CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Tracking the Rubik's Cube

Using point trackers for semi-automatic video annotation

Master's thesis in Applied Data Science

David Fagrell & Tim Zetterquist

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Tracking the Rubik's Cube

Using point trackers for semi-automatic video annotation

David Fagrell

Tim Zetterquist

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Tracking the Rubik's Cube
Using point trackers for semi-automatic video annotation
David Fagrell & Tim Zetterquist

Supervisor: Peter Damaschke, Department of Computer Science and Engineering
Advisor: Ludwig Friborg, Wiretronic
Examiner: Matti Karppa, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Tracking the Rubik's Cube
Using point trackers for semi-automatic video annotation
David Fagrell & Tim Zetterquist
Department of Data and IT
Chalmers University of Technology and University of Gothenburg

Abstract

The need for extensive manual labor in annotating video datasets causes a scarcity of them. Without video datasets of good quality and sufficient diversity, the research and development of machine learning algorithms in computer vision is impeded. To challenge this problem, we apply two point tracker algorithms CoTracker and TAPIR as a semi-automatic method for annotating video data and compare them to the object detector YOLOv5. We also conduct an ad-hoc qualitative analysis examining the properties of video data where the point trackers fail and succeed in their tasks. The evaluation of the methods used two datasets: the TAP-Vid DAVIS benchmark, and a new dataset made for this thesis - the Rotating Rubik's Cube dataset. With regards to intersection-over-union, position accuracy, and occlusion accuracy, YOLOv5 scored highest with TAPIR and CoTracker following respectively. The qualitative aspects in which the point trackers fail are due to camera movement, occlusions, and irregular light changes among others. However, given that CoTracker and TAPIR do not need to be trained before annotation, they can be used when no annotated data is available.

Keywords: Data science, machine learning, deep learning, point tracking, data annotation, object detection, thesis.

Acknowledgements

We would like to thank the colleagues at Wiretronic and our company supervisor, Ludwig Friborg, who has provided valuable and encouraging guidance throughout the whole thesis project. We would also like to thank Filip Persson who was very helpful with the technical assistance in the creation of the dataset. Without them, this project would not have been possible. Lastly, we would like to thank our academic supervisor, Peter Damaschke, who provided support and constructive feedback helpful in shaping this work.

David Fagrell & Tim Zetterquist, Gothenburg, 2024-05-30

Contents

List of Figures

List of Tables

1

Introduction

1.1 Annotation of Video Data

Machine learning (ML) has become a focal point in computer vision, demonstrating remarkable success in tasks such as image classification, object segmentation, and detection. Despite the prominence of ML, the scarcity of annotated video data poses a significant challenge, leading to the training of models on synthetic datasets not based on realistic scenarios, such as Flying Chairs or Flying Things (Neoral et al., 2024; Teed and Deng, 2020; Shah and Xiang, 2021). Synthetic, in this context, meaning data which is imitative of real scenarios by manipulating images. The Flying Chairs dataset by Dosovitskiy et al. (2015) for instance, has renderings of 3D chair models as foreground objects with a photograph of a landscape or mountain as background. Dosovitskiy et al. (2015) generate pairs of these images by applying linear transformations to the background and the chairs, making it appear as if the chairs are moving. When the models are applied to real-life environments with little similarity to the synthetic data they were trained on, they risk being erroneous (Shah & Xiang, 2021).

An important issue of ML is *garbage in, garbage out*, emphasizing the need for high-quality data to produce good models. Currently, many benchmark datasets used for computer vision, such as the Densely Annotated VIdeo Segmentation (DAVIS) dataset (Pont-Tuset et al., 2017) and the COCO-O dataset (Mao et al., 2023), contain annotated still images from independent photographs or all of the consecutive frames from videos. They have bounding boxes or masks isolating an area of interest in the image with an attached label, providing accurate ground-truth data representing the reality of what is being shown. However, annotated video sequences are far behind (Gaur et al., 2018). This is likely due to the dynamic behavior of videos where a short clip can be made up of thousands of images, and manually annotating these is not a trivial problem (Gaur et al., 2018).

For this study, we defined annotation in two different distinctions. Firstly, as described above, is the bounding box annotation. Secondly, a key-point annotation is defined as an x and y coordinate with an associated label in a given frame that represents the key-point of interest. The reason for having these two definitions of annotation is due to the test and evaluation of different models giving either a bounding box or a point as output. Furthermore, we distinguish between a key-point and a point. A key-point is a x and y coordinate denoting a specific part of an object

e.g., an eye, whilst a point is any x and y coordinate located within a frame.

Recent advancements in ML, exemplified by models such as CoTracker (Karaev et al., 2023) and TAPIR (Doersch et al., 2023), offer new possibilities in video data annotation which this thesis explores. These models are point tracking models, capable of predicting optical flow and tracking moving objects. Point tracking models take an image frame sequence as input and estimate a given point's localization in the consecutive frames and the point's trajectory throughout a video (F. Chen et al., 2022). To the authors' knowledge, point tracking models have not been applied for the task of semi-automatic video data annotation. Semi-automatic in this context means that the point trackers are given the ground-truth annotation of the frame in which a key-point initially appears and track said key-point in all consecutive frames. Thereby, a user only needs to annotate the first frame of a video in which a key-point appears rather than all of the frames in the video. By tracking specific parts of an object or the object itself, these models may overcome some challenges associated with manual video annotation, this is discussed in more depth in section 1.1.1.

This thesis aims to explore the differences in the performance of point tracking models and object detectors in semi-automatic video annotation and seeks to address the scarcity of high-quality annotated video data by utilizing state-of-the-art point tracking algorithms.

1.1.1 Manual Labor Problem

The number of video datasets with annotated point trajectories is even more scarce than datasets based on bounding boxes or masks, especially for arbitrary real-world videos. This probably has to do with the time-consuming process of annotation, e.g., when researchers created the TAP-Vid dataset, they found that *'on average, roughly 3.3 annotator hours are required to track 30 points through every frame on a 10-second video clip'* (Doersch et al., 2022, p.2). Given that the dataset consisted of 1,219 videos with roughly 25 points each, annotated by a small pool of workers, it is clear that this is a costly task in terms of time and money. Vondrick et al. (2012) estimates that even when using crowd-sourcing techniques, annotating a good video dataset with bounding boxes can cost up to tens of thousands of American dollars.

There are experiments on reducing the workload for manual annotation using object detectors or interpolation between key frames throughout a video (e.g. Ince et al., 2021; Ericsson, 2020; Chadwick and Newman, 2019). However, object detectors still need high-quality training data to be sufficiently accurate, thus if presented with data that is out of the distribution of the training data, the need for manual annotation arises again.

To reduce the cost of annotating data, it is common to outsource this task to other countries for cheaper labor. However, this often comes at a cost where workers endure poor working conditions or inadequate compensation (Le Ludec et al., 2023; Perrigo, 2022).

Since point trackers are trained to have a generalized ability to track any point

of interest, regardless of the type of objects in the videos they have been trained, we apply the point trackers to annotate any object in a video with out-of-the-box models. This differs from the application of object detectors for the same task, due to not needing training data that is within the distribution of whatever desired annotations the user has. By automating the process of video data annotation, or partly automating the process, the creation of training data for various computer vision models would require less manual labor and take less time. What we explore in this study is whether it is possible to improve this process by using point tracking algorithms instead of a classic object detection model.

Research Question 1 *Do point tracker models demonstrate superior location accuracy compared to object detector models in semi-automatic video annotation?*

Research Question 2 *What qualitative properties does video data have when point tracker models fail to track an object of interest?*

We answered these questions by running experiments on two datasets with a state-of-the-art object detection model and two state-of-the-art point tracker models (Doersch et al. (2023); Jocher et al. (2020); Karaev et al. (2023)). One of the datasets is a benchmark called the TAP-Vid DAVIS dataset (Doersch et al., 2022), which is based on the DAVIS dataset mentioned in section 1.1, and the other dataset is one we created for this study, called the Rotating Rubik's Cube dataset. We discuss the models and the datasets in more depth in sections 2.2 and 3.1 respectively.

1.2 Ethical Considerations

The data to be used follow GDPR regulations and adhere to principles of anonymity and consent. This includes the open-source TAP-Vid DAVIS dataset (Pont-Tuset et al., 2017) containing videos of people, animals, and inanimate objects in various scenarios.

2

Theory

2.1 Related Work

2.1.1 Annotated Video Datasets

There are a few benchmark datasets commonly used in the field of computer vision and video analysis. These are used for a variety of tasks and challenges including, but not limited to, classifying pixels or areas of pixels in frames to the background or foreground objects - video segmentation (Tsai et al., 2016), estimating the per-pixel motion between video frames - optical flow (Teed & Deng, 2020), and visual odometry, estimating the localization of the camera or object to which the camera is attached (Aqel et al., 2016).

These tasks require thoroughly annotated datasets to evaluate the performance of models and algorithms attempting to solve said tasks. Some commonly used benchmark datasets for such purposes are KITTI (Geiger et al., 2013), containing videos from a driver's point-of-view in different traffic situations, and the DAVIS dataset (Perazzi et al., 2016). Their popularity is due to the wide variety of videos showing various scenarios and differing motions while still containing dense annotations, allowing researchers to evaluate their models quantitatively and qualitatively (Doersch et al., 2022; Raji et al., 2023; Q. Wang et al., 2023).

Synthetic datasets such as the Flying Things++ by Mayer et al. (2016), are also commonly used (Karaev et al., 2023). Flying Things is a dataset consisting of everyday objects with randomized flying 3D trajectories. This dataset provides dense annotations of random objects and random textures of every object with smoothly flying and rotating trajectories for every scene (Mayer et al., 2016).

To our knowledge, no other annotated video datasets exist that are focused on rotating objects, making our dataset unique.

2.1.2 Point Tracking

Sand and Teller (2006) first mentions tracking individual pixels in videos using a middle-ground between feature matching and optical flow. In 2022, this task gets further developed when Harley et al. (2022) deploy Persistent Independent Particles (PIPs) that can track points through occlusions. PIPs take the RGB-values of a T-frame video, together with the $(x,\ y)$ coordinates of the queried tracking target and

create a $T \times 2$ matrix representing the target's positions across every frame T. The query points are tracked with a fixed sliding window, meaning the algorithm uses a rectangular region with a static width and height that slides across each frame to localize objects of interest. If a point is occluded, the algorithm restarts the tracking from the last frame the point was visible. A limitation of this approach is the fixed window size since the algorithm will lose track of the points if they are occluded for a longer duration than the size of the window (Karaev et al., 2023).

In the same year as PIPs were released, Doersch et al. (2022) deployed a benchmark dataset, TAP-Vid, to measure point tracking accuracy. This dataset contains real-world videos with human-made annotations of point trajectories and a baseline point-tracking model to which other point-tracking algorithms can be compared. This baseline model is, however, unable to track occluded points.

