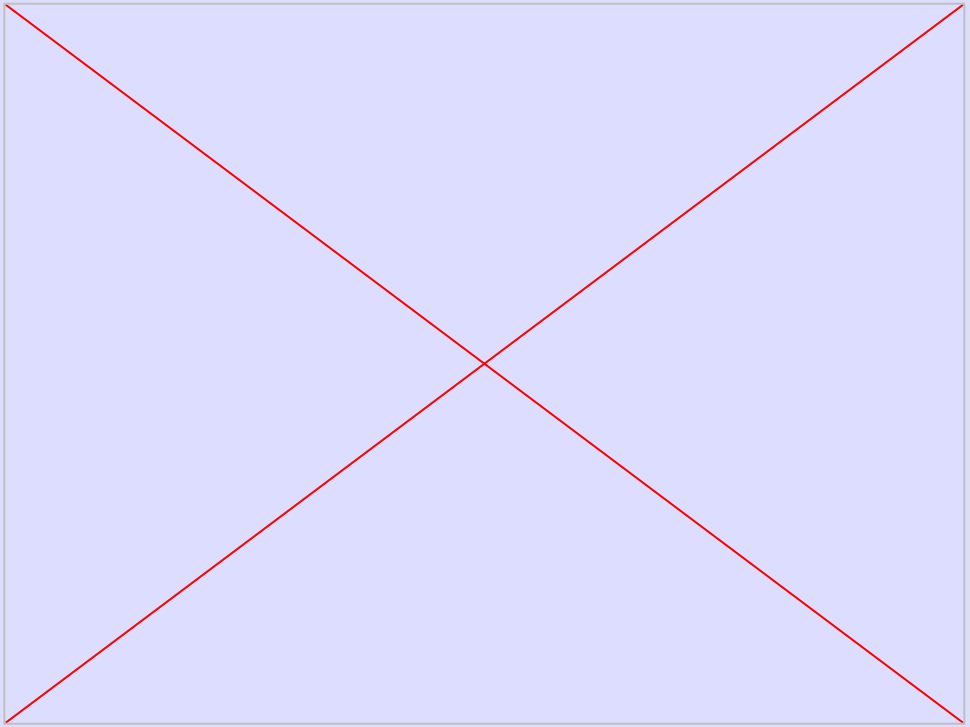
TAPIR Keypoint Tracking Demos



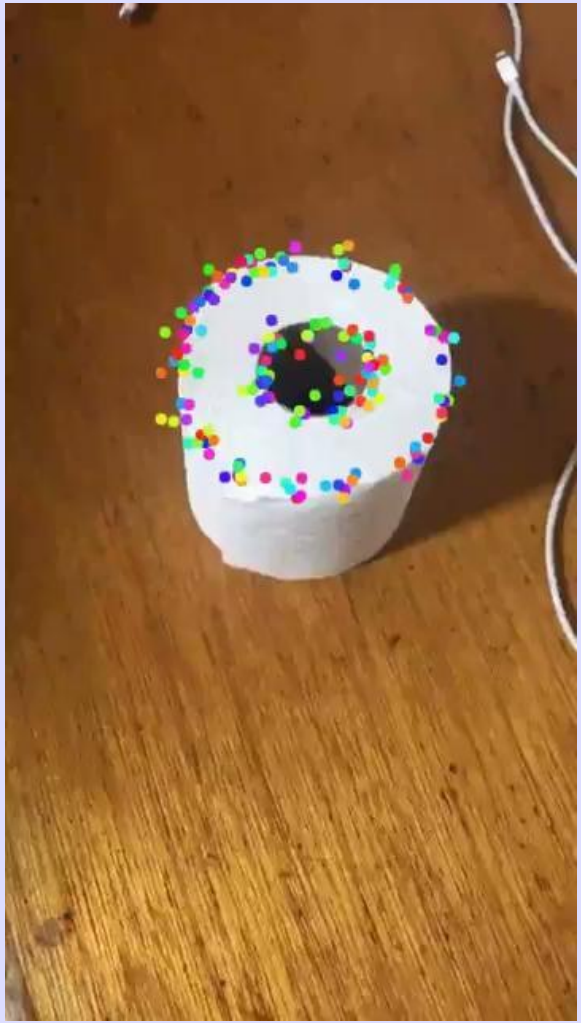
TAPIR Keypoint Tracking Demos (Horse)

TAPIR Keypoint Tracking Demos (Atharva)

