Hochschule Konstanz
Fakultät Informatik

Deformable Object Tracking

HTWG Konstanz: Teamprojekt WS 24/25 und SoSe 25 in Kooperation mit Subsequent

Arian Braun, Leonie Fauser, Jonas Kaupp, Lukas Reinke, Yoshua Schlenker

Konstanz, 31. August 2025

Contents

1. Introduction **3**

2. Methodology **3**
2.1. Previous papers . . . 3
2.2 Comparison of existing tracking methods . . . 5
2.2.1 TAPIR . . . 5
2.2.2 CoTracker . . . 7
2.2.3 SpatialTracker . . . 8
2.3 Adaption and implementation . . . 9
2.3.1 Sliding window approach . . . 9
2.3.2 Depth Estimation . . . 11
2.3.3 Segmentation . . . 13
2.4 Data generation . . . 16
2.4.1 Ground truth . . . 16

3. Evaluation **17**
3.1. Runtime evaluation . . . 18
3.1.1 Results of the Runtime evaluation . . . 18
3.1.2 Key Findings . . . 22
2D tracking efficiency . . . 22
Comparison between ToF ground truth and tracker . . . 24
Coordinate mapping (processed → ToF) . . . 24
2D tracking efficiency . . . 25
Comparison between ToF ground truth and tracker . . . 29

Conclusion **31**
Future work . . . 32

Appendix **32**
Appendix I . . . 32

1. Introduction

Deformable Object Tracking (DOT) is a challenging problem in computer vision, as it requires robustly estimating the position, shape, and motion of objects whose geometry changes over time. Unlike Rigid Object Tracking, where the spatial configuration remains constant, deformable objects can undergo non-linear transformations such as bending, stretching, sheering or twisting. These deformations significantly increase the complexity of the tracking process, especially in real-world environments where occlusions and lighting variations are present.

Accurate tracking of deformable objects has numerous applications, ranging from robotic manipulation of flexible materials and human motion analysis to biomedical imaging and augmented reality. Recent advances in machine learning, particularly Deep Neural Networks, have demonstrated strong potential for capturing the complex dynamics of non-rigid motion. However, achieving high tracking accuracy in dynamic and unconstrained scenarios remains an open research challenge.

The goal of this project is to research and develop a system for tracking deformable objects, specifically sports equipment like resistance bands and gymnastic balls. The goal is to support physical therapists and their patients in their exercise execution with an automated, AI-driven solution.

Therefore, possible approaches are discussed in chapter 2. Disregarding most of them because they did not fit the requirements leads to one approach called SpatialTracker (Paper SpatialTracker) which we will be focusing on during the future work. Also in this chapter, the original model will be adapted, e.g, with a sliding window approach for loading longer videos and an automated segmentation for selecting the region of interest. The results and evaluation of the work will be stated in chapter 3. Two different metrics and the runtime depending on video quality and grid size of the points to track are evaluated. In chapter 4 the work is summarized and a brief outlook in future work is given.

2. Methodology

This section will provide the theoretical foundation and describe our progress throughout the two semesters of the project.

2.1. Previous papers

Five papers were investigated which dealt with tracking non-rigid objects:

1. Huang et al., Tracking Multiple Deformable Objects in Egocentric Videos (Paper 1)
2. Zhang et al., Self-supervised Learning of Implicit Shape Representation with Dense Correspondence for Deformable Objects (Paper 2)

3. Henrich et al., Registered and Segmented Deformable Object Reconstruction from a Single View Point Cloud (Paper 3)
4. Wang et al., Neural Textured Deformable Meshes for Robust Analysis-by-Synthesis (Paper 4)
5. Xiao et al., SpatialTracker: Tracking Any 2D Pixels in 3D Space (Paper 5)

The task was to analyze whether one of the papers provides a useful approach for our task. Therefore, let us first consider the goals for this project:

1. Target objects of the tracking algorithm are resistance bands, gymnastic balls and smaller soft balls.
2. The work is motivated by the need to support physical therapy patients through the monitoring and assessment of correct exercise execution.
3. Most of the time one of the above mentioned objects is present in the frame. We focus more on the quality of tracking than on several objects.
4. Runtime of the approach is initially secondary.
5. The training data is mainly in 3D.

So let us have a short look on the first four papers.

Paper 1 by Huang et al. focuses on tracking multiple deformable objects in egocentric videos where the camera is mounted on a person. The method relies on template matching with learned features to track objects. This approach is problematic for tracking objects like resistance bands because they undergo significant and often unpredictable deformations, including self-occlusions and changes in topology (e.g. twisting). This approach cannot handle such extreme changes as the movement of resistance bands and gymnastic ball is too complex.

Zhang et al.'s work (Paper 2) addresses self-supervised learning of implicit shape representations for deformable objects. The method is designed for learning a 3D representation from multiple views or a canonical pose. The primary goal is to establish a dense correspondence for a single object, not to track its position and deformation in a dynamic video sequence. Furthermore, the objects considered in this paper, e.g. animals, are typically less flexible than a resistance band, making the approach unsuitable for the extreme deformations of a resistance band or a soft ball. The method would likely be too slow and would not generalize well to the continuous, unpredictable movements of sports equipment.

Paper 3 by Henrich et al. contributes to reconstructing and segmenting deformable objects from a single-view point cloud. This method is primarily concerned with 3D reconstruction, not tracking. It takes a static snapshot (a single point cloud) and tries to reconstruct the object's shape and segment it. This is a fundamentally different problem from tracking an object's movement and deformation over a sequence of video frames. While it deals with deformable objects, its single-view, static-frame nature makes it unsuitable for the continuous, dynamic process of tracking gym equipment in motion.

Wang et al. propose neural textured deformable meshes for robust analysis-by-synthesis. This paper focuses on reconstructing and tracking deformable objects with a pre-trained template mesh. The method requires a canonical shape of the object. While it can track deformations, it's designed for objects whose topology remains relatively stable and predictable, such as a face or a piece of clothing. A resistance band's deformation is often so complex that it can change its topology, twisting and folding in ways that are difficult to model with a single pre-defined mesh template. A ball, while simpler, can also be challenging due to its smooth, textureless surface, which provides few features for the method to attach to. This approach would likely fail when the object undergoes extreme changes not captured by the initial mesh.

As a short conclusion, it can be said that the first four approaches are not useful for achieving the desired results mainly due to the extreme deformation of the objects. Therefore, all of the mentioned papers except the SpatialTracker were dismissed in our future work.

2.2 Comparison of existing tracking methods

The only option that was left is the SpatialTracker. This is the one focused on in the next steps. In the SpatialTracker's paper, the performance was compared with other tracking models. We decided to have a closer look on two of them because we considered them as relevant in the beginning. For this in the following paragraph we focus on the models listed below:

- Karaev et al., CoTracker: It is Better to Track Together (Paper CoTracker)
- Doersch et al., TAPIR: Tracking Any Point with per-frame Initialization and temporal Refinement (Paper TAPIR)
- Xiao et al., SpatialTracker: Tracking Any 2D Pixels in 3D Space (Paper SpatialTracker)

2.2.1 TAPIR

TAPIR is a Deep Neural Network designed for the task of **Tracking Any Point** (TAP). Its main goal is to accurately follow a specific point of interest throughout a 2D video sequence, even if that point is on a deformable object, becomes occluded, or changes its appearance. This is accomplished in two steps: **per-frame initialization** and **iterative refinement**. Figure 1 shows the architecture of TAPIR.

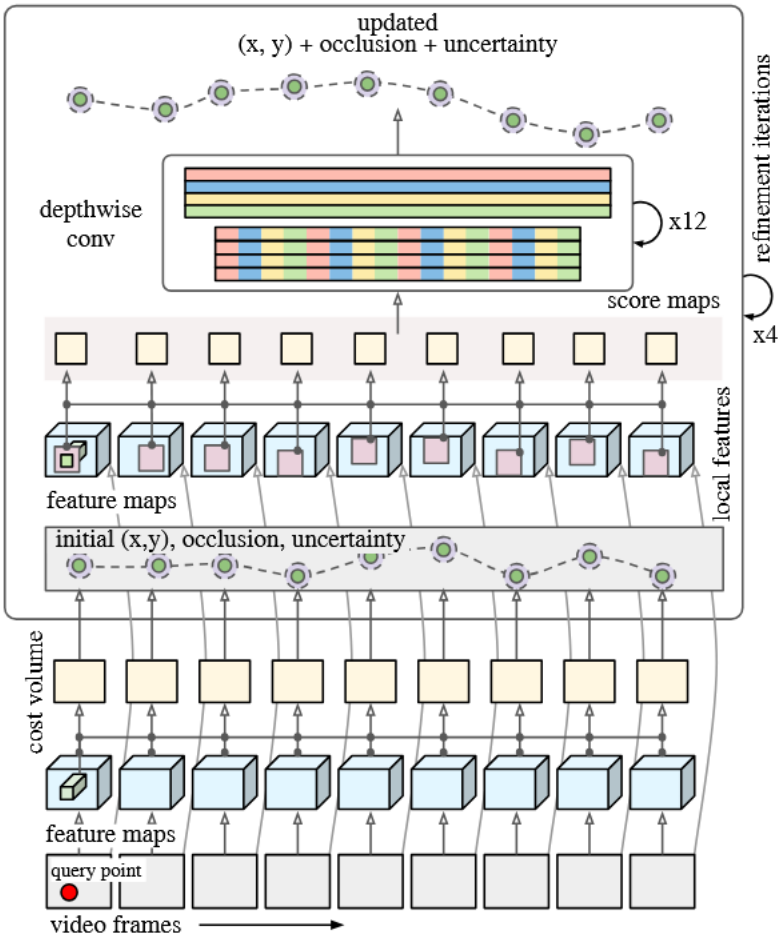


Figure 1: TAPIR architecture

The bottom part of the figure shows the **per-frame initialization**. It focuses on finding potential matches for a given query point in each new frame of the video. The key idea is to use a matching network (CNN) that compares the query point's features with the features of every other pixel in the target frame. This step outputs three initial values:

- initial guess for the trajectory (x, y)
- probability for occlusion
- uncertainty probability

In the second stage (**iterative refinement**) the above mentioned initial guesses are improved. For each potential position found during the initialization phase, the model defines a local neighborhood window. This window acts as a search area for refining the point's exact location. The refinement itself is achieved by comparing the visual features within this neighborhood to the features of the original query point. This comparison generates score maps which indicate the similarity between the query

point's features and every pixel within the neighborhood window. The highest-scoring pixel in this map represents the most probable refined position for the tracked point in that specific frame. This process is executed iteratively, allowing the model to correct small errors and ensure that the final trajectory is smooth and consistent over time.