Zheng et al. (2023) upgrades PIPs into PIPs++, a simplified version that improves long-term tracking. They also released PointOdyssey, a benchmark dataset for long-term tracking. Both TAP-Vid and PIPs++ track points individually, while Q. Wang et al. (2023) use a volumetric representation in their model Tracking Everything Everywhere All at Once, that refines estimated correspondences in a canonical space. However, Track Everything All at Once is not suitable for many applications since it requires test-time optimization and is computationally expensive.

Another model, TAPIR (Doersch et al., 2023), is inspired by both TAP-Vid and PIPs and uses a feed-forward point tracking algorithm with a matching stage and a refinement stage. CoTracker (Karaev et al., 2023), on the other hand, uses a transformer-based model that, unlike previous models, tracks dense points in a frame jointly across a video sequence. Unlike the baseline model by Doersch et al. (2022), CoTracker and TAPIR also predict the trajectories during occlusions, however, their accuracy reduces as the duration of the occlusion increases (Doersch et al., 2023; Karaev et al., 2023).

2.1.3 Object Detection

Object detection is an important computer vision task that deals with detecting instances of visual objects in digital images or videos. The question it is supposed to answer is *What objects are where?* and can be broken down into two main topics that determine a model's accuracy; classification and localization (Zou et al., 2023). The process of object detection can further be broken down into three parts, as described by Zhao et al. (2019): *Informative region selection* which scans the image for possible object locations. *Feature extraction* which extracts visual features from the objects to discriminate between instances and provide a semantic representation. *Classification* which distinguishes the target object from all other categories to make representations more hierarchical, semantic, and informative for visual recognition.

Object detection has been a long-standing challenge for the computer vision community, but good advancements have been achieved over the past ten years with the emergence of deep learning and convolutional neural networks (CNN), which can now be considered to be industry-standard (Zhao et al., 2019). Most CNNs are

built up of multiple layers of nodes where each layer has a set of different feature maps that convolve over the image and identify small regions in the image space as features. These features hold relevant information that is then represented in a lower dimensional map in the pooling layer. This process is repeated until the last layer where every node from the pooled map is connected to a hidden layer of the network where a softmax classifier calculates the probabilities of each input image belonging to each of the possible image classes (Zhao et al., 2019).

Implementation and application of CNNs for object detection can be split into two categories. The first one is traditional object detection, which has been done using a region proposal-based method, and the second is where object detection is considered as a regression or classification problem using a unified framework method (Zhao et al., 2019).

2.1.3.1 Region Proposal-Based Method

The region proposal-based method uses a sliding-window method to generate candidate regions-of-interest (RoI) from the input image. After this first stage, features are extracted from these regions. Then a trained classifier identifies these candidate RoI's as belonging to classifications until, lastly, revising these classifications and optimizing the resulting detections (Zhiqiang & Jun, 2017). Generating the candidate regions is essential as this stage proposes where RoI's are, but since objects may appear in different sizes and aspect ratios anywhere in the image, there is also a need for many sliding windows of various sizes and aspect ratios as well. This results in this stage of the whole process requiring a lot of time to be executed and the use of many sliding windows which are redundant (Zhiqiang & Jun, 2017). It is also important to consider that using a fixed amount of sliding windows, in order to reduce computational resources and redundant sliding windows, may produce incorrect RoI's (Zhao et al., 2019).

From the RoI's, visual features are extracted and help the classifier to classify the object-of-interest within. The features are often hand-crafted representations in order to generate visual information usable for the classifier so that the distinctive features are robust to changes in shape, illumination, and environment (Lowe, 2004; Zhao et al., 2019; Zhiqiang and Jun, 2017). Examples of hand-crafted features are Scale Invariant Feature Transform (SIFT) and Histograms of Oriented Gradients (HOG) (Zhao et al., 2019; Zhiqiang and Jun, 2017). Even though these features are designed to be robust to contextual changes, it is difficult to design feature descriptors that capture all objects and the variation of external factors (Zhao et al., 2019; Zhiqiang and Jun, 2017).

After feature extraction, a classifier model is used to categorize the features. Commonly used models are Support Vector Machines (SVM), AdaBoost, or Deformable Part-based Model (Zhao et al., 2019). However, in this stage the multiple redundant windows generated in the first step of this process are still present. To handle the redundant windows, a non-maximum suppression algorithm is used as a post-processing stage (Zhao et al., 2019; Zhiqiang and Jun, 2017). This algorithm removes the redundant windows and optimizes the detection results. Different region pro-

posals are scored as either positive regions or negative background regions. Highly scored regions are selected and overlapping neighboring regions with low scores are removed or combined with the regions having higher scores in bounding box regression (Hosang et al., 2017; Zhao et al., 2019; Zhiqiang and Jun, 2017). Hosang et al. (2017) highlights issues with NMS due to it basing the decision to remove candidate regions with only one parameter. Based on this parameter, the algorithm removes neighboring regions with high scores due to them likely being false positives, but in crowded scenes and separate objects are close to one another, these regions are likely true positives. Thus, depending on how this parameter is set, it sacrifices precision or recall (Hosang et al., 2017).

2.1.3.2 Regression or Classification Based Method

Instead of being made up of several stages like region proposal-based methods, regression or classification based methods are one-step frameworks based on regression or classification (Zhao et al., 2019; Zhiqiang and Jun, 2017). Doing so reduces the time required to complete the task. This method divides the input image between a grid of cells where each cell predicts bounding boxes and class probability (Zhiqiang & Jun, 2017). There have been several approaches conceptualizing object detection as a regression or classification problem and therefore there are several CNN architectures used Zhao et al., 2019. In this project, we focus on the state-of-the-art model You Only Look Once (YOLO). We delve deeper into this model's architecture and processing in section 2.2.3.

2.1.4 Optical Flow

Point tracking and optical flow have similarities in their function. However, they differ in the fact that optical flow concerns the estimation of the displacement of each pixel in a video's frame sequence (Fortun et al., 2015; F. Chen et al., 2022), and while it is caused by the relative motion between the observer and the objects in the scene it does not necessarily concern the actual motion of the objects within the scene (Fortun et al., 2015). This differs from point tracking where the objective is to estimate the motion of a point, an object, or a specific region of interest in the frame sequence (F. Chen et al., 2022). It is possible to estimate the optical flow using CoTracker as it can track every pixel, however, this is not how the algorithm is applied in this project as we are focusing on point tracking. We do not apply the models for optical flow because we want specific key-points and objects-of-interest in motion annotated rather than the whole frame annotated.

2.1.5 Semi-automatic Annotation

Typically, video annotation tools have human annotators that locate the object of interest by drawing a bounding box around it and giving it a label (Yuen et al., 2009; Vondrick et al., 2012). To reduce the workload for annotators most tools offer label propagation which automatically generates labels for the same object in subsequent frames, and label linear interpolation which handles object motion

throughout frames. These tools, however, work only if the motion of the object is linear with constant speed (B.-L. Wang et al., 2018).

More recent works have integrated computer vision algorithms based on deep neural networks for object detection and point tracking. B.-L. Wang et al. (2018) combines an object detector and a point tracker with the video labeling tool VATIC to provide a semi-automatic video labeling system, resulting in a 32% decrease in workload based on the KITTI dataset.

Semi-automatic annotation methods have shown to decrease the need for manual labor but the need for a human in the loop still remains (Ince et al., 2021). This is due to object detector models' need for annotated training data from the start. Without annotated training data, the object detector is unable to analyze and predict novel non-annotated data (Bolya et al., 2020). Furthermore, object detectors are limited to classify and annotate only objects it has been trained on, making it irrelevant for annotation of data to which it is not familiar with (Bolya et al., 2020). Although point trackers also require training data in order to predict motion trajectories, they can be applied to novel data containing objects to which they have not previously been exposed to and predict trajectories (Doersch et al., 2023; Karaev et al., 2023). Consequentially, this makes point trackers more generalizable than object detectors for the purpose of annotating video data.

2.1.6 Segment Anything Meets Point Tracking

Raji et al. (2023) integrated CoTracker and other point tracking models with the Segment Anything Model (SAM) to develop a new method of video segmentation, creating Segment Anything Meets Point Tracking. The model selects query points denoting the target object or the background. The queries are used as a basis for generating point trajectories which are then used as prompts for SAM to produce segmentation masks. This is repeated iteratively throughout the whole video to produce the trajectory for an object of interest (Raji et al., 2023).

2.2 Model Choice

For this project, two point tracker models were used for annotation, *CoTracker* and *TAPIR*, and one object detector, *YOLOv5*, which was used as a baseline model to compare the point trackers against.

We chose these models as they were considered and evaluated to be the state-of-the-art based on the authors' research. To ensure this for the point trackers, we replicated an experiment by Karaev et al. (2023) and Doersch et al. (2023) on the TAP-Vid DAVIS dataset with both TAPIR and CoTracker.

2.2.1 TAPIR: Tracking Any Point with per-frame Initialization and temporal Refinement

TAPIR is a model that can track a given query point on any physical surface throughout a video sequence. Given a video and a query point, TAPIR estimates the 2D location p_t that the query point corresponds to in every other frame t, as well as a probability o_t that the query point is occluded and a probability u_t on the uncertainty of the estimated location where a low value means there is a high degree of confidence in the estimation. This is done through a two-step process, a matching stage, and a refinement stage (Doersch et al., 2023).

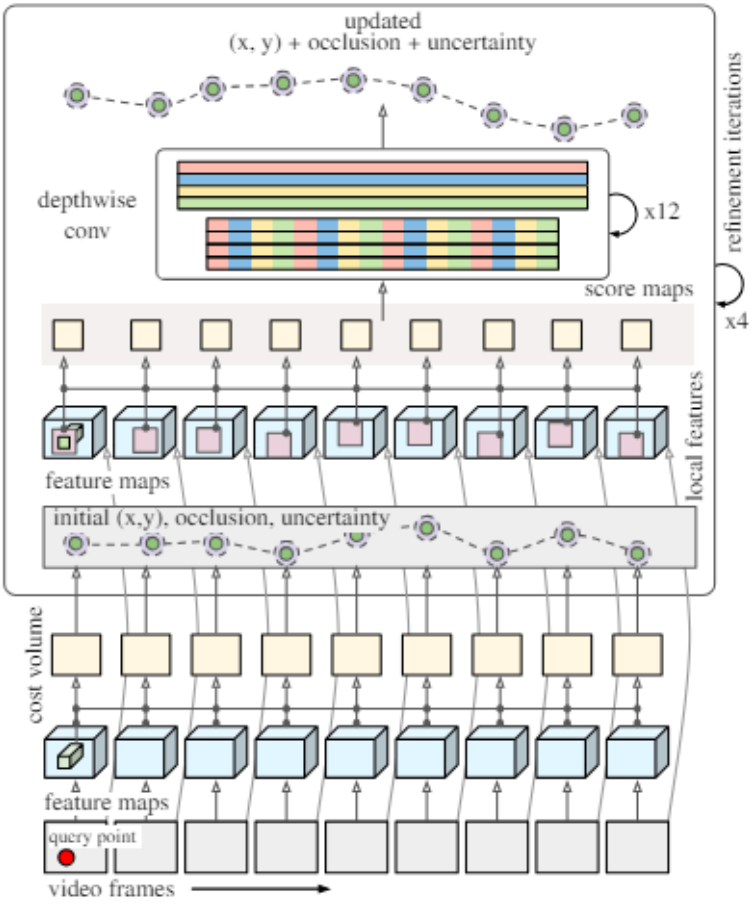

Figure 2.1: TAPIR architecture summary. The model first computes global comparisons between query point features and the features for every other frame to estimate the initial track, including uncertainty estimate. Next, features are extracted from a local neighborhood (shown in pink) surrounding the initial track estimate, these are compared to the query feature at a higher resolution. The similarities are post-processed with a temporal depthwise convolutional network to update the position estimate. The updated position is fed back into the next iteration of refinement for a fixed number of iterations. Multi-scale pyramids are not shown for simplicity (Doersch et al., 2023, p.6).