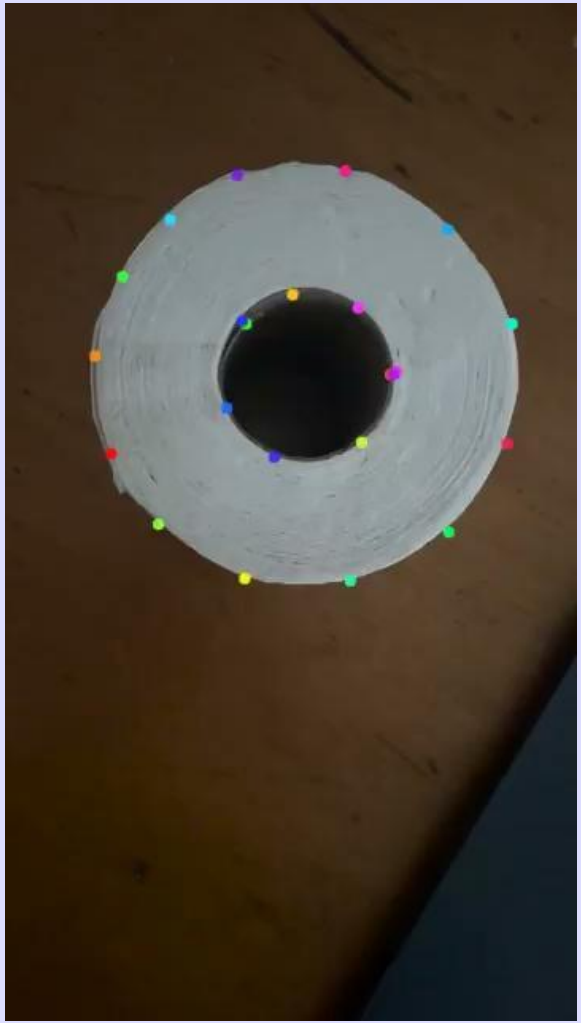




Robustness to Lighting



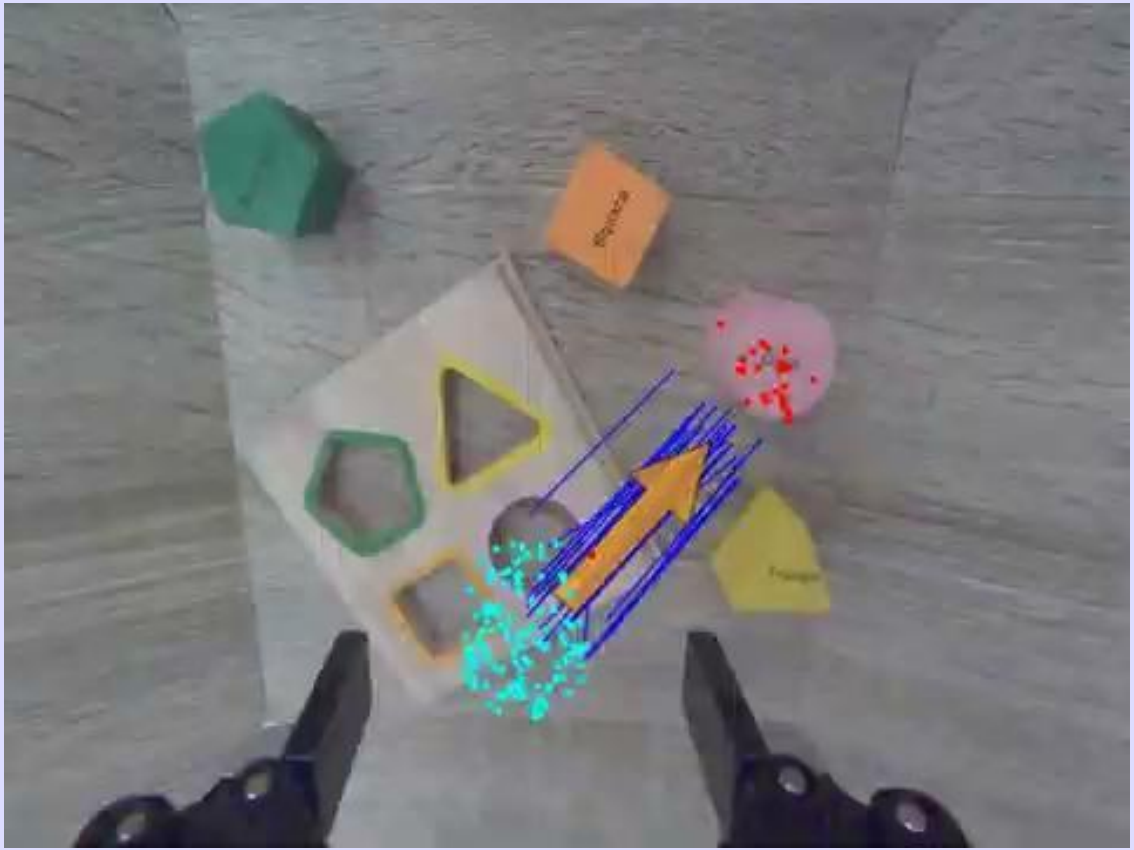
Robustness to Pose/Shape Change (3D)



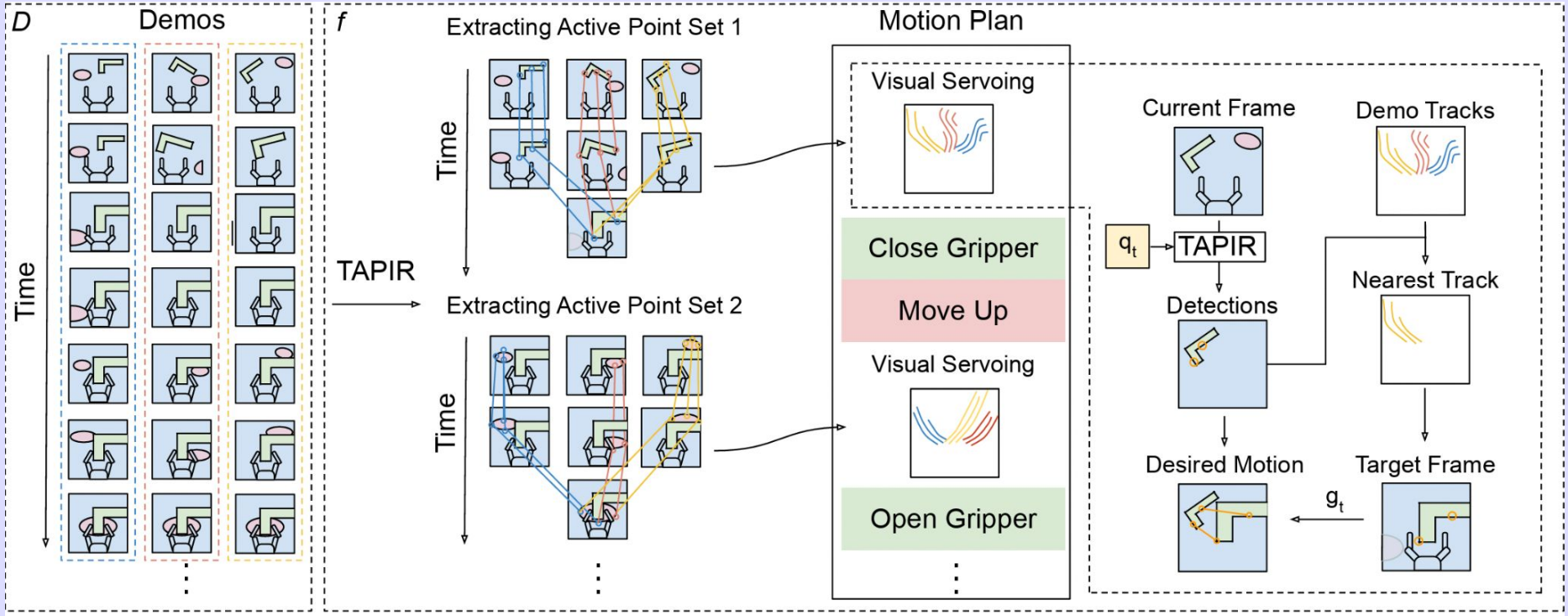
02

ROBOTap (+ 1st Attempt Implementation)