Pros and Cons TAPIR provides smooth motion tracking that detects even small changes. Only in cases of significant occlusion or excessively fast movements is the point lost. TAPIR can also be run in real time. However, a major disadvantage is that only 2D trajectories are estimated and the lifting to 3D is not considered.

2.2.2 CoTracker

CoTracker is a paper that introduces another approach to point tracking in 2D videos. The core idea is that instead of tracking each point independently, it is more effective to track many points jointly, taking into account their dependencies and correlations. Figure 2 shows the a schema of the arcitecture of CoTracker.

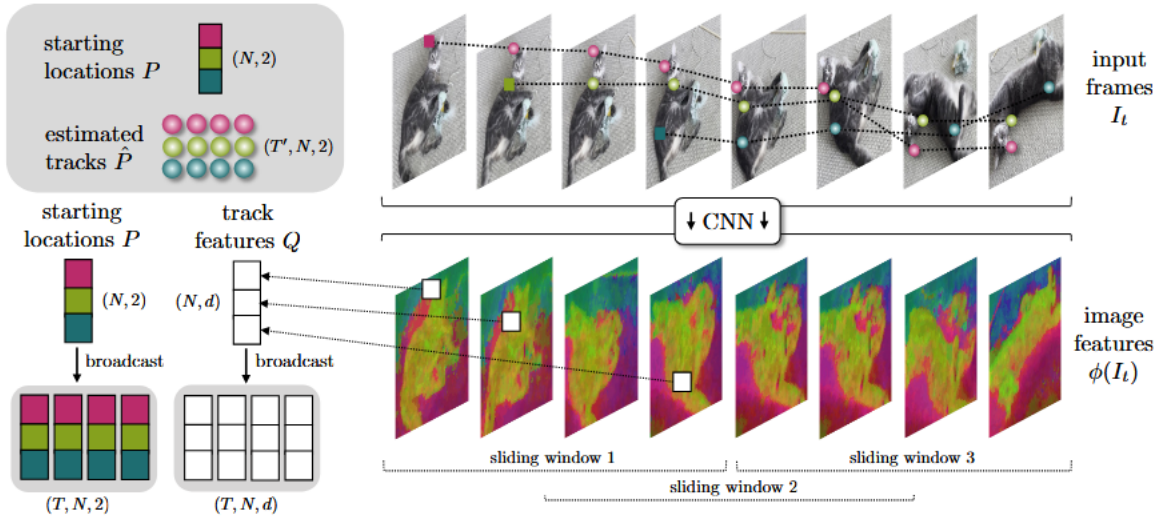


Figure 2: CoTracker architecture

Unlike TAPIR that treats each point's trajectory as an independent problem, CoTracker uses a Transformer-based network to model the relationships and dependencies between multiple points simultaneously. The heart of this system is a **attention mechanism** that enables the network to exchange information between different tracks and across various time steps within a given window of frames. Furthermore, CoTracker is an online algorithm, meaning it can process video frames in real-time. To handle very long videos, it uses a sliding window approach. To achieve this, the model

is trained in an unrolled fashion, like a Recurrent Neural Network. The predictions from one window are used to initialize the tracks for the next overlapping window. This allows the model to maintain track consistency and accuracy over long durations.

The output of CoTracker is:

- Trajectory (x, y) over all frames
- occlusion probability

Pros and Cons By jointly tracking many points with a Transformer, CoTracker exploits inter-point correlations, which improves robustness to occlusions and even when points leave the field of view. It also scales to tens of thousands of points on a single GPU. Its main drawbacks are sensitivity to domain gaps. It was trained largely on synthetic data and therfore can mis-handle reflections and shadows.

2.2.3 SpatialTracker

SpatialTracker is a method that uses the CoTracker approach and extends its 2D point tracking to the 3D domain. As input data it uses either 2D videos (RGB) or videos with depth information (RGBD). The architecture can be seen in Figure 3.

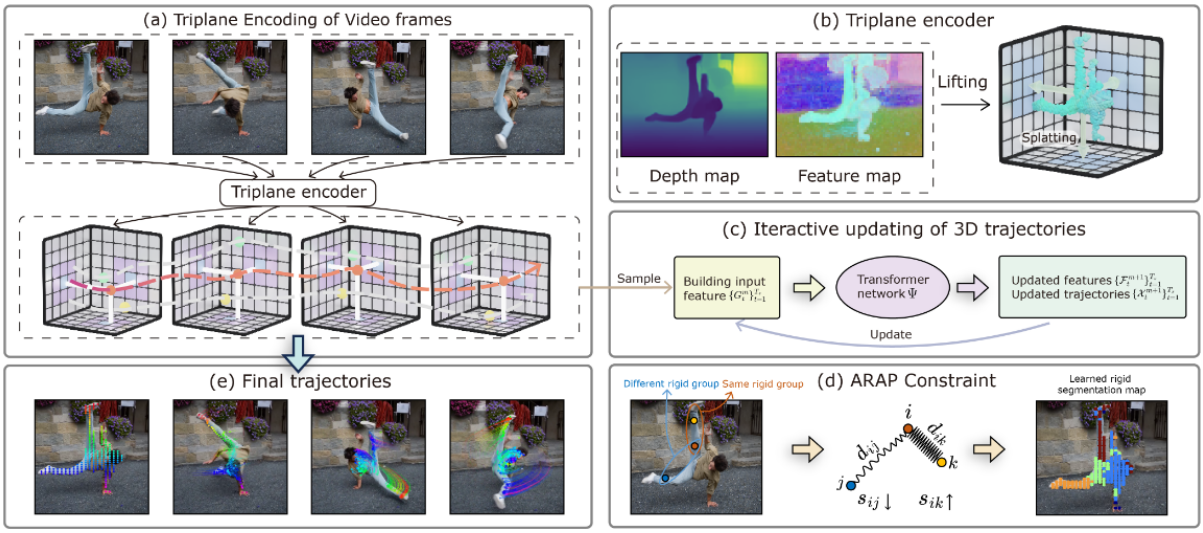


Figure 3: SpatialTracker architecture

SpatialTracker starts by estimating a depth map for each video frame and extracting dense image features, which are used to lift 2D pixels into 3D space to form a point cloud, see (a). For Monocular

Depth Estimation (MDE) ZoeDepth is used. These 3D points are then projected onto three orthogonal planes to create a compact triplane feature representation that enables efficient feature retrieval (b). An iterative transformer network refines the 3D trajectories of query points across short temporal windows, using extracted features from the triplanes as input (c). As a last step, the model learns a rigidity embedding that groups pixels with similar rigid motion. An As-Rigid-As-Possible (ARAP) constraint is then applied. The ARAP constraint enforces that 3D distances between points with similar rigidity embeddings remain constant over time, while a rigidity embedding combined with an ARAP constraint enforces locally consistent motion.

The output of SpatialTracker is:

- 3D trajectory (x, y, z)
- occlusion probability

Pros and Cons By lifting pixels to 3D (monocular depth → triplanes) and enforcing a rigidity embedding with an ARAP contraint This makes it robust to occlusions and out-of-plane motion and achieves strong results on long-range tracking benchmarks. Its accuracy depends on the quality/consistency of the depth input, and the pipeline incurs additional compute/memory from depth inference, triplane feature construction, and transformer updates compared to purely 2D trackers.

2.3 Adaption and implementation

As a result of the SpatialTracker providing the best tracking performance (as it can be seen in the videos in the GitHub) and also provides 3D data as output, we only focus on the SpatialTracker in follow-up work. The other models can considered in future research. This section gives an overview of the adaption we made to the already provided code base: Github SpatialTracker.

2.3.1 Sliding window approach

To make SpatialTracker operate reliably on long and/or high-resolution sequences, we extended the original code base to be able to process the input data in chunks. Instead of processing the entire clip at once, the video is split into temporal chunks (see Figure 4). For each chunk we prepend a small overlap to keep the tracking points consistent across chunks. The model is run on *overlap* + *chunk*, but only the predictions belonging to the non-overlap part are retained. This keeps peak memory usage approximately constant while preserving sufficient temporal context at chunk boundaries.