In the matching stage, TAPIR matches candidate locations to the initial query point in all subsequent frames by comparing query point features with all other features, and post-process the similarities to get an initial estimate. This is done to identify the query point's most likely related point for every frame to follow the query point's trajectory. In the refinement stage, TAPIR updates the trajectory of each query point and the query features based on local features for that frame. Both the matching and refinement stages mostly rely on similarities which are computed

by the dot product of the query features and surrounding features (Doersch et al., 2023).

A query point's track is initialized by estimating a trajectory by finding the best match within each frame independently. This process involves a *cost volume* that for each frame t, computes the dot product between the query feature F_q and all features in the t'th frame. F_q are extracted via bilinear interpolation and the cost volume is computed using a feature map $F \in \mathbb{R}^{T \times H/8 \times W/8 \times C}$, where T is the number of frames, H and W is height and width, and C is the number of channels, computed with a TSM-ResNet-18 backbone (J. Lin et al., 2021). The cost volume is then post-processed by applying a ConvNet to the cost volume corresponding to frame t. The ConvNet produces a heatmap of shape $H/8 \times W/8 \times 1$ for position prediction and a single scalar logit for the occlusion estimate. Then the heatmap is converted to a position estimate by a spatial soft argmax, i.e., a softmax across space to make the heatmap positive and sum to 1 (Doersch et al., 2023).

A position uncertainty estimation is added to each iteration to suppress low-accuracy predictions. According to Doersch et al. (2023), this is applied since it is worse if the algorithm predicts a vastly incorrect position than if it incorrectly marks the point as occluded. Estimating the wrong location tends to happen if there are many potential matches within a frame, and the authors found it beneficial to add the probability of the uncertainty of the estimated location. This is quantified by having the occlusion pathway output a second logit that estimates the probability of the uncertainty u_t^0 that the prediction is likely to be far enough from the ground truth that it is within a threshold δ of the ground truth.

Therefore, a loss function for the initialization for a video frame t is $\mathcal{L}(p_t^0, o_t^0, u_t^0)$ where $p_t^0 = (x_t^0, y_t^0)$, o_t^0 is the occlusion, and $\mathcal{L}$ is defined as:

$$\begin{aligned} \mathcal{L}(p_t, o_t, u_t) = &\ \text{Huber}(\hat{p}_t, p_t) \cdot (1 - \hat{o}_t) \\ &+ \text{BCE}(\hat{o}_t, o_t) \\ &+ \text{BCE}(\hat{u}_t, u_t) \cdot (1 - \hat{o}_t) \end{aligned} \tag{2.1}$$

$$\text{where,} \quad \hat{u}_t = \begin{cases} 1 & \text{if } d(\hat{p}_t, p_t) > \delta \\ 0 & \text{otherwise} \end{cases}$$

In this equation, $\hat{o_t} \in \{0, 1\}$ and $\hat{p} \in \mathbb{R}^2$ are the ground truth occlusion and point location respectively. d is the Euclidean distance, δ is the distance threshold, Huber is a Huber loss, and BCE is a binary cross entropy. $\hat{u_t}$ is the target for the uncertainty estimate u_t and is computed from the ground truth position and the network's prediction: 0 if the prediction is within the threshold δ and 1 otherwise. The algorithm should output that the point is visible if it's both predicted as unoccluded and if the model is confident in the prediction. To do this, a soft combination of the two probabilities is applied, and the algorithm predicts the query point to be visible if $(1 - u_t) * (1 - o_t) > 0.5$ (Doersch et al., 2023).

From the matching stage, TAPIR has each frame's estimated position, occlusion,

and uncertainty. In the refinement stage, the goal of each iteration i is to compute an update $(\Delta p_t^i, \Delta o_t^i, \Delta u_t^i)$ that brings the estimate closer to the ground truth. The update is based on local score maps which capture the query point dot product (similarity) to the neighboring features. To compute these, a pyramid of feature maps in different resolutions is used, so given a trajectory, they have the shape $(H' \times W' \times L)$ where $H' = W' = 7$, the size of the local neighborhood, while L is the number of levels in the spatial pyramid. These similarities are post-processed with a network to predict the refined position, occlusion, and uncertainty, similarly to the initialization, but with the exception that the local score maps are included. The output of the network at the $i$tth iteration is a residual $(\Delta p_t^i, \Delta o_t^i, \Delta u_t^i, \Delta F_{q,t,i})$ which is added to the position, occlusion, uncertainty estimate, and the query feature (Doersch et al., 2023).

These positions and local score maps are fed into a 12-block convolutional network to compute an update $(\Delta p_t^i, \Delta o_t^i, \Delta u_t^i, \Delta F_{q,t,i})$, where each block consists of a 1×1 convolution block and a depthwise block. When $(\Delta p_t^i, \Delta o_t^i, \Delta u_t^i, \Delta F_{q,t,i})$ is obtained, the refinement stage can be iterated as many times as desired with the same parameters. At each iteration, the loss function $\mathcal{L}$ is applied to the output, weighting each iteration the same as the initialization (Doersch et al., 2023).

TAPIR is trained with 64 TPU-v3 cores on the synthetic Kubric MOVi-E (Greff et al., 2022) dataset, albeit with one modification. The Kubric MOVi-E dataset contains videos where the camera is looking directly at the moving objects, but to make it more similar to real videos, a camera panning function was added to close the gap between synthetic and real video data. The training consisted of 50,000 steps with a cosine learning rate schedule, meaning that the model adjusts the learning rate based on a cosine function (Loshchilov & Hutter, 2016), and 1,000 warmup steps (Doersch et al., 2023).

Even though TAPIR is a state-of-the-art model, it has some limitations. It especially has difficulties in precisely perceiving the boundary between background and foreground, as well as struggling with large appearance changes (Doersch et al., 2023)

2.2.2 CoTracker

CoTracker from Karaev et al. (2023) improved upon previous models from Doersch et al. (2023) and Harley et al. (2022) by not only tracking the individual points within a frame, as it was done before, but also computing the correlations that may exist between points. CoTracker's novel architecture allowed for the joint tracking of multiple points that have a strong correlation, e.g. points that belong to the same physical surface of an object in a video, by applying a transformer network modeling the points with specialized attention layers (Karaev et al., 2023).

The goal of CoTracker is to track point tracks $P_t^i = (x_t^i, y_t^i) \in \mathbb{R}^2, t = t^i, ..., T, i = 1, ..., N$ where $t^i \in \{1, ..., T\}$ is the time when the track starts, throughout a video $V = (I_t)_{t=1}^T$ consisting of multiple RGB frames. This is done by estimating a trajectory for each of the query points throughout the video and predicting the query

points visibility $v_t^i \in \{0,1\}$. Thus, the input to the model is a video V, and the starting locations and times $(P_{t_i}^i, t^i)_{i=1}^N$ of N tracks, and outputs an estimate $(\hat{P}_t^i = (\hat{x}_t^i, \hat{y}_t^i), \hat{v}_t^i)$ of the track locations and visibility for all valid $(t \geq t^i)$ times.

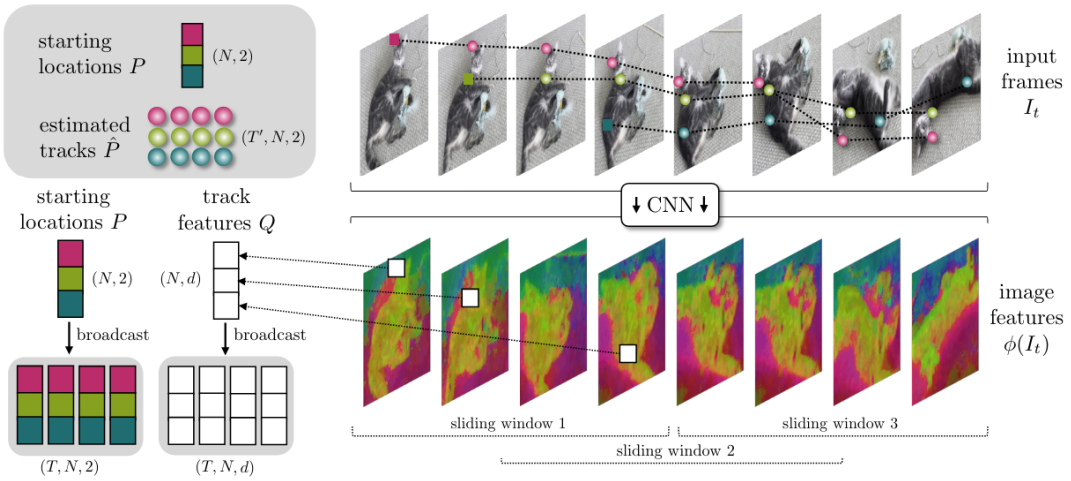

Figure 2.2: CoTracker architecture. Convolutional features $\phi(I_t)$ are computed for each frame and processed using sliding windows. Track features Q are initialized by bilinearly sampling from $\phi(I_t)$ with starting point locations P. Locations P also initializes the estimated tracks $\hat{P}$ (Karaev et al., 2023, p.3).

To do this, Karaev et al. (2023) describes CoTracker as a transformer network with a CNN-based foundation to improve the initially estimated tracks $\Psi : G \mapsto O$. The tracks are encoded as a grid of input tokens G_t^i, one for each track $i = 1,...,N$ and time $t = 1,...,N$. The updated tracks come as a grid of output tokens O_t^i. These grids contain:

Image features: d-dimensional appearance features $\phi(I_t) \in \mathbb{R}^{d \times \frac{H}{k} \times \frac{W}{k}}$ are extracted from each frame using a CNN that is trained end-to-end with the transformer. The resolution of the video is reduced by $k = 4$ for computational efficiency and the downscaled features are obtained by applying average pooling to the base features.

Track features: Each track is assigned a feature vector to capture its appearance. The vectors are time-dependent to catch changes in the track appearance. Each track is initialized by sampling image features from the starting location and is then updated by the neural network.