RoboTAP



Full Pipeline



Overview of ROBOTap

$\pi(s_t, D) = \pi(TAP(s_t, q_t), g_t) = \pi(p_t, o_t, g_t)$ ——> Control policy is π

$g_t, q_t = f(s_t, D)$ ——> f is an active point selection algorithm (novel component)

$TAP(s_t, q_t) = p_t, o_t$ —-> TAP is TAPIR

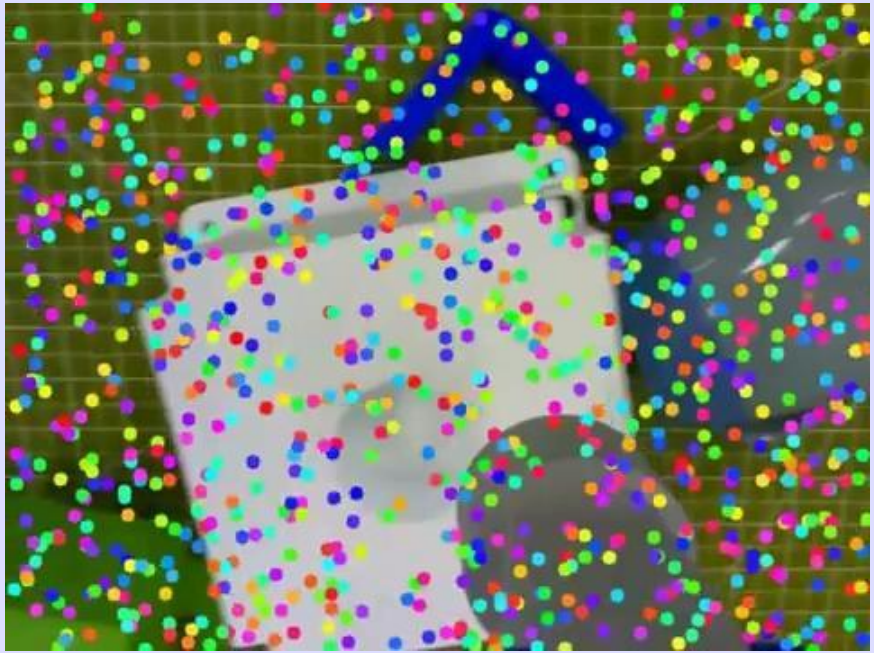
s_t: state image
D: demonstrations
q_t: queries obtained from demos (point features)

p_t: current positions of relevant points
g_t: target positions of relevant points
o_t: point occlusion probabilities

Clustering: $f(s_t, D)$

Combine the overlap between the low cross-demo variance (taken in final frame) and non stationary points to choose the most relevant motion clusters!

Trying it on Our Setup!



TAPIR

Clustering

Brief Reflection on Clustering: Why didn't it work well on our setup?

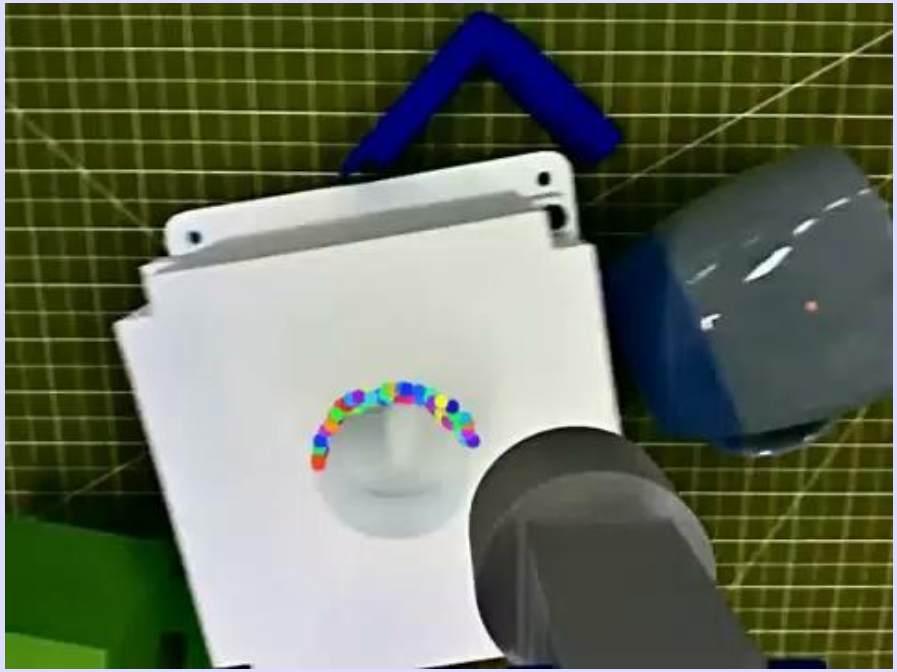
- Hole platform is featureless, preventing high quality detections.
- Detection of surrounding objects is mediocre, but unreliable (they won't necessarily be present).

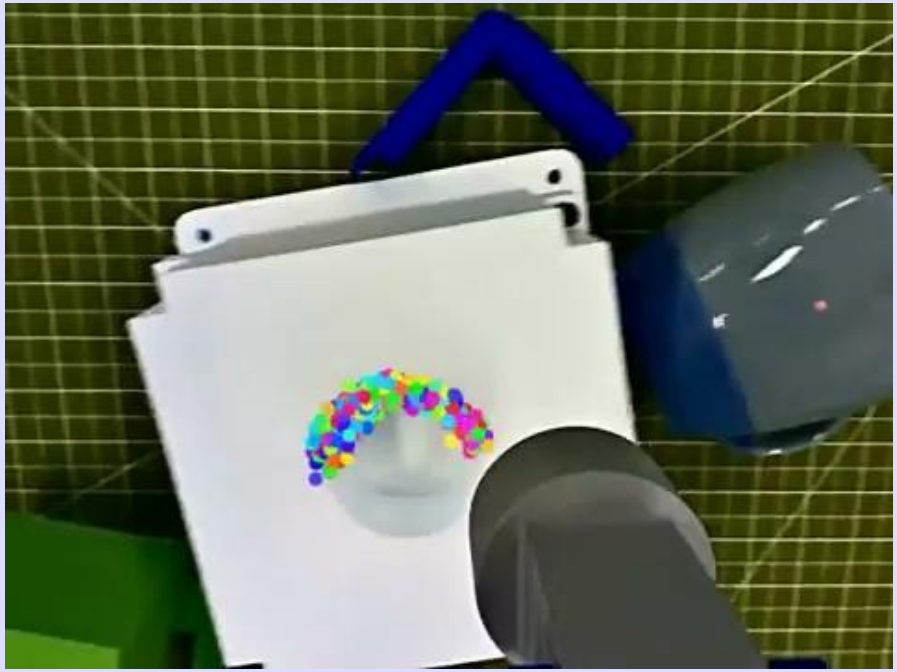
Solutions:

- Rather than using ROBOTap's clustering algorithm to select points, manually select points in the most task-relevant noising and randomly select surrounding points.
- Sharpen the image to add features!