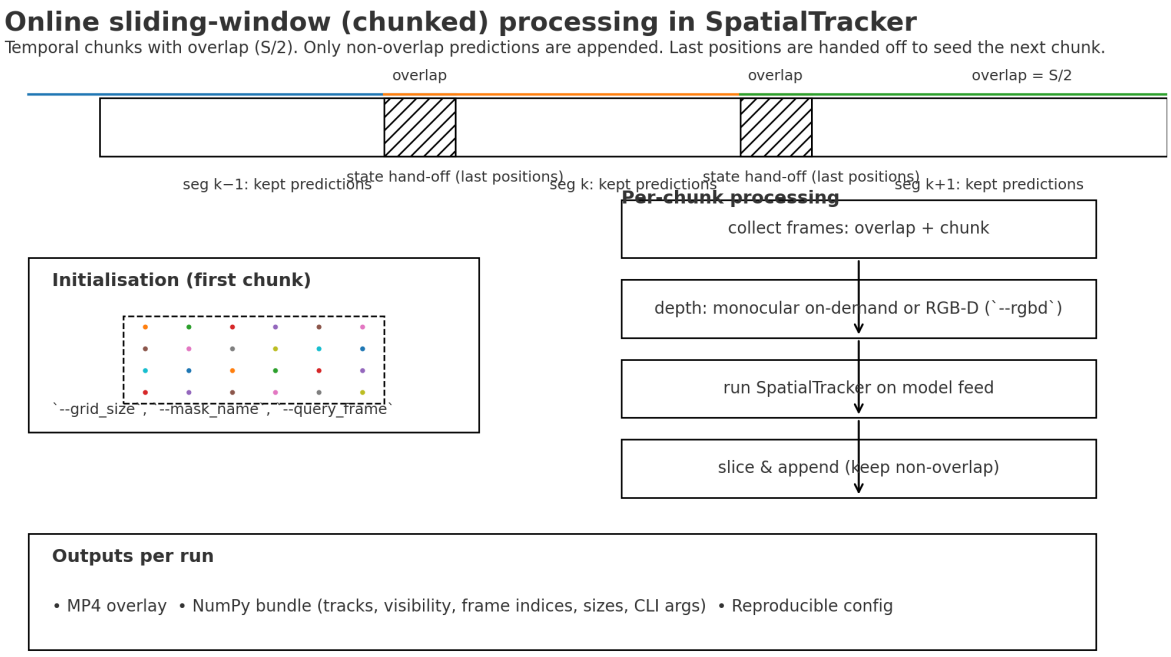


Figure 4: Online sliding-window processing

Initialization In the first processed segment, query points are initialised on a regular grid restricted to an optional segmentation mask. For subsequent segments we do not re-sample, instead the last predicted positions from the previous segment are used as the queries at the new segment start. In practice this yields stable identities and avoids repeated mask processing. If no valid points are available (e.g., prolonged occlusion), the pipeline proceeds with empty/dense queries until tracks re-emerge.

Depth handling The script supports both monocular and RGB-D inputs. By default monocular depth is computed on demand for the frames inside each model call. When RGBD data is provided per-frame depth maps (pre-aligned to the RGB preprocessing) are injected directly, bypassing the MDE. This path is used for comparisons with the depth data from a Time of Flight (ToF) camera.

Preprocessing and outputs We optionally sub-sample frames and apply crop/downsample operations before inference. All trajectories are mapped back to the overlay/original resolution when saving. Each run exports an MP4 with overlays and a NumPy bundle with trajectories, visibility flags, frame indices, spatial metadata, and the full CLI configuration to ensure reproducibility.

Key arguments

- `--chunk_size`: frames per output segment.
- `--s_length_model`: model window; overlap is half of this value.
- `--grid_size`, `--mask_name`, `--query_frame`: first-segment initialisation.
- `--rgbd`: use external depth instead of monocular depth.
- `--fps`, `--downsample`, `--crop`, `--crop_factor`: temporal/spatial preprocessing.
- Visualisation controls: `--point_size`, `--len_track`, `--fps_vis`, `--backward`, `--vis_support`.

Example

```
python chunked_demo.py \
  --root ./assets \
  --vid_name Gymnastik_1_5s \
  --mask_name Gymnastik_1_5s_mask.png \
  --grid_size 50 \
  --chunk_size 30 \
  --s_length_model 12 \
  --fps 1.0 \
  --downsample 0.8 \
  --rgbd
```

2.3.2 Depth Estimation

This section deals with depth estimation models used by SpatialTracker. It supports the output of a range of monocular depth estimation models such as *MiDaS*, *ZoeDepth*, *Metric3D*, *Marigold*, and *Depth-Anything*. But the provided code could only handle *ZoeDepth* out of the box, other modles had to be added by hand.

The inference process produces either relative or metric depth, and in some cases, such as with Depth-Anything, additional alignment steps are performed to calibrate the predicted relative depth to metric scale using a reference model. The script also converts the estimated depth maps into 3D point clouds using the camera intrinsics. The resulting 3D reconstructions are exported as colored point clouds in `.ply` format, which enables further processing.

By default the ZoeDepth estimator is used (see Figure 5).

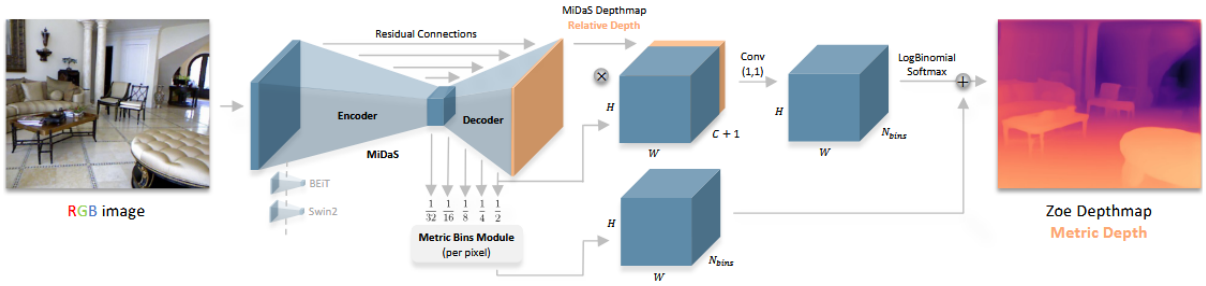


Figure 5: ZoeDepth architecture

ZeoDepth depth estimation ZeoDepth depth estimation consists of two stages. First, the model is extensively pre-trained on a vast amount of data to understand relative depth, learning which objects in a scene are closer or farther from each other without worrying about specific units of measurement. Second, the model is then fine-tuned on smaller datasets that contain ground truth metric depth (exact distances in meters). Instead of simply learning a single, precise value, ZoeDepth uses a unique **metric bins module** that estimates a range of possible depth values for each pixel.

For our work we extended the SpatialTracker to be able to ues another depth estimator. We were recommended Video Depth Anything: Consistent Depth Estimation for very long videos (Paper VideoDepthAnything) by Chen et al.

Video Depth Anything Video Depth Anything (see Figure 6) is build upon the strengths of Depth Anything to handle long-duration video sequences with high quality and temporal consistency. The authors introduce a lightweight **spatiotemporal head** on top of the Depth Anything V2 encoder that allows the network to share information across consecutive frames. Instead of relying on optical flow or camera pose (absolute depth values), they introduce a simple temporal consistency loss that encourages smooth changes in depth across frames. For long videos, the method processes clips segment by segment.

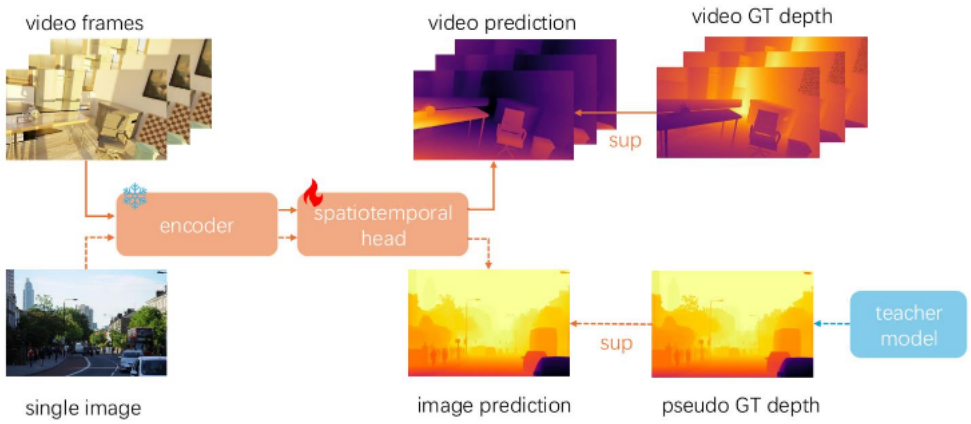
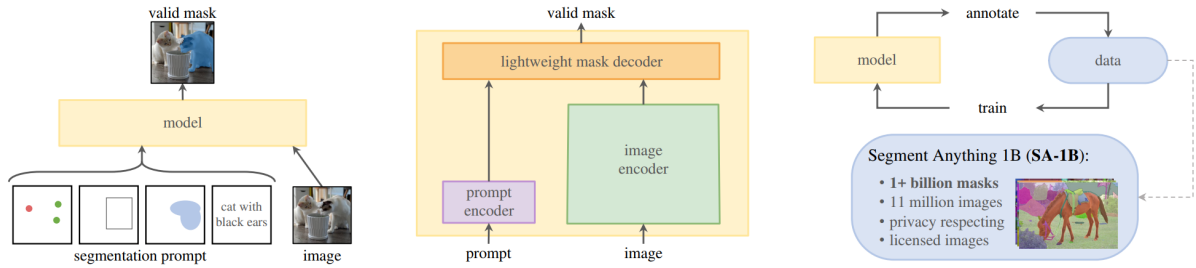


Figure 6: Video Depth Anything architecture

2.3.3 Segmentation

To determine the initial points tracked on the object, we generate a mask that highlight the sports equipment. For implementation, Meta AI's Segment Anything Model (SAM) [Github SAM] (see Figure 7) is used to generate segments, and then Clip (Paper CLIP) is used to select the most suitable segment for mask creation.