Correlation features: The correlation features C_t^i are obtained by comparing track features Q_t^i with image features $\phi_s(I_t)$ surrounding the estimated current location $\hat{P}_t^i$ of the tracks. This is done to assist in matching the track to corresponding points in the video frames. The image features $\phi_s(I_t)$ are sampled at non-integer locations using bilinear interpolation and border padding.

The input tokens $G(\hat{P}, \hat{v}, Q)$ code for position, visibility, appearance, and correlation of the tracks and is given by the following concatenation of features:

$$G_t^i = (\hat{P}_t^i - \hat{P}_1^i, \hat{v}_t^i, Q_t^i, C_t^i, \eta(\hat{P}_t^i - \hat{P}_1^i)) + \eta'(\hat{P}_1^i) + \eta'(t) \tag{2.2}$$

η is the sinusoidal positional encoding of the track location with respect to the initial

location at time $t = 1$. The encoding η' is added of the start location P_1^i for time t with appropriate dimensions. The output tokens $O(\Delta\hat{P}, \Delta Q)$ contain updates on location and appearance as $O_t^i = (\Delta\hat{P}_t^i, \Delta Q_t^i)$ (Karaev et al., 2023).

The transformer is updated M times to progressively improve the track estimates, indexing the estimate with $m = 0,1...,M$ where $m = 0$ is the initialization. Each update computes $O(\Delta\hat{P}, \Delta Q) = \Psi\left(G(\hat{P}^{(m)}, \hat{v}^{(0)}, Q^{(m)})\right)$ and sets $\hat{P}^{(m+1)} = \hat{P}^{(m)} + \Delta\hat{P}$ and $Q^{(m+1)} = Q^{(m)} + \Delta Q$. The visibility mask is not updated by the transformer but is updated once at the final iteration based on the end sigmoid activation function. The initial values for the track features, position, and visibility are derived from the starting location and the time of the track (Karaev et al., 2023).

CoTracker uses a sliding window approach for predicting trajectories. A window fits several frames and makes it easier to predict through longer videos because CoTracker divides the video into overlapping windows where the output of one video serves as input to the next window. The tracking process is iterative so that the transformer is applied multiple times to each window, allowing for prediction over a longer duration (Karaev et al., 2023). During this experiment, we used the default setting of eight frames per window.

CoTracker is trained with 32 V100 32GB GPUs on the TAP-Vid-Kubrick dataset which was created with the Kubric engine consisting of videos with 3D-objects falling and bouncing on the ground under gravity (Karaev et al., 2023). During training, Cotracker had 256 randomly selected points for each sequence that were visible either in the first or middle frames. Each sequence consisted of 24 frames using sliding windows of size eight frames, iterated 50,000 times. During training, Karaev et al. (2023) applied two loss functions to handle overlapping windows. The main one is a track regression loss, summed over iterated transformer applications and windows to measure the difference between the predicted and ground truth positions of points:

$$\mathcal{L}_1(\hat{P}, P) = \sum_{j=1}^{J} \sum_{m=1}^{M} \gamma^{M-m} \|\hat{P}^{(m,j)} - P^{(j)}\|, \tag{2.3}$$

with $\gamma = 0.8$ to discount early transformer updates, $P^{(j)}$ contains the ground truth trajectories restricted to window j. The second loss function is a cross entropy of the visibility flags:

$$\mathcal{L}_2(\hat{v}, v) = \sum_{j=1}^{J} \mathrm{CE}(\hat{v}^{(M,j)}, v^{(j)}) \tag{2.4}$$

Only a moderate number of windows are used in the loss during training due to computational cost, but according to Karaev et al. (2023), in principle, the model should be able to handle any video length.

While claiming the best results on the TAP-Vid DAVIS dataset, Karaev et al. (2023) states that CoTracker makes mistakes easily avoided by humans. Also, it is limited

by processing relatively short windows of points at a time and therefore can fail points that stay occluded for the length of multiple sliding windows, as well as fail to track points over thousands of frames.

2.2.3 YOLOv5: You Only Look Once version 5

YOLO was the first single-stage object detection algorithm, originally designed for object detection in videos which made detection speed one of its key features (Jocher et al., 2020). The original model has undergone rapid development and Jocher et al. (2020) deployed the fifth iteration, YOLOv5, achieving state-of-the-art precision accuracy of nearly 50 mean average precision (mAP) in the MS COCO dataset. MS COCO is a dataset consisting of 164,000 images of 80 object categories of common objects (T.-Y. Lin et al., 2014).

The mAP is a metric used to measure the performance of a model that focuses on object detection tasks and information retrieval on images. This is done by comparing the ground truth bounding boxes to the predicted bounding boxes and determining the precision and recall values that represent the model's ability to find true positives out of all positive predictions, and its ability to find true positives out of all predictions respectively. The intersection-over-union (IoU), which measures how much of the predicted bounding box overlaps with the ground truth bounding box, is derived for all predictions. Then the precision and recall values are plotted in decreasing order which, when plotted, form a curve across a 2D space. By integrating the area under the precision and recall curve, the average precision (AP) is obtained. The mean value of the AP, i.e., the mAP, is obtained when the IoU is varied from 0.5 to 0.95 in steps of 0.05 across the label categories (Pereira, 2022).

It should be noted, that newer versions of YOLO have been released since YOLOv5, e.g., YOLOv6 (Li et al., 2022), YOLOv7 (C.-Y. Wang et al., 2023), YOLOv8 (Jocher et al., 2023). From YOLOv5, the different models have fine-tuned the trade-off between speed and accuracy to suit specific applications and hardware requirements (Terven et al., 2023). The usage of YOLOv5 for this project can be justified by experiments such as Olorunshola et al. (2023) which compared YOLOv5 to YOLOv7 and found that YOLOv5 outperformed YOLOv7 in precision and accuracy, while the latter had slightly higher recall during testing. Terven et al. (2023) also found that models newer than YOLOv5 mainly had faster inference but in some cases less accuracy. Given that inference speed is not an issue for this project, and that YOLOv5 is widely used and more documented than the newer models (Olorunshola et al., 2023), it is a suitable object detector for this project.

In the following sections 2.2.3.1 and 2.2.3.2 we describe the basic functionality of how YOLO-models make predictions and the architecture of YOLOv5 that we apply for our experiments respectively.

2.2.3.1 Functionality

YOLO models object detection as a regression problem. As shown in figure 2.3, the method YOLO applies to detect objects is to divide each input frame or image into

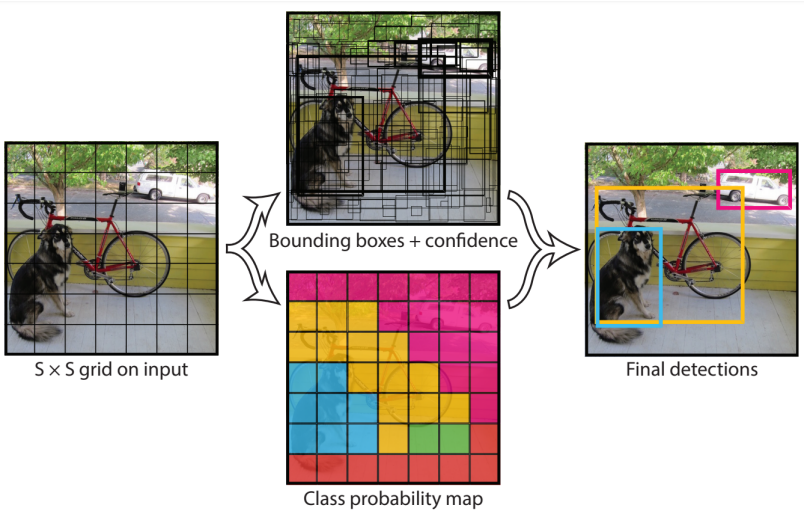

Figure 2.3: Visualization of how the predictions function. Firstly, YOLO divides the image or frame into a $S \times S$ grid. Each grid cell then predicts B bounding boxes, confidence for the bounding boxes, and C class probabilities (Redmon et al., 2016, p.2).

an $S \times S$ grid. Every grid cell predicts B bounding boxes with confidence scores indicating the certainty of whether the box contains an object or not, and how accurate the predicted box is. Each bounding box then consists of five predictions; *x, y, w, h,* and *confidence*. x and y are coordinates that represent the center of the boundary box, w and h are the width and height of the box relative to the whole image, and *confidence* is defined as $\Pr(\text{Object}) * \text{IoU}_{pred}^{truth}$. This confidence score should be zero when there is no object within the cell and equal to the IoU when there is an object. C class probabilities conditioned on the grid cell containing an object are also predicted and is defined as $\Pr(\text{Class}_i \mid \text{Object})$, where i signifies a specific class. Lastly, the conditional class probabilities are multiplied with each of the bounding box confidence predictions as:

$$Pr(Class_i \mid Object) * Pr(Object) * IoU_{pred}^{truth} = Pr(Class_i) * IoU_{pred}^{truth} \tag{2.5}$$

This gives confidence scores for specific classes in every bounding box. The encoding for these predictions are structured as a $S \times S \times (B * 5 + C)$ tensor (Redmon et al., 2016).

2.2.3.2 Architecture

YOLOv5 is divided into five models depending on the depth and width of the architecture, YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. Among them, the YOLOv5n has the fastest calculation speed, but the lowest average precision, whereas the YOLOv5x has the opposite characteristics (Z. Chen et al., 2022).

As shown in figure 2.4, YOLOv5 consists of three components: *the backbone*, *neck*, and *head* (Z. Chen et al., 2022). The backbone consists of CSPDarknet53 that has a strided convolution layer with a large window size. This layer is followed by convolutional a layer that, after inputting an image, aggregates and forms image features on different image granularities (Z. Chen et al., 2022; Terven et al., 2023).

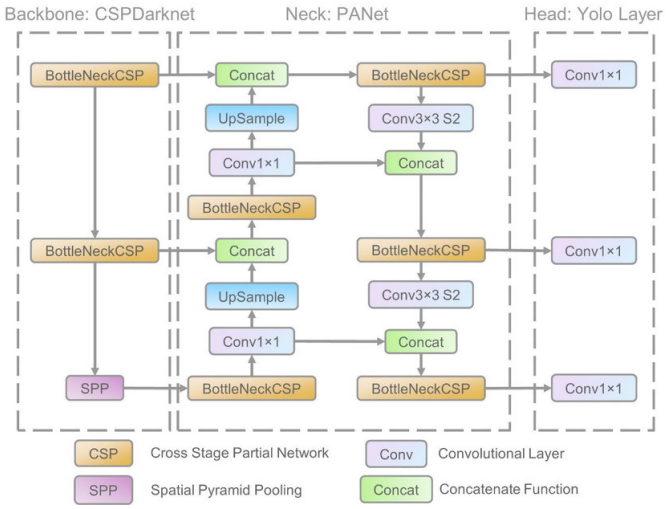

Figure 2.4: Visualization of the YOLOv5-architecture. The data is inputed to the backbone first for feature extraction and then the data is fed to the neck for stitching together features at various scales. Lastly, the detection results are used as output in the head (Xu et al., 2021, p.5).