Manually Selecting Query Points

Sharpening Filter, Im + Laplace(Im)





Overview

Pros

- Few demonstrations necessary.
- No retraining needed to obtain keypoints, minimal per-task engineering
- Closed-Loop
- Highly interpretable
- Robust to clutter and target pose randomization

Cons

- Lacks robustness to gripper pose
- Struggles with visual context-limited tasks
    - Fully occluded task-relevant object
    - Symmetries in goal

(Updated) ROBOTap Weaknesses to Consider

- Gripper pose
  - The peg may be in an orientation at which it is unable to be picked straight-on.
    - Manually selecting points for ROBOTap in the picking portion (and in general) would be problematic for cases in which selected queries aren't guaranteed to be visible!

1. We need to use ROBOTap in every temporal step in order to ensure consistency in starting frame
2. We have to either use clustering to select queries, or develop a pipeline that guarantees the visibility of manually selected queries.

- Struggles with visual context-limited tasks
  - Fully occluded task-relevant object (eg peg covers the hole)
  - Featureless goal
  - Symmetries in goal prevent the use of multiple demonstrations

1. Develop demonstrations that do not occlude points (adjustments to camera or demo trajectory).
2. Add our own features! Eg. scribble on the surface of the hole.

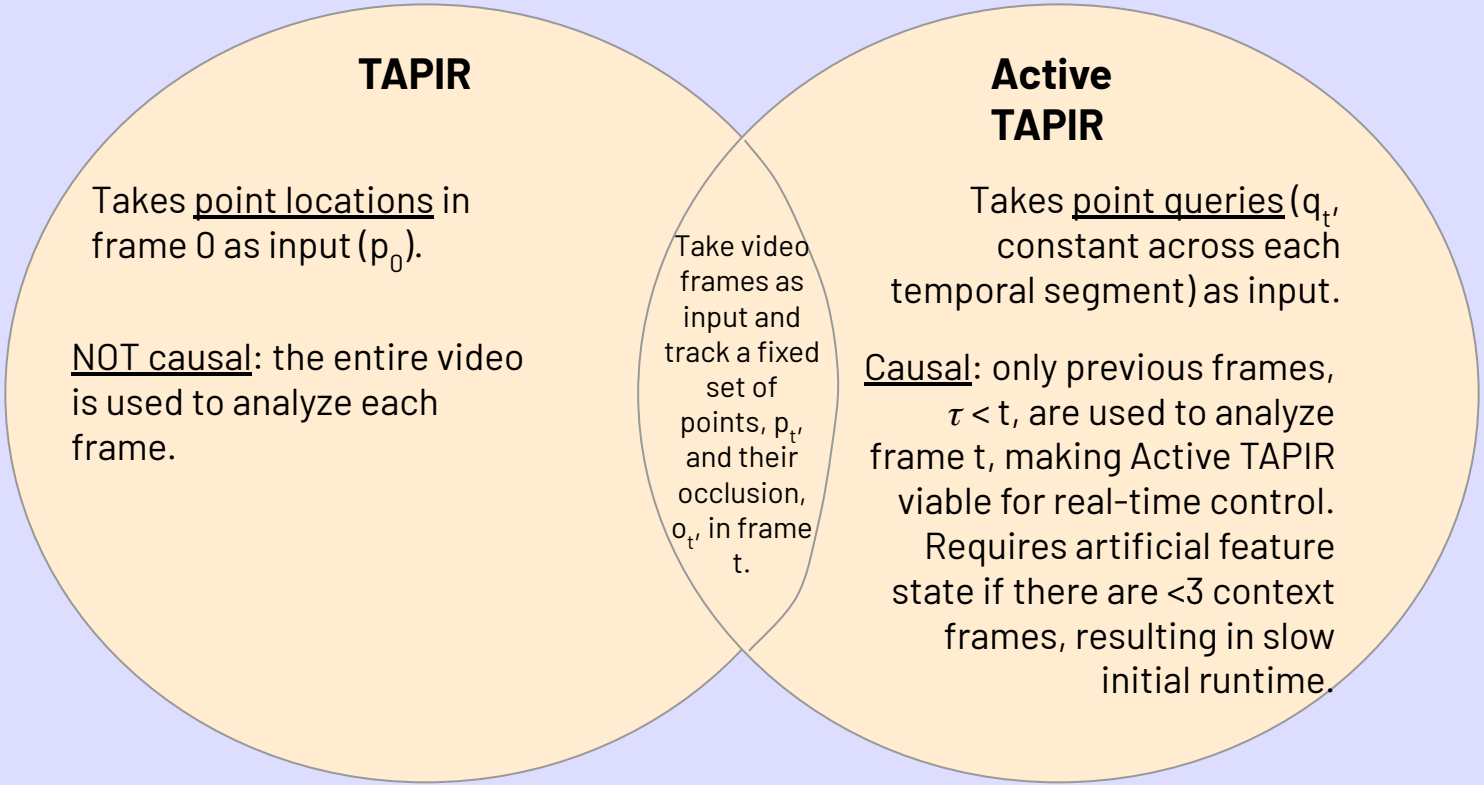
Next Steps (2/24)

- Identifying features and extracting descriptors
  - SIFT / Multi-Scale Oriented Patches
  - How to “average” descriptors, q_t, across multiple demos?
- Find target, g_t
- Active TAPIR to maintain updated features set
  - Temporal convnet -> causal convnet
- Control
  - Choose points with high certainty per TAPIR
  - Compute camera-frame transform
  - Convert to end-effector frame with image Jacobian

02½

Additional Context

TAPIR + Active TAPIR



ROBOTap

Extracting a Task-Relevant Point Set from Demos

- def track_many_points(demos):
  - Combines demos of a given temporal segment and randomly samples points from throughout the video. Extracts points (p_t), features (q_t), and occlusion probabilities (o_t) for each point!
- Form clusters of the sampled points, filtering points that have poor visibility / don't fit well into any cluster.
- Select the most task-relevant cluster and clean it for our final point set!