(a) **Task**: promptable segmentation (b) **Model**: Segment Anything Model (**SAM**) (c) **Data**: data engine (top) & dataset (bottom)

Figure 7: SAM Components

Segment Anything Segment Anything is a model for creating segments. SAM takes an image as input, optionally with prompts such as points, boxes, or masks. The image is processed by an image encoder to extract feature embeddings, which are then combined with the prompt embeddings. A mask decoder then predicts the pixel-precise masks from the combined embeddings. The whole thing was pre-trained on a dataset with over 11 million images, so that the model can now also create a mask for unknown objects (zero-shot generalization). The masks are output as well as a score that indicates the confidence. Text prompts are planned for use as prompts, but are not yet available. We therefore decided not to use prompts, so that segments are placed over the entire frame.

CLIP In a second step we use CLIP (Contrastive Language-Image Pretraining) to select the most semantically appropriate mask. To do this, we selected reference images of sports equipment and used CLIP to obtain an embedding vectors for each reference image, representing their visual features. The cosine-based similarity is calculated for each segment and reference image embedding. The mask with the highest similarity value is then selected as the result:

```
clip_model, preprocess = clip.load("ViT-B/32", device=device)
for i, mask_data in enumerate(masks):
            seg_mask = mask_data["segmentation"]
            seg_embed = encode_image_clip(clip_model, preprocess,
                seg_mask, device)
            score = max(torch.nn.functional.cosine_similarity(seg_embed
                , ref_embed).item()
                  for ref_embed, _ in ref_embeds)
```

Usage Parameters to be selected as input:

`--video-path`: The path to the video. `--reference-images`: Folders containing reference images of objects to be detected. `--use-center-priority`: If set, the segment is selected based on its position (otherwise based on its shape). Priority is given to the most central location.

First, the mask with the highest score is selected and displayed. Subsequent masks can then be chosen in descending order of score (see Figure 8). This procedure ensures that an appropriate mask is identified even in ambiguous cases (see Figure 9).

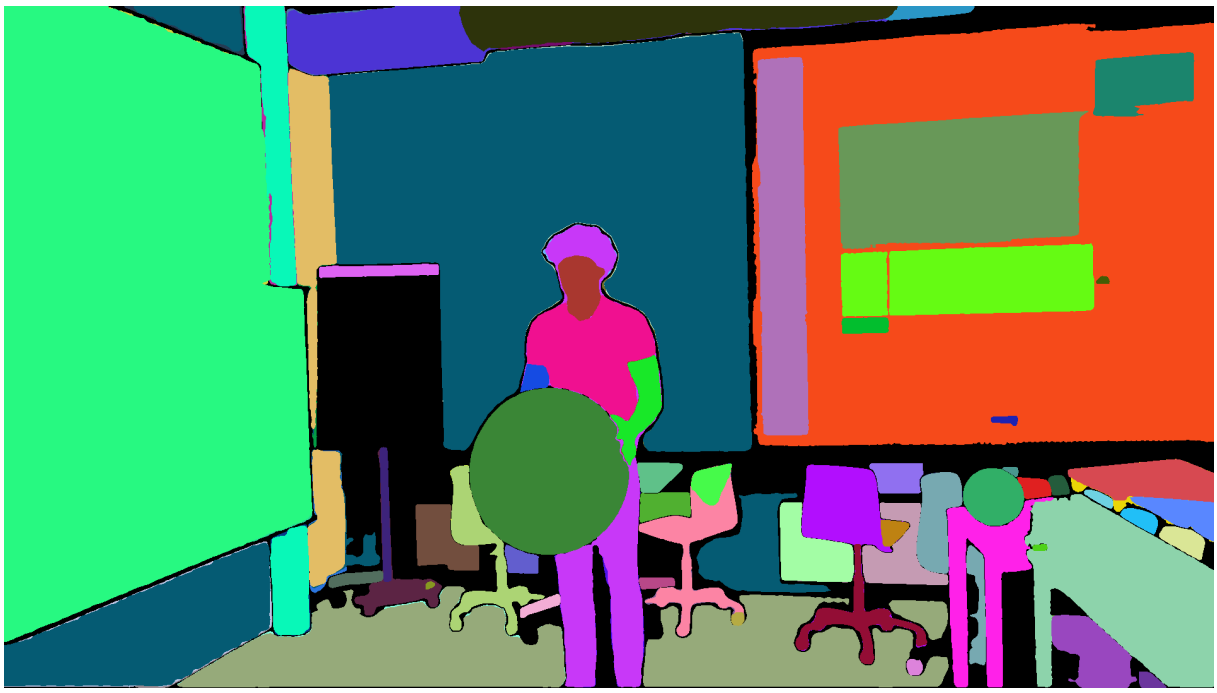


Figure 8: All detected segments with SAM

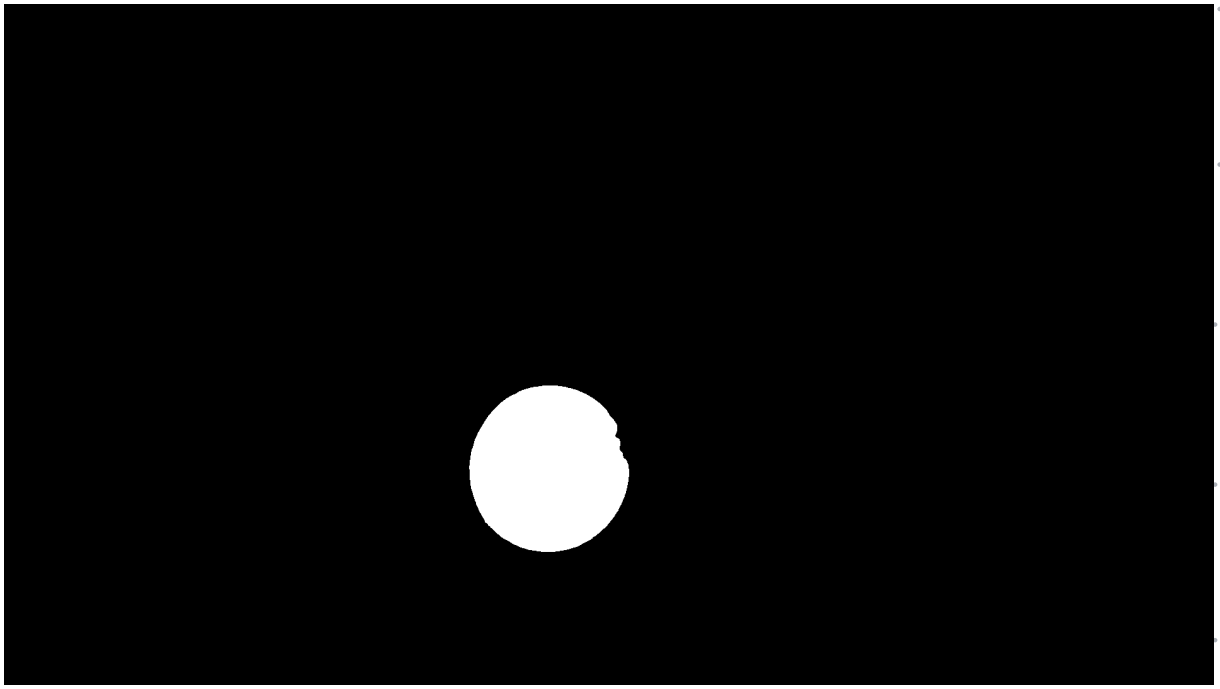


Figure 9: Resulting mask

Difficulties/Evaluation We chose a gym ball, a sponge, and a softball as reference images. These objects are detected correctly. There are difficulties in detecting the resistance band, as its shape in the first frame is ambiguous and not very distinctive. As a result, other objects such as the door handle and those with small rectangular shapes are often detected incorrectly. We therefore implemented an approach that selects the segment with the smallest distance from the center.

2.4 Data generation

For evaluation, a dataset consisting of approximately 50 videos was created. The dataset covers a range of objects, resolutions, and video lengths:

- Objects
  - Resistance band
  - Gymnastic ball
  - Soft ball
  - Sponge
- Video resolution
  - 1080p
  - 720p
  - 360p
- Video length
  - Between 5 and 30 seconds

2.4.1 Ground truth

To obtain reliable ground truth depth information for evaluating the tracking performance, depth images were recorded using a Time-of-Flight (ToF) camera. In our case the *Femto Bolt* from *Orbbec* was used (see Figure 10).

Figure 10: Femto Bolt depth camera

As the ToF camera outputs only individual frames, the color and depth streams were synchronized and merged into a continuous RGB-D video. The depth images, generated by an IR sensor, were read as *.raw* files and converted into NumPy arrays (*.npy*) for efficient loading and processing. In addition, the video frames were concatenated into an *.mp4* file.

For evaluation, the pre-recorded depth information was fed directly into the SpatialTracker, bypassing the monocular depth estimation module. This allowed a direct comparison between the model's predicted depth and the measured ToF depth, providing a robust basis for assessing the accuracy and reliability of the tracker.

3. Evaluation

ToDo

In this chapter we describe our evaluation process. All the metrics used in the three papers of TAPIR, CoTracker, and SpatialTracker did not fit well for our purpose. In the end, we agreed on two metrics,

which are described first. Afterwards, the results of each metric are discussed. In addition, the runtime is evaluated.

3.1. Runtime evaluation

To assess runtime performance, we measured system resource usage for different conditions (video length, resolution, grid size, and data type). Both RGB and RGB-D videos were tested to capture the computational overhead introduced by depth data.

Factor	Values tested	Metrics collected
Video durations	5 s, 10 s, 30 s	CPU usage (%), RAM (GB, %), Disk I/O (MB), GPU utilization (%) and memory (GB)
Resolutions	360p, 720p	
Grid sizes	20, 50, 100 tracking points	
Data type	RGB vs. RGB-D	

Metrics were recorded continuously during tracking. The results are visualized as plots showing temporal resource usage and comparisons across grid sizes, resolutions, and data types.