The neck, using a Spatial Pyramid Pooling Fast (SPPF) layer and following convolutional layers, processes features at various scales and increase the resolution of feature maps (Terven et al., 2023). The neck then stitches the image features and transmits them to the prediction layer. The neck allows the model to recognize the same object in different scales and sizes on unobserved data (Nazir & Wani, 2023).

The head predicts the image features to generate bounding boxes and predicted categories (Z. Chen et al., 2022). It typically consists of one or more task-specific subnetworks that perform classification, localization, instance segmentation, and pose estimation. By processing the features that the neck provides, it generates predictions for each object candidate. Lastly, in post-processing, it uses non-maximum suppression to filter out overlapping neighboring predictions and extracts the most confident detections (Terven et al., 2023). However, Redmon et al. (2016) states that NMS is not critical for the performance of the model, as it is for models with a region proposal-based framework such as a deformable part-based model, non-maximum suppression adds 2-3% in mAP.

The YOLOv5 network uses Generalized Intersection over Union ($GIOU$) as the network loss function as:

$$GIOU = IOU - \frac{|C - (A \cup B)|}{|C|} \tag{2.6}$$

A, B represents two arbitrary boxes, C represents the smallest convex box enclosing both A and B, $IOU = |A \cap B|/|A \cup B|$, and $|\cdot|$ represents the number of pixels. When the input network predicts image features, the optimal target frame is filtered by combining the loss function $GIOU$ and the non-maximum suppression algorithm (Z. Chen et al., 2022).

YOLOv5 is trained in two steps. The convolutional layers are pre-trained on the Imagenet 1000-class competition dataset (Russakovsky et al., 2015), afterwards, the

model is trained and validated on the PASCAL VOC 2007 (Everingham et al., 2007) and 2012 (Everingham et al., 2012) datasets. YOLOv5 is trained for 135 epochs, using a batch size of 64, a momentum of 0.9, and a decay of 0.0005. To avoid overfitting, dropout and data augmentation were applied as stated by Redmon et al. (2016). Even though YOLOv5 generally has good accuracy, it sometimes has difficulties detecting small objects (Benjumea et al., 2021).

3

Methods

3.1 Data

Doersch et al. (2022) created the TAP-Vid dataset which consists of four distinct datasets. One of which, TAP-Vid-DAVIS, was done by annotating the trajectories of key-points on specific parts of objects in 30 videos from the DAVIS 2017 validation set (Pont-Tuset et al., 2017), allowing for the evaluation of predicted trajectories by the point trackers. For each video, up to five objects and five points per object were annotated. The point tracks were annotated at 1080p resolution and the whole dataset was resized to 256x256 (Doersch et al., 2022). This dataset was used to replicate experiments done by Karaev et al. (2023) and Doersch et al. (2023) as in section 4.1.

Furthermore, a custom dataset was generated with our collaborator. This consists of a Rubik's Cube as a base object, and the objects of interest, i.e. key-points to detect and track, were made up of stickers in varying geometrical shapes and colors placed on different cube squares. The number of stickers ranges from three to five with alternating placements to have different backgrounds. Depending on the camera angle, the number of visible stickers ranges between two and five, sometimes visible throughout the whole video, and sometimes not. The number of stickers that are visible throughout the video varies due to the cube's rotation and the stickers disappear and reappear in relation to the camera. There are six stickers: *Big red circle, small blue circle, small orange circle, small gray circle, small star*, and *medium star*. Both of the stars are reflective gold in color. The size indications are relative, whether the sticker is big, medium, or small it still fits within a square of the Rubik's cube. Each object of interest was annotated as described in section 3.1.1.

Using stickers on a Rubik's cube allows for easy manipulation where it is possible to change the background and location of the objects of interest. For the dataset, several videos were recorded where the cube is rotating 360 degrees horizontally and is captured by cameras at four angles: 0, 30, 60, and 90 degrees. Zero degrees means parallel to the ground and 90 degrees means perpendicular to the ground. Some videos contain an externally occluding object, and in two videos, the cube slides across the frame without rotation.

The final test dataset used a mixture of these videos to include different sticker

setups from multiple angles to get a diverse dataset. In the end, the final test dataset consisted of 18 videos with a length of approximately three seconds each. Eleven videos of the cube rotating in one direction where five of them had an external occluding object, five videos of the cube rotating in two directions back-and-forth, and two videos where the cube slides across the frame, one of which had an external occlusion. In the rotating videos, the externally occluding object was a strip of tape temporarily blocking the view of every sticker. In the sliding video, the cube was blocked by a screwdriver. See figure 3.1 for an example of the videos without occlusions.

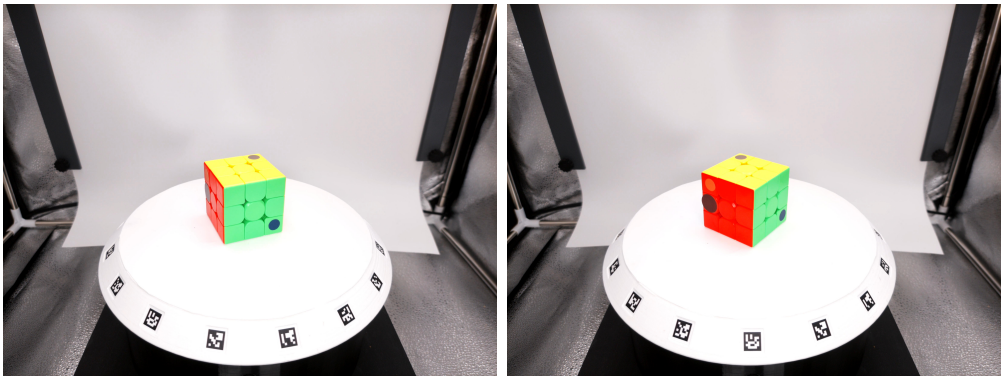

Figure 3.1: Two frames in a 30-degree angle from the generated dataset used in our study. The cube has stickers on different sides marking the areas of interest which the models track.

This dataset also has a separate training dataset with images used for training YOLOv5. This training dataset consists of 649 annotated images of the cube rotating without occluding objects and with varying sticker setups. None of these sticker setups or images was used for the final testing datasets.

3.1.1 Annotation

Annotation of video data filmed with our collaborator was made in the online software Roboflow (Dwyer et al., 2022). All videos were split into images to represent 15 frames per second. For each frame, every object of interest was annotated manually using a rectangular bounding box fitted tightly around the object of interest.

The annotations are provided in two ways that allow for evaluation comparable between the point trackers and the object detector. Firstly, the bounding boxes around the objects of interest are fitted such that the whole object is annotated with a minimum amount of background. For instance, a bounding box of a circle marks the complete area of the circle with the edges matching the circle's highest, lowest, leftmost, and rightmost points but some background is included in the corners of the bounding box. Secondly, the bounding box centers were used as the ground-truth key-point trajectory to evaluate the point trackers. We also used the point tracker's predicted coordinates for every frame as a center point from which we computed a bounding box. The bounding box dimensions, like the query points used as input for the point trackers, were extracted from the ground truth of the first frame in which the object appeared. These dimensions were used to compute the bounding boxes of every predicted coordinate pair made by the point trackers.

Utilizing this method provides two forms of data for comparing the different models. The point trajectory extracted from the predicted bounding boxes made by the object detector is comparable to the output of the point trackers. The bounding boxes estimated from the point trackers' predicted coordinates are comparable to the output of the object detector. The metrics of evaluation are explained in section 3.3.

3.1.2 Queries

The input queries given to the point trackers are the center points of the ground truth annotation from the first frame in which the object-of-interest appears. The centers of the stickers are the key-points which the models are supposed to track. The input for CoTracker was a tensor with shape (n, 3). n is the number of queried key-points to track consisting of three elements (t, x, y). t is the frame number in which the key-point first appears while x and y are the coordinates where the key-point is located in the corresponding frame.

The input queries for TAPIR was collected in the same way as for CoTracker but as an array with shape (n, 3). n signifies the number of points to track consisting of the same three elements as for CoTracker but in a different order (t, y, x). t is the frame number and (y, x) are the coordinates.

3.2 Training

YOLOv5 was trained on our custom dataset of Rubik's Cube images until convergence using an NVIDIA GeForce RTX 3050 4GB GPU, running for approximately 11 hours over 150 epochs. All default training hyperparameters were used except for batch size which was set to 2 instead of 16 for computational reasons, which according to Keskar et al. (2016) might have a positive effect in making the model more generalizable. The training was initialized with the YOLOv5s, i.e., the second smallest architecture, for faster training which is pre-trained on Imagenet, PASCAL VOC 2007 and 2012 (Redmon et al., 2016). To make the model more robust considering our relatively small dataset, augmentation such as blurring, gray-scaling, and cropping was applied as per the default settings when training the model (Jocher et al., 2020). The training dataset comprised 454 training images, 131 validation images, and 64 test images of a horizontally rotating Rubik's Cube with multiple sticker setups from four different camera angles as described in section 3.1, without any external occluding objects. None of these images or sticker setups were used for the final testing of the models. The training of YOLOv5 was done using Python 3 (Van Rossum & Drake, 2009).

For CoTracker and TAPIR, no custom training of the models was executed and pre-trained models were used. The reasoning for this was that both Karaev et al. (2023) and Doersch et al. (2023) claim that the models can track any point on a physical surface in a video without additional training. Thus, to simplify the video annotation process compared to e.g., object detectors, both CoTracker and TAPIR

were used without any additional training or modification. See sections 2.2.1 and 2.2.2 for an explanation of how these models were trained.

3.3 Evaluation Metrics

To compare the performance of the trackers and the object detector, we used two common metrics used for point tracking benchmarks, *precision plot* and *success plot* (Wu et al., 2013; Henriques et al., 2012; Brdjanin et al., 2020). The average precision or success rate is obtained by executing a point tracker on a video sequence, where the starting key-point is selected from the ground truth boundary box center in the first frame (Brdjanin et al., 2020). The precision plot shows *how well* the tracker found the object, while the success plot shows *whether the tracker found the required object* in the frame (Brdjanin et al., 2020).

To measure the overall performance of the models in terms of precision- and success plots, the *Area Under Curve* (AUC) was used as proposed by Wu et al. (2013), who states that using a single threshold for tracker evaluation might not be fair or representative. Therefore, the AUC is derived for each plot and model to rank the performance of the models. The AUC measures the entire two-dimensional area underneath the precision and success curve where a higher AUC represents better performance. In the context of precision and success plots, a model with a higher AUC has more frames where the object of interest is within the given thresholds for Euclidean distance or fulfills the intersection over the union threshold.

Unlike the point trackers, YOLOv5 categorizes the objects it detects. For this project, the classification accuracy of YOLOv5, i.e., if it classifies a red circle as a blue circle, was considered irrelevant since the point trackers do not classify the object of interest. Instead, we only focused on whether it identifies objects of interest and the spatial location of these objects to compare with the ground truth.