Robot Control

- Query final point set in current camera frame using Active Tapir.
- Select a demonstration from which to obtain target point set (g_t).
- Use camera-frame Jacobian to translate T: p_t -> g_t to an SE(3) end-effector transformation.

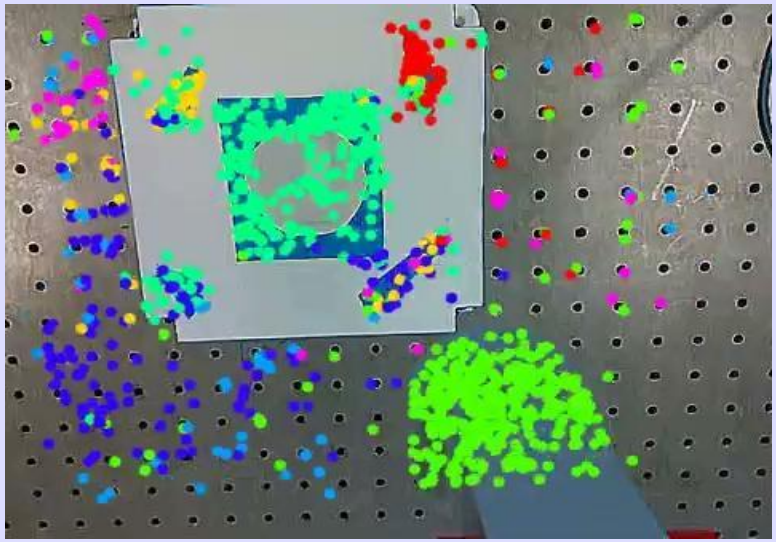
03

Extracting the Query Points (2nd Attempt, Success!)

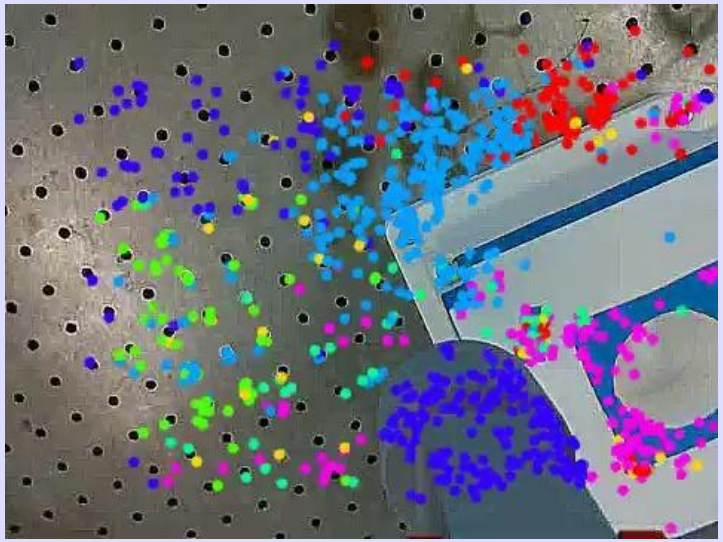
Demos

Tunable Parameters

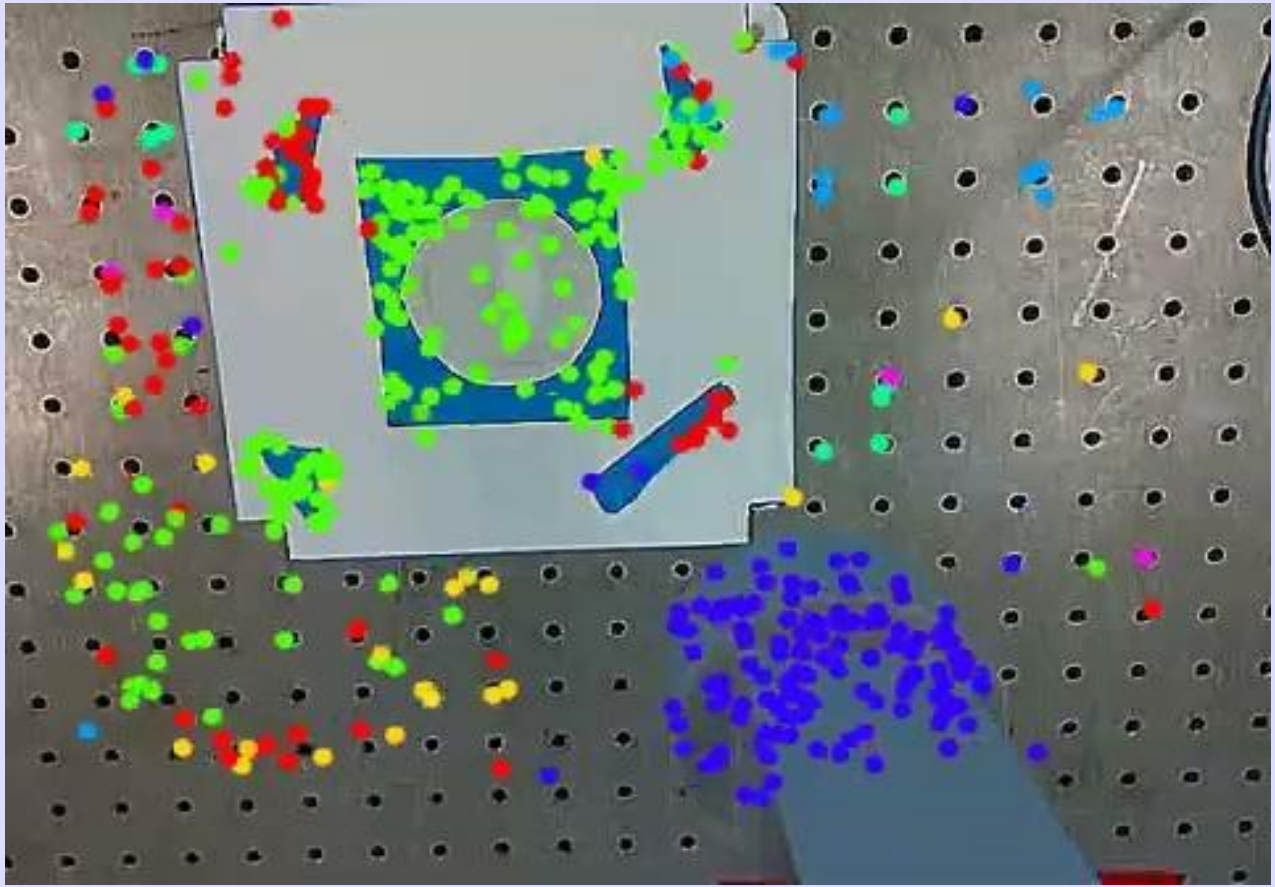
- max_clusters (before merges) and final_num_clusters — 12 and 7.
- Points sampled from every other frame (frame_stride = 2) determines point density.
    - Combining the videos ensures the same points are sampled across all demos.
- ~2 hours to track points on T4 colab GPU, ~20 minutes to learn clusters.