3.1.1 Results of the Runtime evaluation

Aspect	Findings	Reference
Execution Time	Runtime increases with video length and resolution. 720p ~5–10% slower than 360p.	see Figure 11
GPU Utilization & Memory	GPU is main bottleneck (70–100%). Memory: 22.8 GB (360p) vs. 23.3 GB (720p).	see Figure 12
CPU Usage	Low to moderate (7–9% avg, up to 65%). Not a bottleneck.	see Figure 13
System Memory (RAM)	360p: 8–22 GB. 720p: up to 45 GB. Resolution roughly doubles RAM needs.	see Figure 13
Disk I/O	Low. Increases with video length (35–200 MB). Not a bottleneck.	see Appendix I

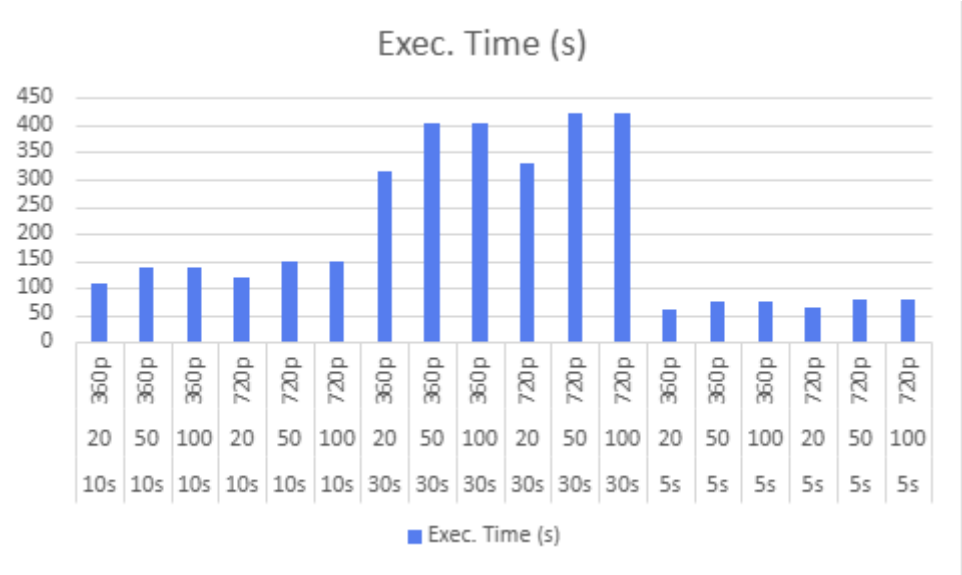


Figure 11: Execution time on RGB videos

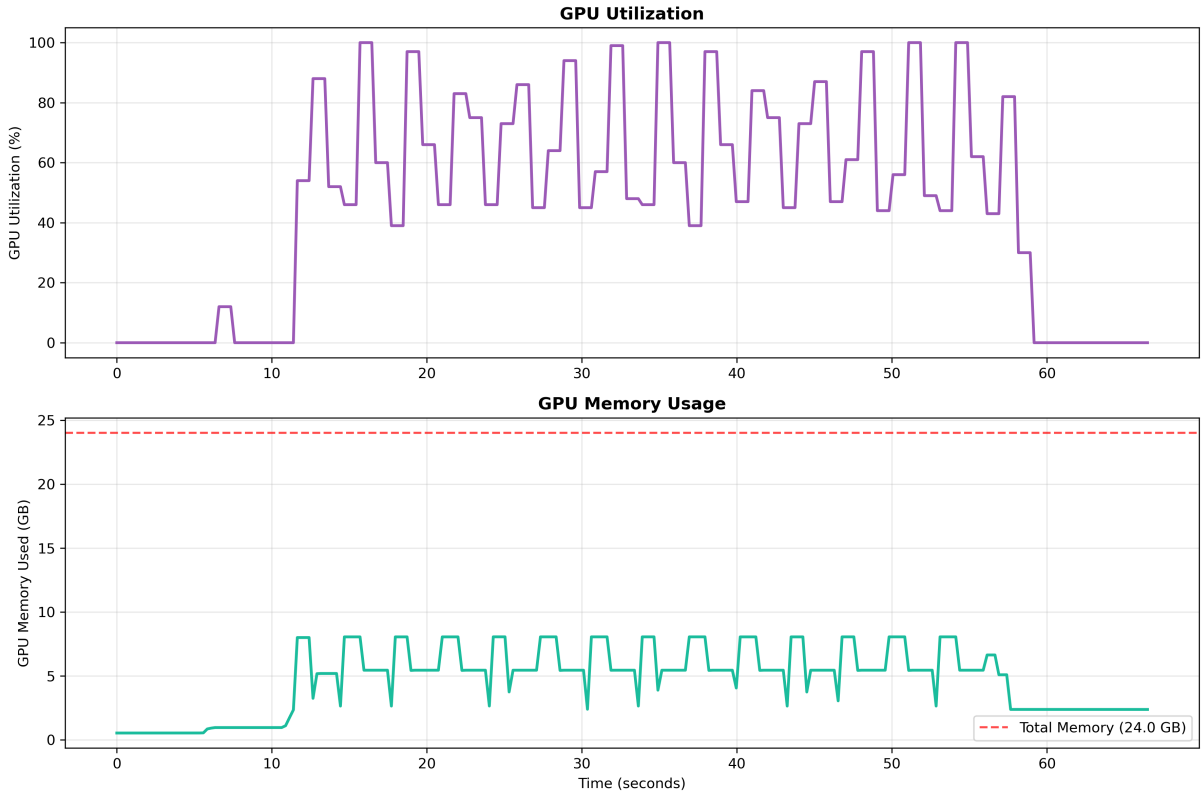


Figure 12: GPU and grapics memory utilization

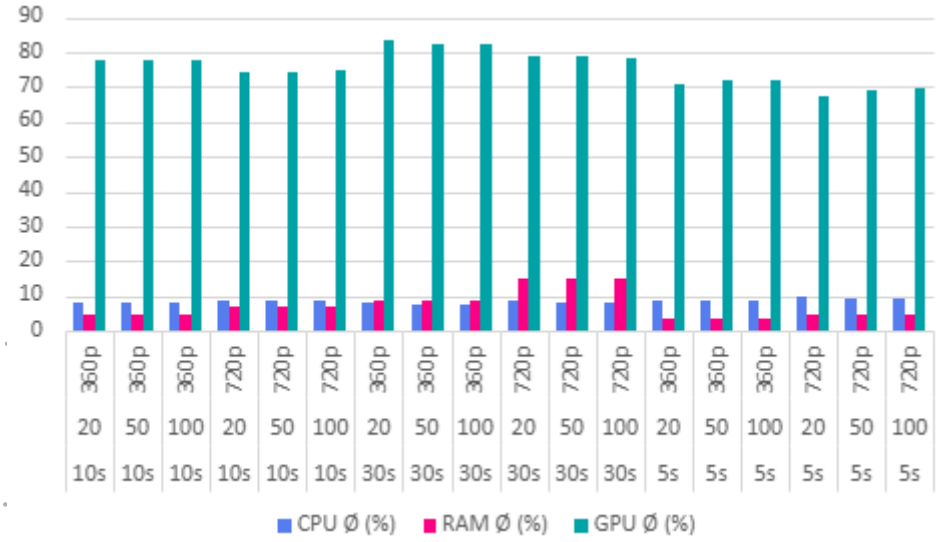


Figure 13: Resource utilization on RGB videos

RGB vs. RGB-D Processing with RGB-D consistently required more resources than RGB-only runs. Execution times were 10–20% longer (see Figure 14), GPU utilization increased by 1–3%, and memory usage rose by 200–500 MB (see Figure 15). Disk writes were also slightly higher (20–40 MB more per run). These differences reflect the overhead of handling the depth channel, while CPU usage remained almost identical in both modes. Overall, RGB-D provided richer tracking information, but at a stable and predictable computational cost across all test conditions.