Precision plot shows the percentage of frames that the tracker successfully followed the object of interest for a range of threshold distances. We use the same threshold distances as the ones used by Doersch et al. (2023) and Karaev et al. (2023), namely: 1, 2, 4, 8, and 16 pixels. These threshold distances are set as the standard based on the TAP-Vid metrics by Doersch et al. (2022). The Euclidean distance between the tracker center point and the ground truth center point is measured and a frame is deemed successful if the distance is less than or equal to the threshold. The precision is defined as:

$$precision = \frac{N_t}{N_{frames}} \tag{3.1}$$

N_t denotes the number of successfully tracked frames that the center location error is less than threshold t (e.g., 30 pixels). N_{frames} denotes the number of total frames in a sequence (F. Chen et al., 2022).

This method is preferred to the average center location error (CLE) which is the pixel-wise average Euclidean distance between the predicted center point and the ground truth center point. A precision plot indicates the performance more representatively

because if a tracker tracks an object closely for most of the video, but loses track completely in the last frames, the CLE might be higher than a tracker that sticks to the object throughout the video but not as closely (Babenko et al., 2011; F. Chen et al., 2022).

Success plot measures the bounding box overlap. Given a predicted bounding box r_t and the ground truth bounding box r_g, the overlap score is defined as the intersection-over-union (IoU) between r_t and r_g as:

$$IoU = \frac{|r_t \cap r_g|}{|r_t \cup r_g|} \tag{3.2}$$

$|\cdot|$ represents the number of pixels, and $\cap$ and $\cup$ represent intersection and union operators. The overlap score of IoU is calculated for each frame and measures the model's success while tracking an object. If the IoU is higher than threshold t, the model successfully tracked the object. The thresholds used were from 0 to 100 percent overlap with a 20 percent increment as reported by Brdjanin et al. (2020).

Then the average overlap measure based on s can be defined as the mean of IoU among the whole image sequence. So, given a threshold A, the success of a sequence can be computed as:

$$R_A = \frac{N_A}{N_G} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_A(s_i) \tag{3.3}$$

N_A is the number of frames accurately annotated within threshold A and N_G is the total number of frames in a frame sequence or video. s_i is the IoU between the tracking result of the ith frame and the ground truth bounding box. $\mathbb{1}_A(s_i)$ is an indication function which got 1 if $s_i > A$ and 0 otherwise, and R is the success rate (F. Chen et al., 2022). We plot the number of frames within thresholds ranging from 0 to 1 and compute an AUC to examine which of the three models had the most successful predictions above the various thresholds. The model with the highest AUC is then determined as the one with the highest performance with regards to IoU.

TAP-vid metrics

For evaluating the accuracy of a point tracker, Doersch et al. (2022) proposed three evaluation metrics that range between 0 and 1, multiplied by 100 for interpretability. The closer the metric is to 100, the better the performance:

1. *Occlusion Accuracy (OA)*, a binary classification accuracy for the point occlusion prediction on each frame defined as:

$$OA = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

   Positive (P) and negative (N) indicate if a point is occluded or visible respectively. T and N indicate if the predicted value is in accordance with the ground-truth. Thus, OA is the number of correctly predicted values divided by the total number of predictions.

2. $< \delta^x$ evaluates the position accuracy on frames where the point is visible in ground truth. For a threshold of δ (1, 2, 4, 8, and 16 pixels as reported by Doersch et al. (2022)) it measures the fraction of points within that distance threshold, measured in pixels, compared to their ground truth.

$$u_t = \begin{cases} 1 & \text{if } d(\hat{p}_t, p_t) < \delta \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

   u_t is the binary classification defining if a predicted point's position, $\hat{p}_t$, in frame t is less than the distance threshold δ as computed by the Euclidian distance d to the point's ground truth position, p_t. u_t is 1 if $d(\hat{p}_t,\ p_t)$ is less than δ and 0 if not.

3. *Jaccard at* δ evaluates both occlusion and position accuracy as per equation 3.6.

$$J = \frac{TP}{TP + FP + FN} \tag{3.6}$$

   This is the fraction of true positives (points within the threshold of any visible ground truth points), divided by true positives plus false positives (points predicted visible but the ground truth is either occluded or farther away than the threshold), plus false negatives (ground truth visible points that are predicted as occluded or predicted farther away than the threshold). *Average Jaccard* (AJ) averages Jaccard across the same thresholds as $< \delta^x_{avg}$ as proposed by Doersch et al. (2022).

To evaluate the accuracy of YOLOv5, the center point of the predicted bounding box of each identified object is compared to all ground truth center points in the same frame. The predicted bounding box is assumed to correspond to the ground truth bounding box with the shortest Euclidean distance between the predicted center point and the ground truth center point. Once all the predicted bounding boxes have been matched to the closest ground truth bounding boxes, the AJ and $< \delta^x$ of the model is evaluated.

3.4 Qualitative Analysis

Besides the quantitative analysis of the performance of the different models, we performed a qualitative ad-hoc analysis of where the point trackers fail based on the properties of the videos where they make their predictions. We examined the five videos for each model where the model performed best and worst based on AJ, $< \delta^x_{avg}$, and OA for both of the datasets. Examining the videos allowed for discovering potential differences in these image sequences.

Following the video analysis methods proposed by Fazeli et al. (2023), given that all videos lack oral communication and are not focused on the interaction between agents, the unit choice we focused on was visual features (scene, frame or camera shot) and common threads (similar patterns, events, or ideas) in the videos. Therefore an inductive content analysis was performed without a pre-established data

classification matrix since we did not know beforehand what properties of the analyzed videos had and how those properties affected the prediction by the models. Inspiration also comes from the Video Data Analysis framework by Nassauer and Legewie (2021), who looked at the depicted in a video, i.e., "the raw content of a visual, the objects, people, surroundings, and so on" (p.139).

However, some properties considered beforehand were based on the reported weaknesses of the models. As discussed in section 2.2, TAPIR reportedly struggles in discerning the boundaries between the background and foreground, and when the object of interest undergoes large appearance changes. CoTracker can fail with objects of interest that stay occluded for the length of multiple sliding windows, as well as fail to track points over thousands of frames. Since all videos are about three seconds long and the maximum number of videos adds up to 20 for each dataset, all the worst and best-performing videos were examined.

4

Experiments

4.1 Replication of TAP-Vid DAVIS Results

Figure 4.1: The predicted outputs of CoTracker and TAPIR on the video 'car-shadow' from the TAP-Vid-DAVIS dataset.

Both CoTracker and TAPIR used TAP-Vid-DAVIS (Doersch et al., 2022) for benchmarking (Karaev et al., 2023; Doersch et al., 2023), but given that both models were published recently, little research has been done on replicating these results. Therefore, we ran both models on the same dataset and assessed tracking accuracy using the TAP-Vid metrics (Doersch et al., 2022).

We used the pre-trained models, so no adjustments were made to the models, i.e., they were trained as described in section 2.2 and evaluated on the same metrics as proposed by Doersch et al. (2022) i.e., OA, $< \delta^x$, and AJ. In their papers, both CoTracker and TAPIR evaluated accuracy on both *first query*, where the model is initialized only on the first frame, and *strided*, where the model is fed the ground truth points every fifth frame and predicts the track between them (Karaev et al., 2023; Doersch et al., 2023). For this project, we only reproduced the case of the first query because it is the same method to which we apply the models on our custom dataset.

For each video, a set of query points, *(t, x, y)*, are given to track where t is the frame number where the key-point first appears, *(x, y)* are the coordinates. The predicted tracks and occlusion status are then compared to the ground truth provided by the dataset to evaluate the accuracy.

As seen in table 4.1, our AJ, OA, and $< \delta^x_{avg}$ are lower than the ones reported

Model	DAVIS		
	AJ	$< \delta^x_{avg}$	OA
MFT (Neoral et al., 2024)	47.3	66.8	77.8
PIPs (Harley et al., 2022)	42.2	64.8	77.7
CoTracker (Original) (Karaev et al., 2023)	62.2	75.7	89.3
TAPIR (Original) (Doersch et al., 2023)	56.2	70.0	86.5
CoTracker (Ours)	**58.1**	**72.4**	**88.4**
TAPIR (Ours)	**53.3**	**65.9**	**83.9**

Table 4.1: Comparison of our results from CoTracker and TAPIR to the published results, and two other state-of-the-art models on TAP-Vid DAVIS. $< \delta^x_{avg}$ is the fraction of points non-occluded in the ground truth and prediction is within a threshold, ignoring occlusion estimate; OA is the accuracy of predicting occlusion. AJ is the Average Jaccard, which considers both position and occlusion accuracy, a high AJ indicates a highly accurate prediction with respect to the ground-truth (Doersch et al., 2023).

in their respective papers (Karaev et al., 2023; Doersch et al., 2023). Our results from CoTracker had higher accuracy on all metrics than the other models, while the results from TAPIR had higher accuracy on all metrics except for $< \delta^x_{avg}$ compared to MFT.

4.1.1 TAP-Vid DAVIS Videos with worst performance results

Table 4.2 presents the average results from the five videos in which CoTracker and TAPIR performed worst and best respectively. Both models shared top and bottom videos with only one exception, indicating that key-points on objects in some videos are inherently easier or more difficult to track and that both models appear to have similar issues with their capability, even though CoTracker on average performs better as seen in table 4.1.

	Worst			Best		
Model	AJ	$< \delta^x_{avg}$	OA	AJ	$< \delta^x_{avg}$	OA
TAPIR	31.3	47.7	70.2	78.5	86.6	97.0
CoTracker	29.1	50.7	68.0	82.9	91.3	97.9

Table 4.2: AJ, $< \delta^x_{avg}$, and OA means computed from the videos with the five worst and five best performances of the models on TAP-Vid DAVIS.

Common themes and visual features (Fazeli et al., 2023) in the best-performing videos are almost static camera, minimal rotation, no occlusion, non-human agents, slow object movement, and multiple query points on background objects. Common themes for the worst-performing videos have the opposite characteristics, i.e., panning camera, object rotation, partial occlusion, the main object moving fast, and most of the query points are located on the main object.

More specifically, common denominators of the worst-performing videos are that the object moves quickly with either a rotation change or partial occlusion. This aligns with the statement of Doersch et al. (2023) who mentioned that TAPIR has difficulties with large appearance changes, i.e., when the object rotates horizontally or is out of frame for some time. All of these videos had the camera panning and following the main object which was moving relatively fast across the scene. Furthermore,

due to the camera movement, the background also went through changes, and all but one was filmed in an urban environment with lots of appearance changes indicating that the scene affects how well the point trackers perform. The light in the videos was constant, but with a varying background there were also color changes in the frame, and because both CoTracker and TAPIR compare query features to all features in each frame (Karaev et al., 2023; Doersch et al., 2023) this indicate a risk of finding suboptimal matches for the query features as the background changes.