Demo 0



Demo 1



Sparse + Primary
Cluster Leakage

Finding The Right Cluster

ROBOTap uses cross-demo variance in the final frame and point activity to vote on a cluster. There were a few issues... Here's what I did instead!

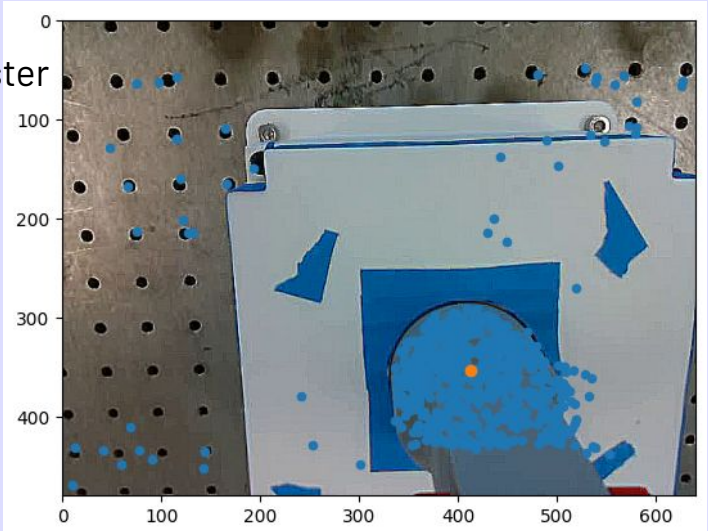
- Identify most stationary point-set.
  - Measure final **frame position - initial frame position**. Determine fraction of points above 0.3 * 90th quantile:

array([0.958, 0.956, 0.559, 0.979, 0.0582, 0.772, 0.789])

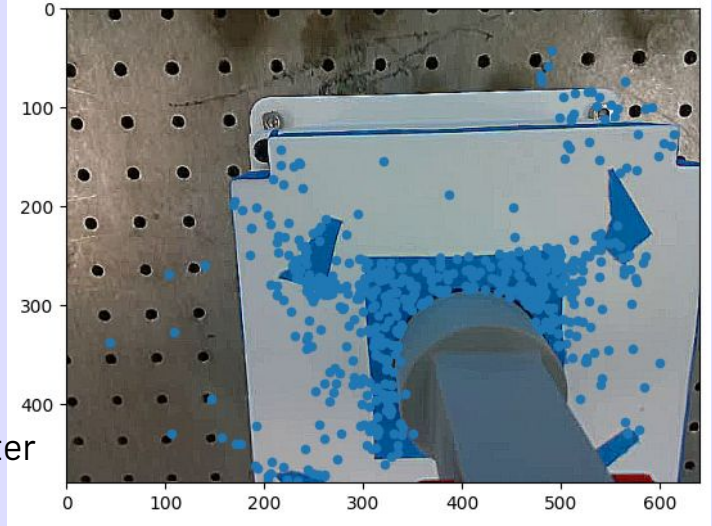
- Evaluate closest active cluster!

array([286, 273, 364, 145, inf, 291, 328])

Stationary Cluster

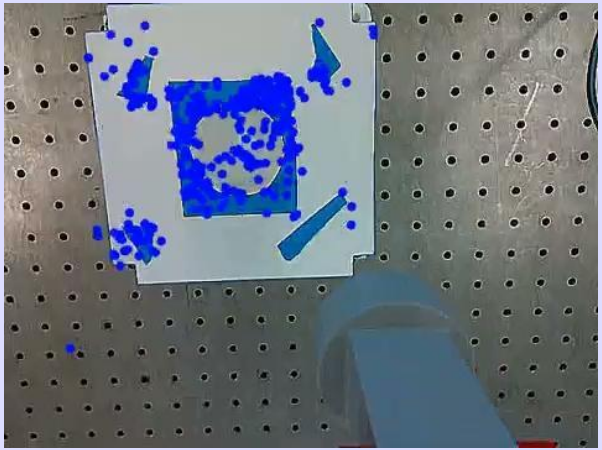


Active Cluster

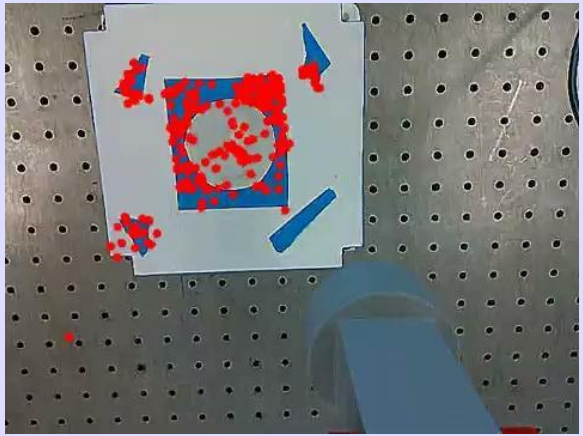


Cleaning The Cluster

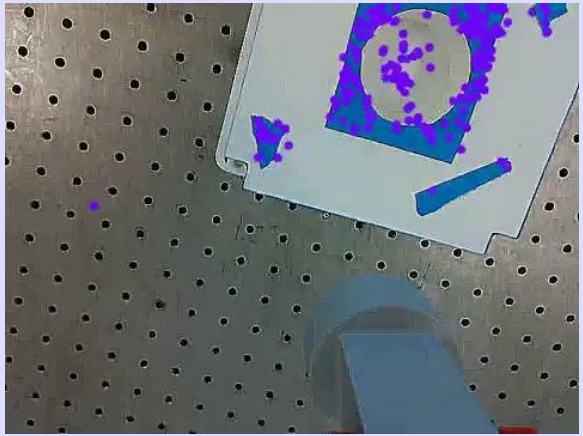
- Saliency Metric (0.4, 0.25):
  - Keep visibility of points in first half of video above 0.4
    - Doing this on the whole video resulted in **bottom-points scarcity**.
  - Remove 25% points that most frequently “turn off”
- Remove 10% highest K-Distances (k=8)