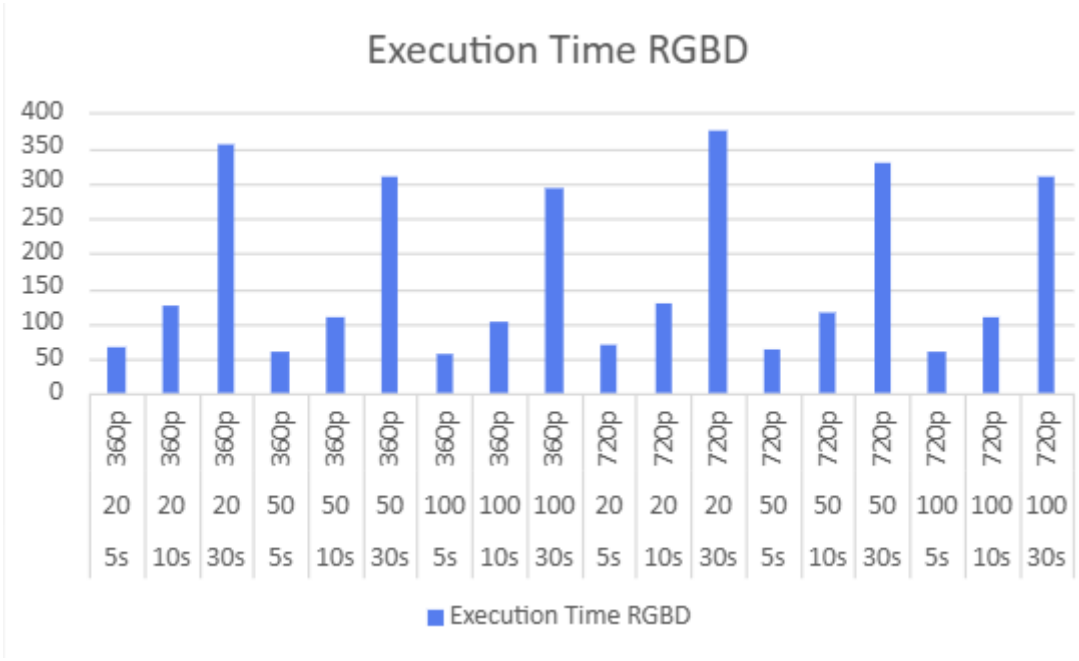


Figure 14: Execution time on RGBD videos

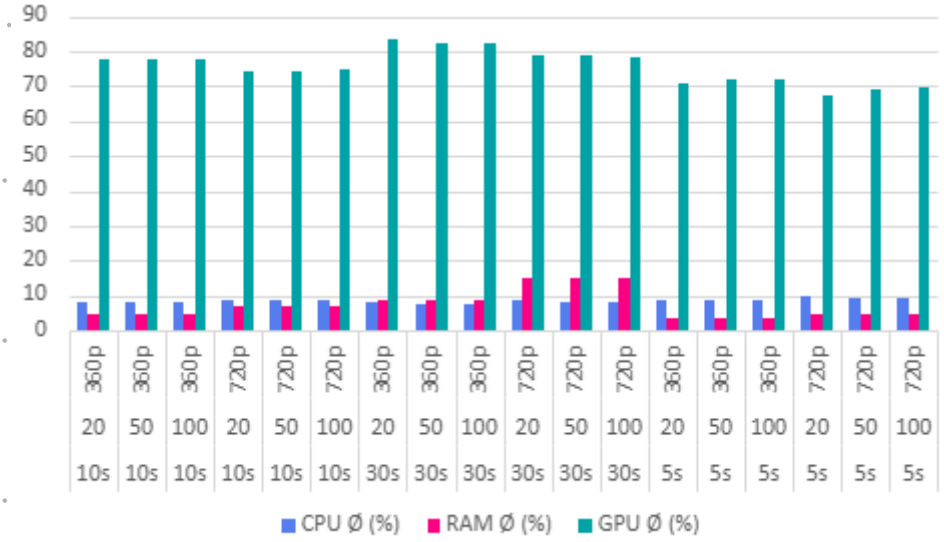


Figure 15: Resource utilization on RGBD videos

Scaling Effects Scaling factors such as resolution, grid size, and video length all showed proportional effects. Higher resolutions and larger grids led to longer runtimes and increased GPU memory demand, while longer videos extended processing time in a nearly linear fashion. Importantly, the

relative overhead of RGB-D compared to RGB remained consistent under all scaling conditions.

3.1.2 Key Findings

- Execution time grows with resolution and length; 720p is consistently ~5–10% slower than 360p.
- GPU resources are the main bottleneck, with utilization frequently reaching 100%.
- RAM demand scales strongly with resolution, and 720p runs may exceed typical workstation memory.
- CPU usage remains moderate and stable, while disk I/O is negligible.
- RGB-D adds predictable overhead (10–20% more runtime, ~200–500 MB extra memory, slightly higher GPU and disk usage) without changing the scaling behavior.

2D tracking efficiency

With this approach the tracking efficiency in the 2D space is evaluated. For this a short introduction into ArUco markers, as shown in the figure below, is needed. ArUco markers are binary square markers widely used in computer vision for camera pose estimation and object localization. Each marker consists of a unique black-and-white pattern which ensures robust detection under varying lighting conditions. By identifying the specific ID encoded in the marker's pattern, computer vision algorithms can distinguish between different markers in a scene. In our tracking application, the detection of the position of the ArUco markers establish a ground truth trajectory which can then be used to compare against the real trajectory from the SpatialTracker.

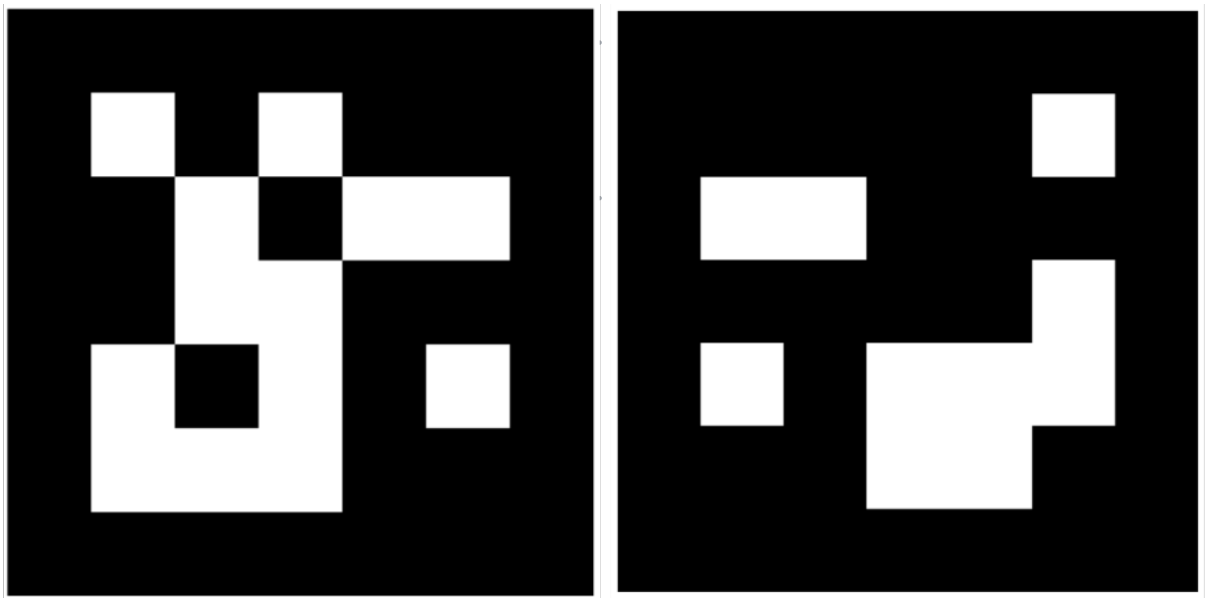


Figure 16: 5x5 ArUco markers

The figure below presents a schematic overview of the approach for one single frame. This setup is for demonstration purposes only and does not represent an actual experiment.

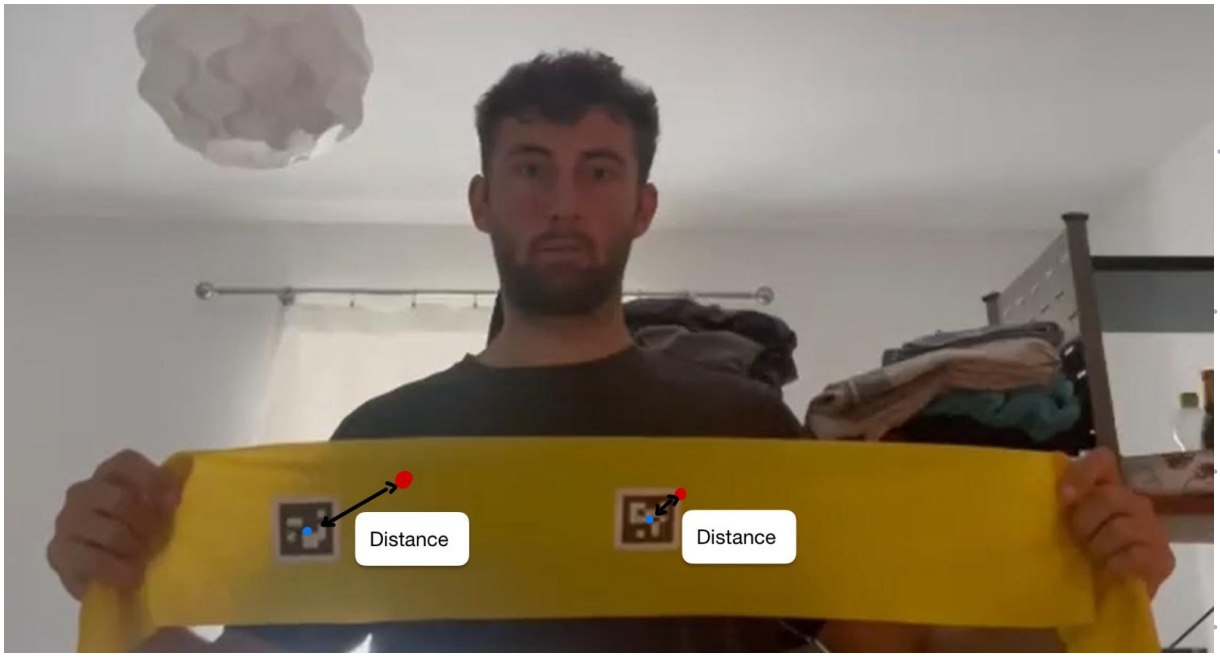


Figure 17: Schematic 2D evaluation with ArUco markers

The experimental setup involved recording three video sequences, each the two ArUco markers

shown above in a size of 4cm x 4cm attached to a resistance band which is recorded approximately from a distance of 1,5 meters. The setup includes three levels of difficulty in handling the resistance band:

1. Movement only – the band is simply moved without deformation.
2. With stretching – the band is actively stretched, introducing shape changes.
3. With occlusion – the ArUco markers are fully occluded during motion. The arm is stretching so that the other side of the Theraband is visible and the markings are completely covered for 1 second.

Each video was first processed using a Python script (`arucoDetection.py`) which detects the ArUco markers in every frame. For each detected marker, the pixel coordinates of its midpoint were calculated and stored as a NumPy array file (.npy) for subsequent analysis. In above figure these are the blue points.