The best-performing videos, on the other hand, have objects that are fully visible throughout the clip and move linearly, making them easier to track. All of these videos had an almost static camera angle and only had the main object move across the frame quite slowly compared to the worst-performing videos. Furthermore, all but one video had almost no rotation of the main object, and the one with rotation had the camera following as the object of interest drove past, i.e., the camera angle changed but the object itself did not rotate. Following this, it also matters which points were tracked, from examining the videos, it becomes apparent that videos with several query points on static background objects are easier to track than query points on the object in focus.

4.2 Rotating Rubik's Cube Dataset

The testing and evaluation on the custom dataset were executed with the same method as for the TAP-Vid DAVIS dataset. CoTracker and TAPIR were given a set of query points *(t, x, y)* corresponding to the number of objects to track, unlike YOLOv5, which was given the video to detect objects it had been trained on. For each video, we measured AJ, $< \delta^x_{avg}$, and OA. From the predicted trajectories, we also derive precision within a set of distance thresholds and success rates with regard to IoU thresholds.

CoTracker, TAPIR, and YOLOv5 were used on 18 videos from the Rotating Rubik's cube dataset with eight sticker constellations and four angles. Six of the videos had an obstacle temporarily blocking the view of the objects of interest. Out of the 18 videos, 16 videos had a rotating cube, and two videos where the cube was sliding horizontally across the frame. No additional training was done on CoTracker or TAPIR. YOLOv5 was trained as explained in section 3.2. For the point tracker models, each query point was initialized on the object of interest when it was first visible in the video.

The dataset was further split up into two groups. One split, the not occluded videos, contains videos without any external obstacles to block the view. The second split, occluded videos, contains videos with external obstacles blocking the visibility of the objects of interest temporarily. The performance of these videos is shown in table 4.4.

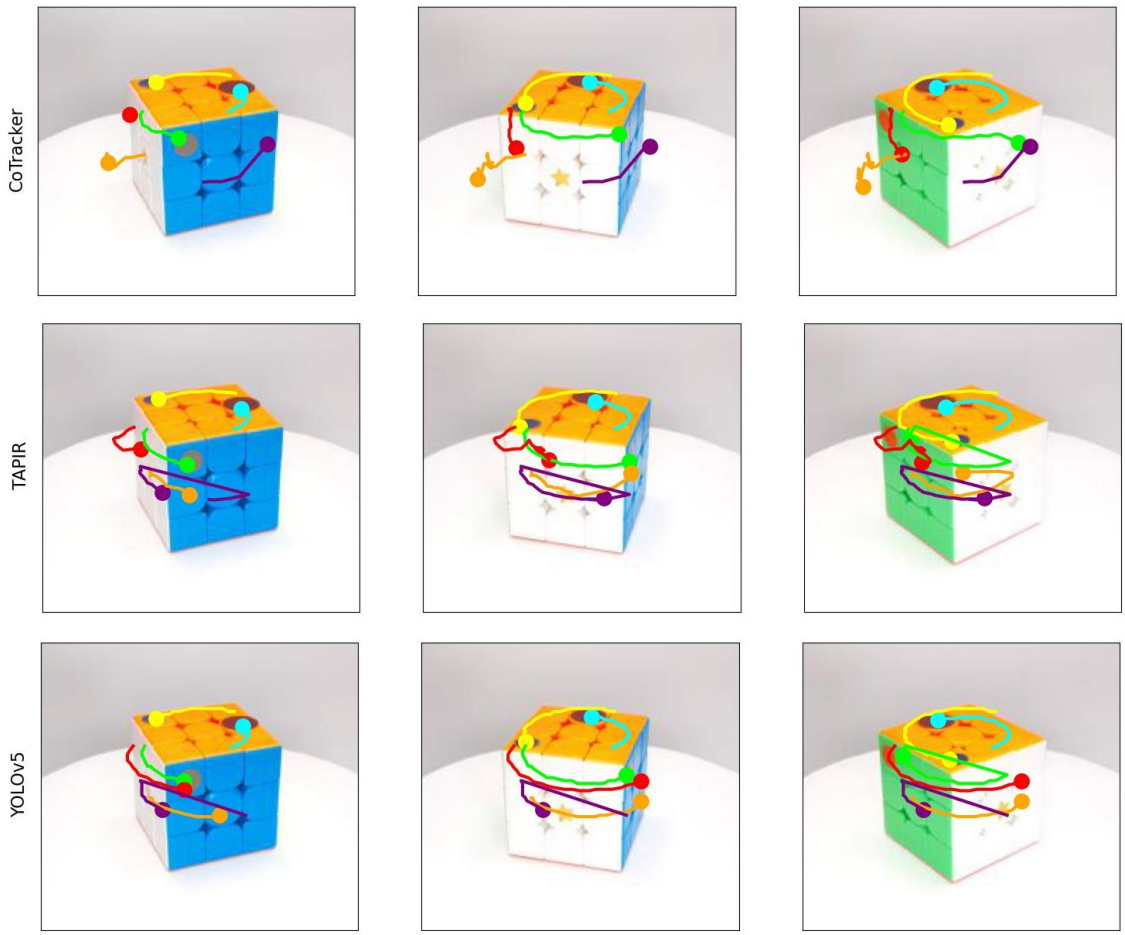

Figure 4.2: Three of the same frames from the three different models output. The lines show the predicted trajectories of the various objects of interest, and the colors of the lines indicate the different shapes being tracked. For instance, the cyan colored line follows the red circle on the top and the orange line follows the star shape on the white side.

4.2.1 Quantitative results

As seen in tables 4.3 and 4.4, YOLOv5 had the highest accuracy in regards to AJ, $< \delta^x_{avg}$, and OA. The second best was TAPIR while CoTracker had the worst performance. This holds when combining all data, and videos with occluding objects. For videos without occluding objects, YOLOv5 had the highest AJ and OA, while Cotracker had the highest $< \delta^x_{avg}$.

For reference, figure 4.2, shows the predicted output of the three models. Two of the lines, cyan and yellow, follow the objects of interest quite well as the objects being tracked are constantly visible throughout the video. However, as the cube rotates the models fail to track the objects becoming occluded or the objects which appear in later frames. This demonstrates the difference in accuracy as the predicted trajectories from YOLOv5 are of a more circular motion compared to TAPIR and CoTracker, whose trajectories do not follow stickers placed on the side of the cube which are sometimes occluded.

From the precision- and success plots in figures 4.3 and 4.4 showing the model accuracy on all data, we can see that YOLOv5 performed best, TAPIR second, and

Model	All Data		
	AJ	$< \delta^x_{avg}$	OA
YOLOv5	42.4	51.0	90.0
CoTracker	32.8	44.5	78.5
TAPIR	35.3	48.4	84.9

Table 4.3: Comparison of TAP-Vid metrics for CoTracker, TAPIR, and YOLOv5 on Rotating Rubik's Cube dataset

	Not Occluded			Occluded		
Model	AJ	$< \delta^x_{avg}$	OA	AJ	$< \delta^x_{avg}$	OA
YOLOv5	46.1	54.5	92.5	34.9	44.1	85.2
CoTracker	43.6	59.2	86.5	17.0	27.2	60.8
TAPIR	40.5	53.2	88.8	25.0	38.9	77.3

Table 4.4: Comparison of TAP-Vid metrics for CoTracker, TAPIR, and YOLOv5 for the divided dataset. Not occluded indicates videos without temporary visual obstacles. Occluded are videos with temporary visual obstacles.

CoTracker worst. When dividing the data between videos with and without external occlusion, the ranking of the models remains the same. We also calculated the AUC for each precision and success plot, both the combined data and divided between external occlusion or not, and each model as is seen in table 4.5. These results also show that YOLOv5 has a higher accuracy than CoTracker and TAPIR given that a higher AUC score indicates a larger area and consequently more data points predicted within the various distance and IoU thresholds.

As seen in tables 4.5 and 4.4, all models had worse accuracy on all metrics when the video contained externally occluding objects. CoTracker had the highest performance gap between the non-obstructed and the obstructed video with a 26.6-point difference in AJ while TAPIR differed by 15.5 points and YOLOv5 11.2, indicating that CoTracker is the most sensitive to occlusions.

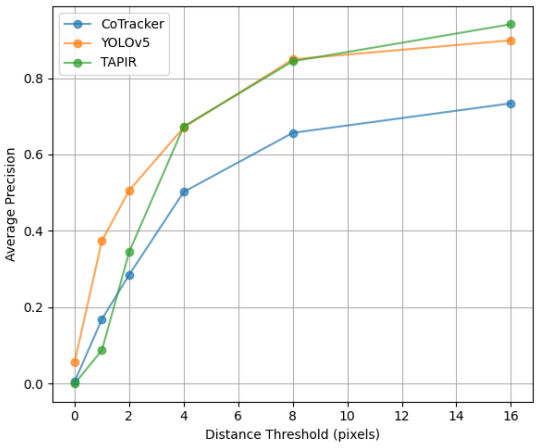

Figure 4.3: Precision plot for all Rubik's Cube videos.

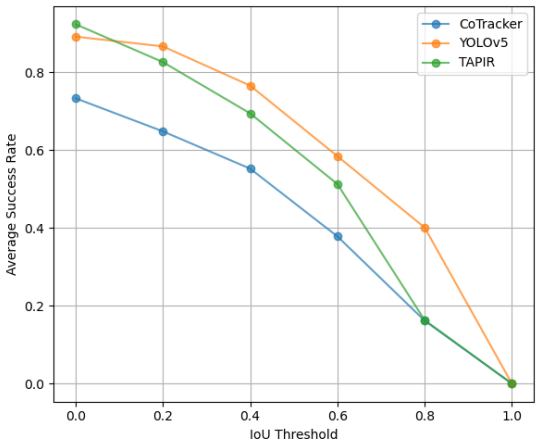

Figure 4.4: Success plot for all Rubik's Cube videos.

From figure 4.3 we see that YOLOv5 has a higher average precision rate than CoTracker and TAPIR except for the thresholds of four and eight pixels where YOLOv5 and TAPIR have almost the same precision rate. Between 8 and 16 pixels, TAPIR has slightly higher precision than YOLOv5 which is the only instance where YOLOv5

	Precision plot AUC			Success plot AUC		
Model	All Data	Not Occluded	Occluded	All Data	Not Occluded	Occluded
YOLOv5	11.86	12.09	11.49	6.62	6.21	7.32
TAPIR	11.45	11.80	11.24	4.85	4.65	4.97
CoTracker	8.98	9.30	8.46	3.95	3.99	3.89

Table 4.5: Area Under Curve results for each model on the combined Rubik's cube dataset, and divided between with and without external occlusion.

is not the highest-scoring model part from figure 4.4 where TAPIR has a higher average success rate when the IoU threshold is below approximately 0.1. Table 4.5 shows that YOLOv5 scored the highest in AUC for both precision and success rate for the whole dataset as well as for the two splits.

When examining the accuracy of each shape of the stickers, combined precision and success plots of all videos show that all models had the highest precision and success rate for the big red circle and the lowest accuracy for the medium-sized star. For the remaining shapes, there were no clear patterns of which shape was more or less successfully tracked.