Chosen Cluster

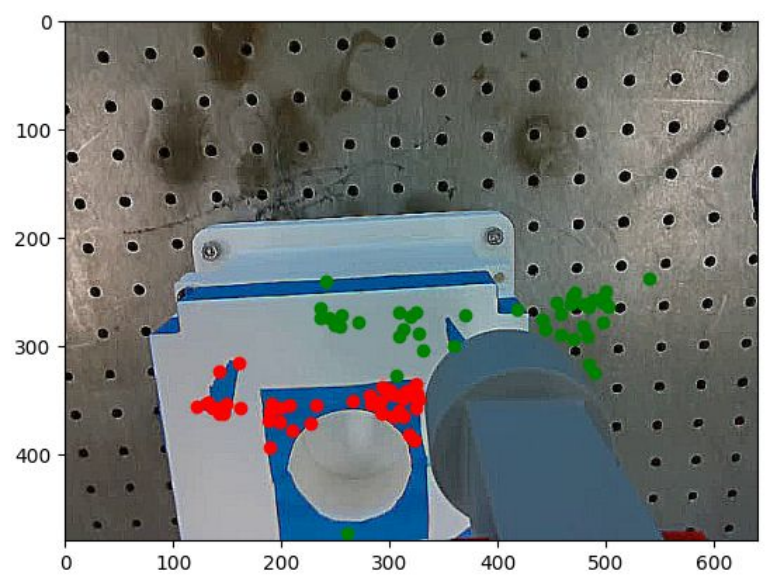


Post-Saliency Filter

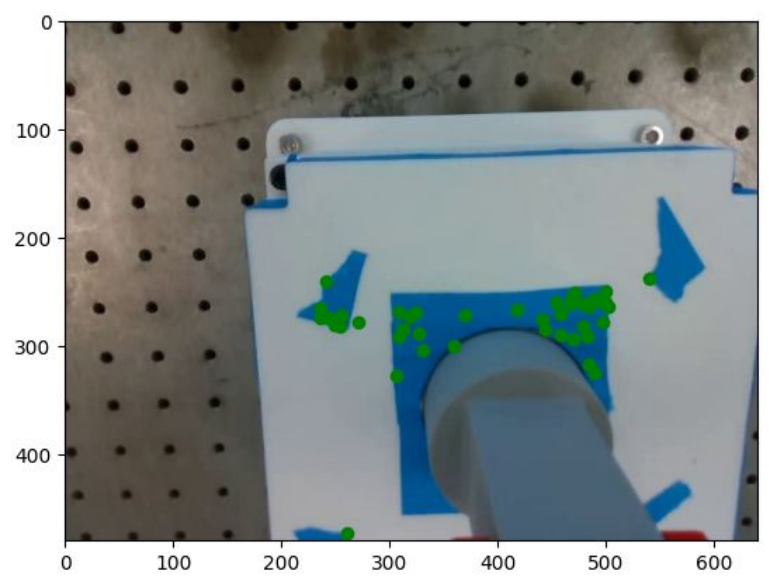


Post-KD Filter

Querying the Points on an UNSEEN Demo



First-Frame Query



Final Frame of Demonstration 0
(Closest to final frame of demo 6)

Runtime Analysis

Passing each frame as is

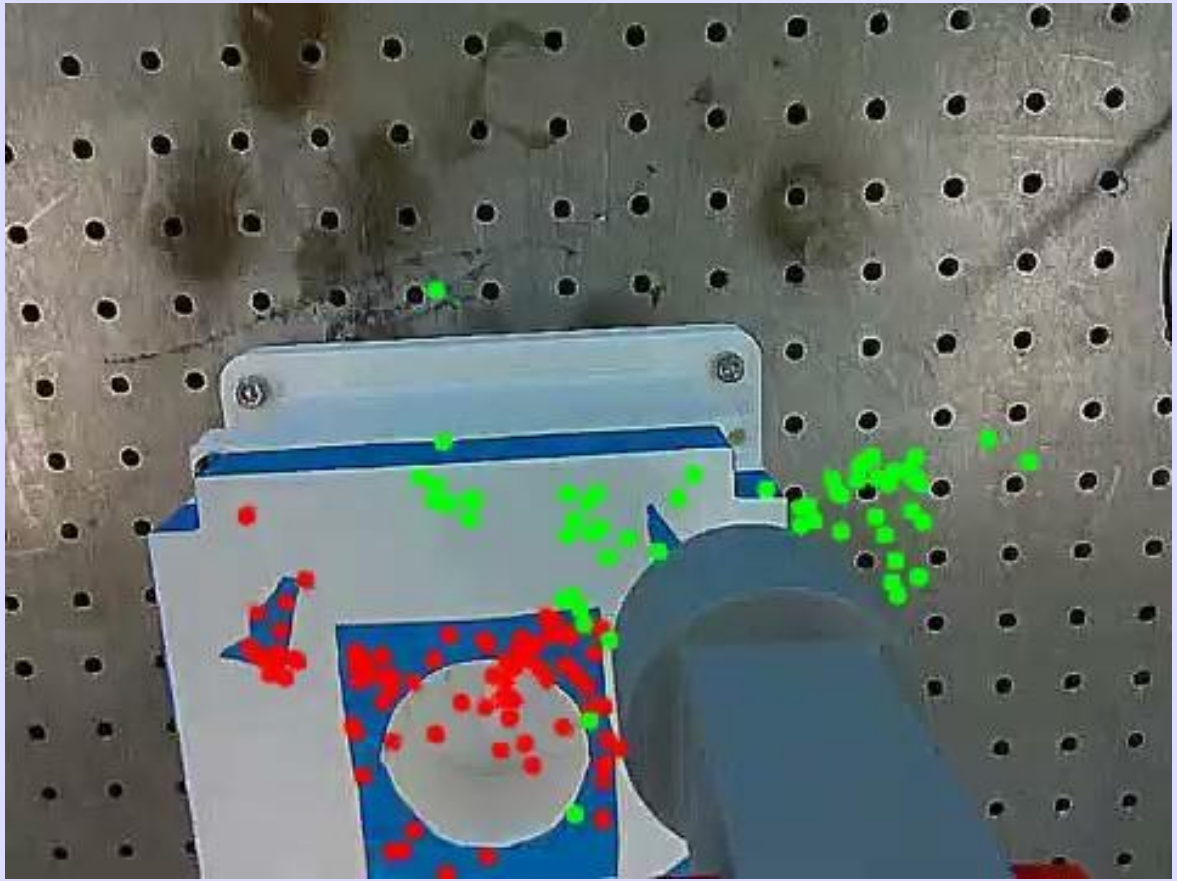
- First two frames: 50s each
- Last 347 frames: 10s

11.6 fps!