In the next stage, the same video was fed into the SpatialTracker. The points selected for tracking were the midpoints of the ArUco markers in the first frame of the video. SpatialTracker then computes the corresponding 3D trajectories of the points over time which were exported in JSON format for further evaluation. In the schematic these are represented by the red circles.

In the python script `evaluateTracking.py`, both stored files are loaded. The Euclidean distance between the pixel coordinates obtained from the SpatialTracker and those from the ArUco marker detection is then computed for each frame. The tracking evaluation produces several quantitative metrics, namely the maximum error, the root mean square error (RMSE), and the standard deviation of the errors. A larger distance indicates poorer tracking performance, whereas a smaller distance corresponds to better accuracy.

Comparison between ToF ground truth and tracker

We implemented an end-to-end pipeline (`eval_pipeline.py`) to compare SpatialTracker outputs against ToF depth frames. Given a clip slug, the pipeline downloads and re-encodes the RGB video, derives a first-frame object mask (from the project's segmentation), runs the chunked online variant of SpatialTracker, fetches the corresponding ToF `.npy` frames, performs a depth-aware analysis, and generates a ToF overlay video with track visualization.

Coordinate mapping (processed → ToF)

Let (x_p, y_p) be tracker coordinates at the processed resolution $(W\_\text{proc}, H\_\text{proc})$, and $(W\_\text{tof}, H\_\text{tof})$ the ToF frame size. We map to ToF pixel indices by

$$s_x = \frac{W_{\text{tof}}}{W_{\text{proc}}}, \quad s_y = \frac{H_{\text{tof}}}{H_{\text{proc}}}, \qquad (x_{\text{tof}}, y_{\text{tof}}) = (\,\text{round}(s_x\, x_p),\ \text{round}(s_y\, y_p)).$$

Mask warp (geometric prior) The first-frame binary mask is used only as a **geometric prior**. A homography H is estimated from points that started inside the mask and are visible at $t = 0$ and t:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \qquad H = \text{RANSAC}(\{(x_0, y_0) \leftrightarrow (x_t, y_t)\}).$$

The mask is warped to frame t and slightly dilated to tolerate small warp errors. Membership in the warped mask counts as a positive vote.

Local ToF depth and object depth band For a projected point $(x\_\text{tof}, y\_\text{tof})$ we take a **robust local depth** as the median in a small window, ignoring zeros (no return).

To decide whether a tracked point is consistent with the ToF signal, we maintain an **object depth band** per frame. The band is estimated from ToF samples and smoothed over time using an exponential moving average. In the overlay path, depths are sampled inside the warped mask; in the metrics path, samples are taken at the current visible track locations (mask-agnostic). The band half-width uses the larger of a fixed minimum tolerance (≈6 cm) and a term proportional to the median absolute deviation (≈2.5×MAD). A point is accepted by depth if its local ToF median lies inside the current band.

Decision rule and outputs Geometry and depth are combined: a point counts as correct if it is **inside the warped mask** or **passes the depth test**. A short majority vote over recent frames reduces flicker from transient sensor dropouts. In the overlay, accepted samples are drawn **green**, rejected **red**, and optionally **yellow** if the tracker reports invisibility while ToF has no local return.

2D tracking efficiency

The following table summarizes the results for all three videos. Overall, we are looking at videos with an image size of 832x464 pixels:

Video / Condition	Marker ID	RMSE (pixels)	Std. deviation (pixels)	Max. deviation (pixels)
Movement only	0	2.53	1.34	7.69

Video / Condition	Marker ID	RMSE (pixels)	Std. deviation (pixels)	Max. deviation (pixels)
	1	4.82	2.80	10.17
With stretching	0	2.29	1.40	6.24
	1	6.20	3.06	12.08
With occlusion	0	3.87	3.39	40.21
	1	41.59	34.04	88.28

There is a clear difference between the values of the two Aruko markers. Marker 0 is located in the middle of the band and therefore performs less stretching. Marker 1, on the other hand, is located at the edge.

The movement and stretching achieve very good values with a maximum deviation of 12 pixels. The tracking positions are therefore consistently close to the marker. This is also evident from the low standard deviations.

Figure 18: Tracking errors during stretching

Figure 19: Tracking error during movement

There are problems with occlusion. The maximum deviation is large and persists throughout large parts of the video. The following figure shows the progression of the deviations. When the markers are occluded, the position information is lost. The tracker then assumes incorrect position values for both markers. When the marker becomes visible again, the tracker only recognizes marker 0 and the correct trajectories are determined there again.

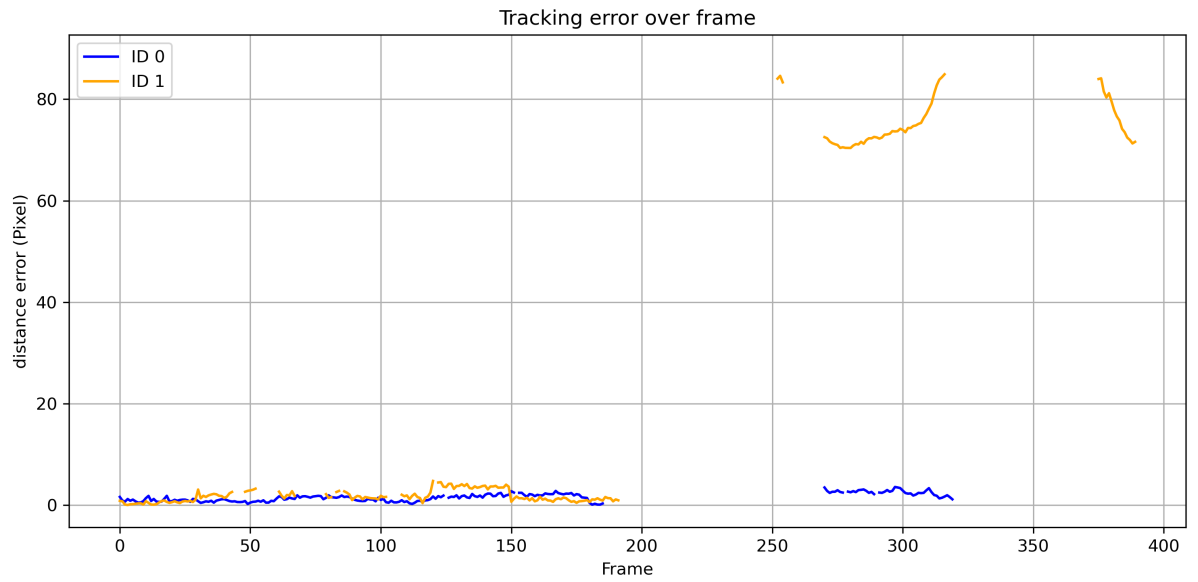


Figure 20: Error deviation during occlusion

The detection of occlusion is faulty. Occlusion is detected for a short period of time, but then another visible point is tracked (see following figure).

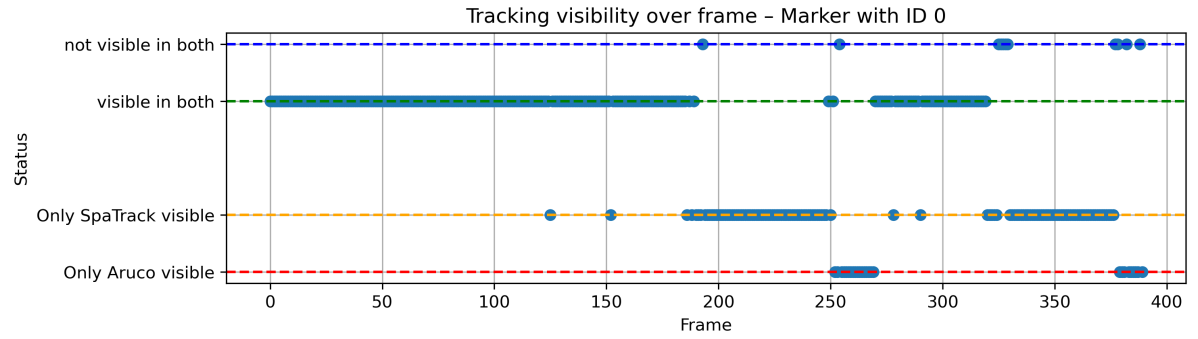


Figure 21: Comparison of visibility with Aruco Id 0

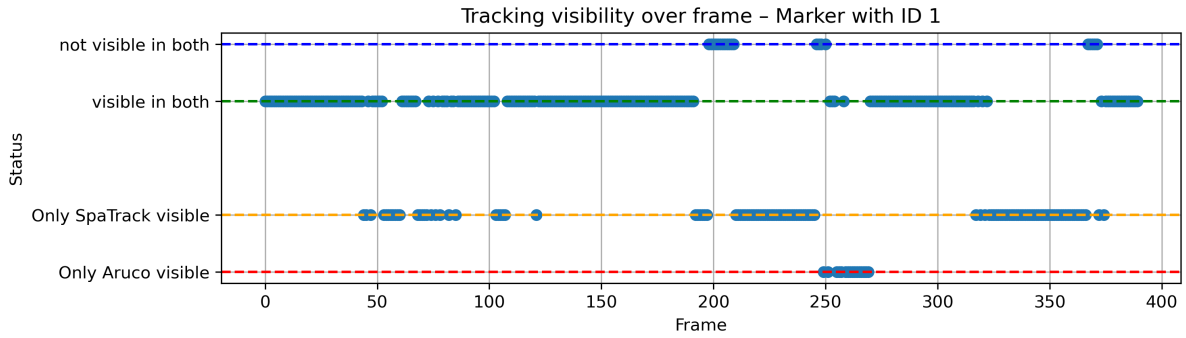


Figure 22: Comparison of visibility with Aruco Id 1

Overall, it can be said that movements and changes in shape are detected very well. There seem to be difficulties with occlusions. It should be noted that we are only using one video sequence as a reference. With this occlusion lasting approx. 100 frames, the Spatial Tracker does not show good results.