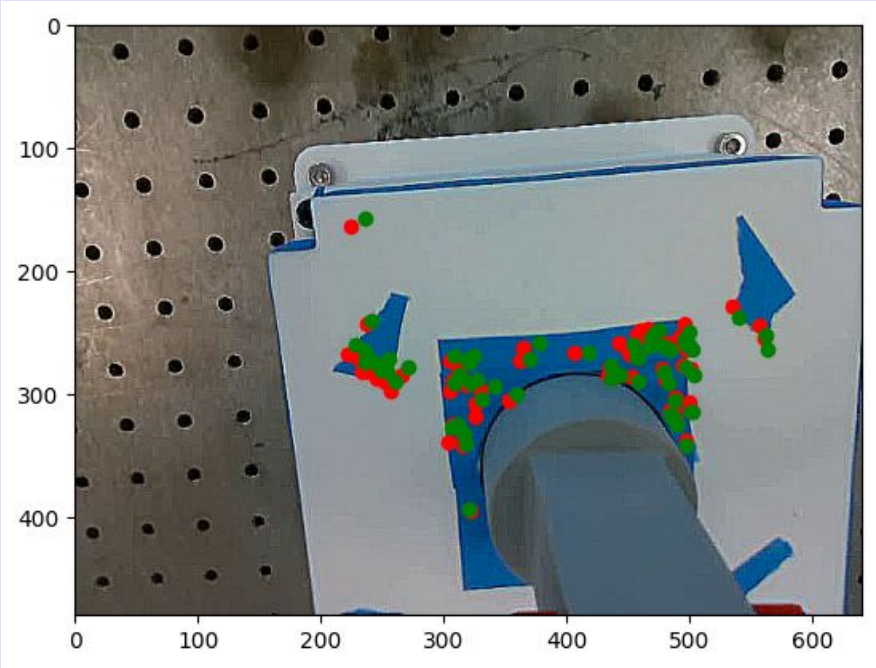
Reducing frame size to 256x256 (TAPIR was trained on this)

- First two frames: 18s each
- Last 347 frames: 8s

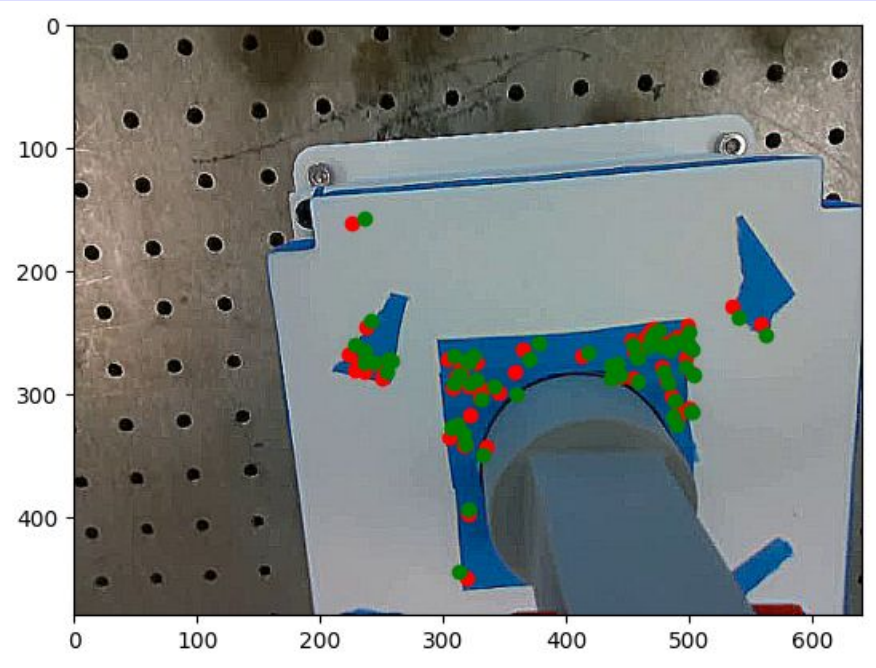
43.7 fps! (+ reducing frame size processing time)



Comparing Final Point States of TAPIR on Original Image and Compressed Image



Original Size



256x256

The green points shown are the demo0 targets

Summary of Adjustments to DeepMind's Implementation

- Supplementing with Diffusion EDF.
  - Ensures similar starting position between demos and live control (queries can be subject to scaling errors, though they're robust to lighting and angle changes).
- Cluster selection heuristic: using the cluster closest to the stationary.
  - BIGGEST change. Better for dynamic context precision task and robust to camera-view limitations.
- Adapting the cluster-cleaning saliency metric to occlusions later in the demos.
  - First half of video for saliency.
  - Full video for "turning off"

Implementing the Controller

To ensure robustness:

- Use 30 most confidently detected points in real time (increase this number if we query more than 128 pts)

Issues w/Their Implementation:
They do **1) demonstration following** for the initial phases and **2) mean target following** for final alignment.

1) In the initial phase, they use the demonstration with the minimum **Euclidean distance** between detected camera points and the points of the first frame of the demos. Then, they use the ONE demonstration to do **frame-by-frame trajectory** matching: servo when above 30th percentile error threshold, move to next frame when below. When reaching the final frame, move to phase 2!
    a) BUT this may be backwards in some cases (demos started farther out than real life will). We can either **check later demo frames** for initial position or **take new demos**.
2) Take the mean of the final frame of all demos and use it as our final target!
    a) This doesn't work for us unless we handle **rotational symmetry**! Our target pose varies across the demos.

THANK YOU