Comparison between ToF ground truth and tracker

The comparison between the ground truth provided by the ToF-camera and the results produced by our SpatialTracker shows that the result quality depends largely on the type of object being tracked. While simpler objects such as the “exercise ball” and the “blackboard eraser” were tracked consistently over the duration of the video, tracking the “Theraband” proved more challenging. Adding markings to the band improved the tracking performance to some extent, but not in a consistent manner — the results still varied considerably. This may be due to the nature of the Theraband, which is highly unstable in its dimensions and can be stretched into a line or compressed into a ball. This appears to be a complex issue that merits further investigation.

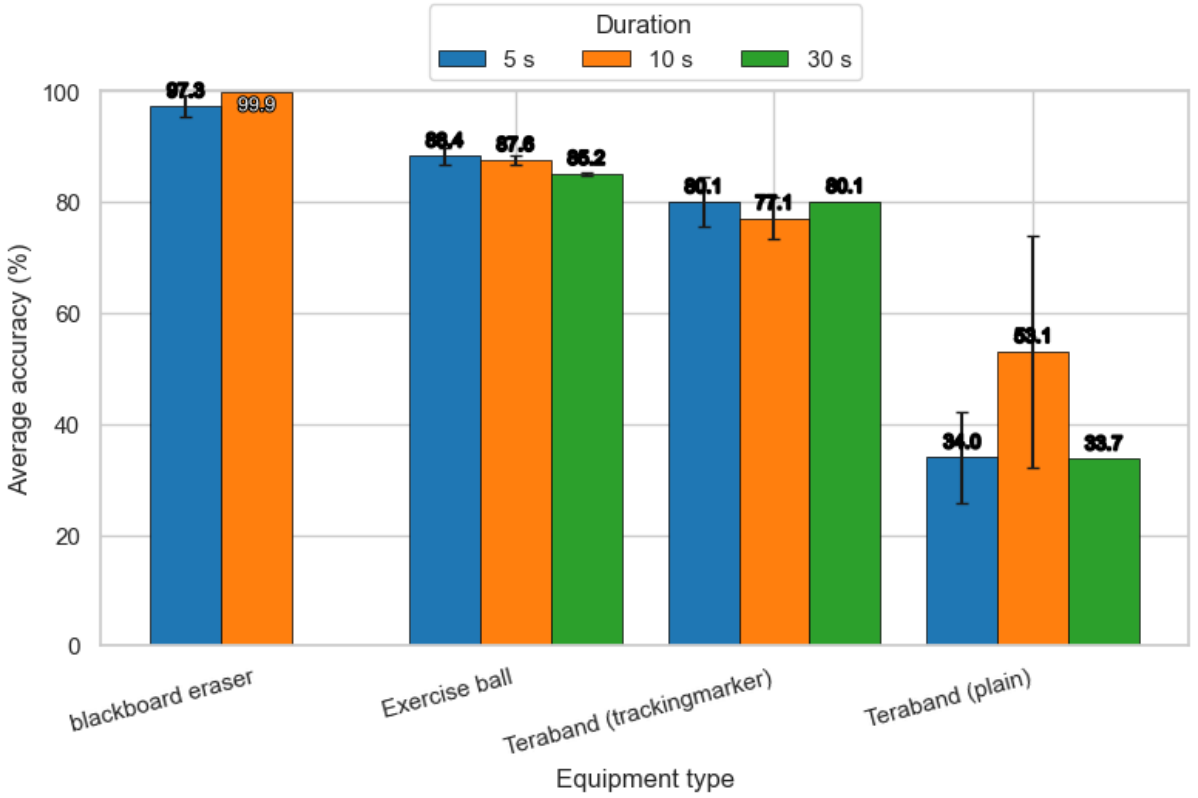


Figure 23: Average accuracy per equipment type and clip duration

In contrast, the tracking performance over time remained fairly consistent. The tracking quality in the third dimension only decreased slightly. This was not necessarily what we expected.

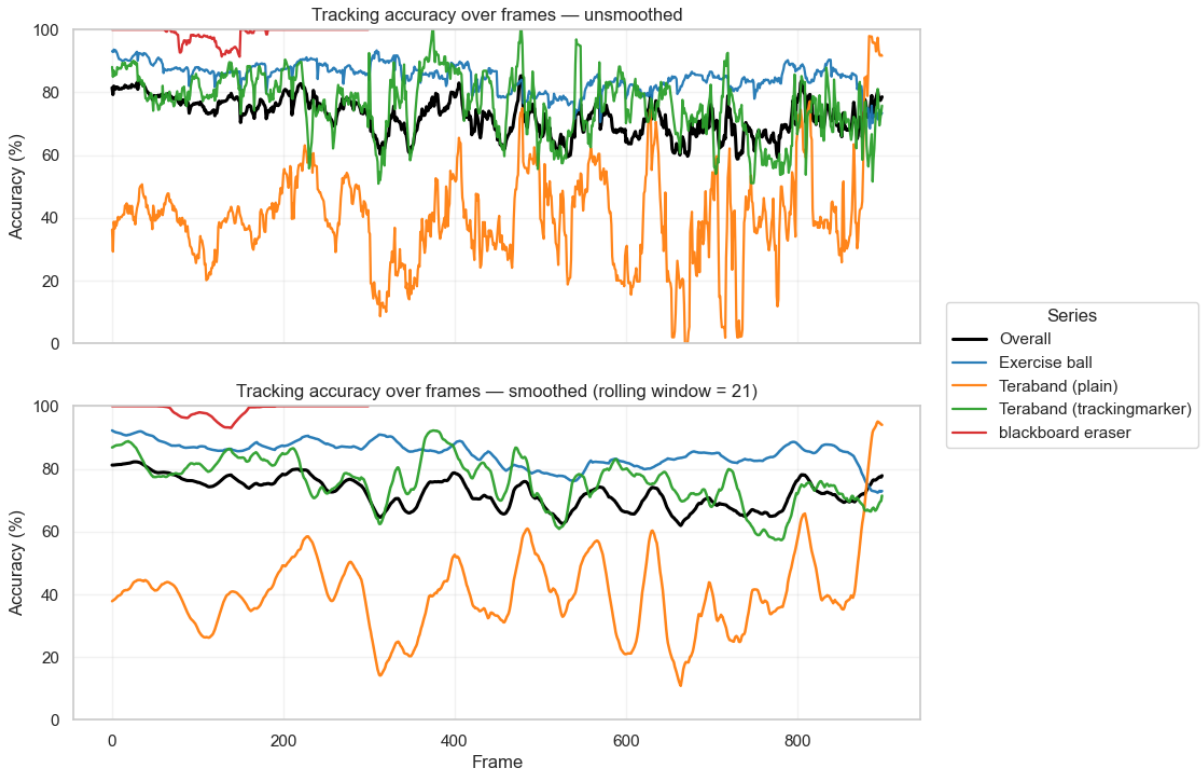


Figure 24: Average accuracy over time and per equipment type

Again, the plot above shows that tracking accuracy depends primarily on the type of workout equipment rather than on the duration.

Another point to note in the process is that it only worked reliably with good initial masks. For automated testing, some automatically generated masks — especially those for the Theraband — were of insufficient quality and had to be replaced manually. This was done to focus on the quality of tracking rather than on the inline masking.

Conclusion

Deformable Object Tracking remains a complex yet highly impactful area within computer vision, particularly for applications in healthcare, sports, and human-computer interaction. This project explored the challenges of tracking non-rigid objects, focusing on sports equipment such as resistance bands and gymnastic balls. By reviewing several approaches, the SpatialTracker method was identified as the most suitable for the task due to its ability to capture complex deformations over time.

In the next step, we made several adaptations to the SpatialTracker. To handle longer videos with-

out running into memory limitations, we implemented a sliding window approach. Furthermore, we added another monocular depth estimator called Video Depth Anything and a segmentation pipeline to automatically extract the region of interest. Additionally, we created a video library for testing purposes and used a time-of-flight camera to generate ground-truth depth data.

Overall, our evaluation shows that SpatialTracker can reliably track deformable objects under a range of conditions, but its performance varies strongly by object type. While simple objects such as gym balls and blackboard erasers were tracked consistently, resistance bands posed a significant challenge due to their highly variable shape, often reducing accuracy despite added markings. Runtime experiments revealed that GPU resources are the main bottleneck, with higher resolutions and longer clips increasing processing time and memory requirements, and RGB-D processing adding a predictable overhead of 10–20%. 2D efficiency tests with ArUco markers confirmed good accuracy under movement and stretching, but tracking degraded heavily under occlusion. Comparison with ToF ground truth further emphasized that accuracy depends more on object characteristics than on video length, and that reliable initialization masks are crucial for stable results.

Overall, the project lays a solid foundation for further research and development of automated tools that can support therapists and patients in exercise monitoring.

Future work

Adaptive re-sampling (e.g., after long occlusions) and learned cross-chunk consistency could further reduce boundary artefacts and drift. Runtime still scales with grid density; uncertainty-aware sparsification would improve throughput on dense masks.

As potential future work, a more detailed exploration of other tracking models could be considered. In particular, the TAPIR model was not evaluated in this project and could be an interesting topic for further research. **TODO: The CoTracker, however, is unlikely to be pursued, as it forms the foundation for the SpatialTracker and generally results in lower performance**

Another avenue for future work could involve relaxing the ARAP constraints within the SpatialTracker. However, this would require significant modifications to the algorithm. Currently, the ARAP restriction groups individual pixels together, which may be disadvantageous in cases where pixels need to move independently. This could be an advantage in case of strong deformations such as resistance bands.

TODO: another depth estimation model?

Appendix

Appendix I