4.2.2 Qualitative results for CoTracker and TAPIR

Common themes and visual features (Fazeli et al., 2023) in the worst-performing videos were as mentioned videos with external occlusion where all stickers were blocked temporarily and videos recorded at 0, 30, or 60 degrees where the stickers rotated out of the view. Looking at a more granular level in videos without external occlusion, it was the placement of the object in relation to the camera that was the determining factor rather than the properties of the video, e.g., stickers placed on the side that disappeared behind the cube's axis, was the biggest factor in performance. This seems natural since the point trackers estimate where the object-of-interest will appear again which is more difficult than keeping track of a fully visible object (Doersch et al., 2023; Karaev et al., 2023).

As stated above, which sticker was used also impacted the result as the medium-sized star had the worst accuracy, likely due to it having a more reflective color than the circle-shaped stickers. This became extra apparent when it was placed on a white background and even when manually inspecting the videos, in some angles, it was difficult to locate the star even though it should be visible. Given that both Cotracker and TAPIR compare query features to all features in each frame independently (Doersch et al., 2023; Karaev et al., 2023), this indicates that videos with light changes cause the query features to be more similar to its surrounding features for some frames can affect the performance negatively, meaning that objects in videos with contrasting colors between background and foreground can be easier to track.

The best-performing videos consisted of videos without external occlusion and of videos with a camera angle of 90 degrees, i.e., recorded from straight above, which is logical in that all stickers were fully visible throughout the video. In videos recorded from the side, this pattern was consistent since the objects on top of the cube yielded the best result, while the disappearing objects were harder to track.

5

Discussion

5.1 Rotating Rubik's Cube Dataset

From running CoTracker, TAPIR, and YOLOv5 on the custom dataset of a Rubik's cube, we can conclude that YOLOv5 outperformed both point tracker models on all metrics except $< \delta^x_{avg}$ on videos without obstacles where CoTracker achieved the highest result. The results of CoTracker and TAPIR reflect the outcome of the experiment on the TAP-Vid DAVIS dataset, where they performed worse on videos with a lot of occlusion. This is also seen in the YOLOv5 outcome as it also had significantly lower scores on the occluded videos compared to the non-occluded ones. Most videos are partially occluded due to the cube rotating around its axis. However, all models performed worse when there was an external occlusion. When the object of interest disappears behind an external object, CoTracker and TAPIR tend to lose track completely, and either move along the obstacle or do not find the object-of-interest again once it reappears. YOLOv5 did not have this issue since it detects objects for each frame. This issue is not as prevalent when the object of interest disappears behind its axis which might be due to the trajectory being relatively static vertically. Since YOLOv5 does not consider trajectory or predict future locations, the results indicate an advantage for semi-automatic annotation to detect objects rather than predict where they will occur in the subsequent frames. However, no changes of hyperparameters were made to CoTracker or TAPIR, had we experimented with these, they might have achieved a higher accuracy.

The different types of predictions between point trackers and object detectors might affect the result because the Cotracker and TAPIR do not generate bounding boxes while YOLOv5 does. For the point trackers, the bounding boxes were estimated based on the ground truth annotation, i.e., the size of the bounding box for all subsequent frames was determined based on its size on the first frame each object appears. Thus, for an object appearing from the side, its first bounding box will have a smaller area compared to when the object is viewed right in front of the camera. For these cases, the IoU can never be 100% due to the size difference between the predicted bounding box and the ground truth bounding box, while YOLOv5 does not have this issue since it generates a custom bounding box for each object and frame. This primarily affects the success plots and corresponding AUC since those are the only metrics that use IoU.

However, CoTracker and TAPIR were both initialized on the bounding box center

point of the first appearance of each object but are evaluated against the annotated bounding box center points throughout the video, i.e., the center point in the first appearance might be slightly different if the object is appearing from the side compared to the center point when the object is right in front of the camera. YOLOv5 on the other hand was evaluated on every detected bounding box center point. This difference might cause some unfairness in evaluating the models to YOLOv5's advantage on all metrics except for occlusion accuracy, indicating that it is not entirely suitable to compare point trackers and object detectors.

Another factor why YOLOv5 outperformed the other models is that YOLOv5 in this case is specified to detect exactly these objects in this setting. Initially, we tried to use YOLOv5 out of the box but with no success since it is trained on datasets (Redmon et al., 2016) unlikely to include stickers similar to the ones we used. Therefore we had to do additional training to be usable in this experiment, and given that the test videos are very similar to the ones it was trained on, it seems natural that it would get good results with the risk of it being overfitted for this specific environment. Due to time restrictions, no additional testing was done, but had we implemented some alternation with backdrops, placing the stickers on another object, the result might have been different.

From this experiment, we can conclude that point trackers are suitable for semi-automatic video annotation but they are not as accurate as object detectors. The advantage of CoTracker and TAPIR compared to regular object detector models is that they are usable out of the box and do not need specific training to annotate any physical surface in a video. However, if time and resources are not a limitation, our results indicate that an object detector might achieve better results. The issue with using an object detector such as YOLOv5 to annotate video data is that one would need annotated images to train the model accordingly, and it does not necessarily solve the issue of extensive manual annotation, while these point tracker models only need to be trained once and can then be applied to most videos.

5.2 Video Properties of Failure and Success Cases

The five videos in TAP-Vid DAVIS where CoTracker and TAPIR performed worst have several characteristics that make the video visually complicated. Both models shared four out of five as their worst-performing videos. These videos have several properties that could complicate tracking the query points, such as the camera moving or the object of interest changing shape due to rotation or moving farther or closer to the camera. Comparing the worst-performing videos to the best-performing, there are some commonalities found. There is however a difference in the amount of properties found in the worst and best-performing videos. A video where CoTracker or TAPIR tracks the query points accurately may have a few properties that affect the performance negatively, but when the negative properties increase the performance worsens increasingly.

Videos in which the models achieve higher accuracy have a clear view of the object of interest throughout the video. The camera movement is horizontal, with a minimal

amount of shaking or movement, and the object of interest is constantly in the frame. Four out of the five videos have objects of interest that undergo minimal or no rotation. The video with query points on a rotating object has the worst performance with regards to AJ, $< \delta^x_{avg}$, and OA. As stated above, the videos where CoTracker and TAPIR performed well have few or none of the negatively affecting properties.

Worse-performing videos have several of the negatively affecting properties. Four of these five videos in the TAP-Vid DAVIS dataset have extensive or quick camera shakes throughout the video, causing the ground-truth tracks to become jagged and irregular in shape. This is accompanied by several occlusions. It is reported by Karaev et al. (2023) that CoTracker handles occlusions well. However, when the object of interest is moving away from the camera while being occluded, the object of interest undergoes shape and size changes, consequently changing the visual context of the key point being tracked. This makes it difficult for both CoTracker and TAPIR to not only find the correct coordinates for the query point when it reappears but also to track the correct point within the consecutive frames. Occlusions may not only be any material object, such as a tree or a pole, but also light flashes. One of the worst-performing videos in the TAP-Vid DAVIS dataset shows a man swinging from a rope and shooting a rifle. Each time the rifle is shot, a flash appears, occluding the area behind it. The light flashes also cause differences in color in the areas surrounding it which causes the RGB-values CoTracker utilizes as input to change irregularly.

A common theme in the best videos was that all but one had the main object in the scene being either a car or an animal, whereas the worst videos had the opposite characteristics, i.e., all but one was of humans doing something. This, however, does not indicate that humans per se are more complicated to track, but humans tend to do more complex actions that are more difficult to track. Following this, it may not be the object itself as much as which query points are chosen that determines the performance of the trackers. Another factor to consider why the point trackers perform worse on videos with humans could have to do with the data the models are trained on. Both CoTracker and TAPIR are trained on the synthetic TAP-Vid Kubric dataset, which does not contain any videos of humans moving around. The reason for training on this dataset is that the circumstances for comparing and evaluating the models using the TAP-Vid benchmark need to be consistent and fair. However, this synthetic dataset might have a distribution that is different from the data in TAP-Vid DAVIS used for evaluation. This is not a conclusive fact and deserves further examination. Training the point trackers on a dataset with humans moving around in various scenarios and then evaluating the model on the TAP-Vid DAVIS dataset would be an interesting topic of research for uncovering how this might affect the qualitative aspects of model performance.

As for the worst-performing videos in the Rotating Rubik's Cube dataset, they were all made up of videos with external occluding objects. This is less prevalent when only the cube rotates and the object of interest disappears behind its axis. Figure 3.1 demonstrates the difference between the cases when the models had good and bad performance as there are fully visible and occluded query points. The query

points that are visible throughout the video are successfully tracked, while the query points on the side of the cube disappear during rotation, causing the models to lose track of their location.

Of the six different stickers, the star stickers were the most difficult for all of the models to track. This is most likely due to the reflective gold colors that the star stickers have. The reflecting quality of the star stickers makes them appear white in certain angles and the model gets stuck on some other part of the Rubik's cube, obstacle, or white background. The sizes of the stickers do affect the performances as all models performed best on the big red circle.

Examining the AJ results among the worst and best-performing videos shows that CoTracker achieves the highest and lowest results compared to TAPIR on TAP-Vid DAVIS videos. The cause of this may be due to CoTracker's accounting of the correlation between the tracked points. Jointly tracking points is reported as beneficial for tracking performance (Karaev et al., 2023) but if a point's trajectory is inaccurately predicted, this affects the prediction of the other point trajectories. If two points have a strong positive correlation and one of them is predicted inaccurately, the correlation may cause the other point to be led astray as well. Therefore, CoTracker manifests a more high-risk and high-reward behavior which is beneficial when the circumstances are correct but is also punishing for more complicated videos.

5.3 Limitations

This thesis primarily looked at rotating objects in a sterile environment, making this a niche dataset. Therefore the results can be seen as proof of concept about the possibilities of using point trackers for semi-automatic video annotation, and we can not say much about how accurate they are in other environments.

Furthermore, YOLOv5 was trained with a batch size of two which might cause sub-optimal performance from the model. Peng et al. (2018) reports that using a small batch size primarily increases the training time, but can also affect the model accuracy. However, during training, YOLOv5 had good performance which justifies using a small batch size.

Moreover, the estimations of the point trackers' bounding boxes based on the object of interest's first appearance possibly had a negative effect on CoTracker and TAPIR's performance as discussed in section 5.1. In retrospect, it might have been better to dynamically estimate the bounding box of each query point for each frame in the video.

5.4 Future Topics of Research

Karaev et al. (2023) states the beneficial effects of the use of correlation but based on the results from this study presents another topic of future research; whether or not the correlations benefit the point tracking output in more difficult videos.

In this study, the estimated bounding boxes from the point trackers were constant in size. Changes to this method could be an interesting topic of research. If the bounding boxes instead were within a changing size range it could give improvements to annotation accuracy with regards to IoU.

Karaev et al. (2023), Doersch et al. (2023) and the models applied in this study were trained on the TAP-Vid Kubric benchmark. Future research could examine performance qualitatively and quantitatively using another training dataset in order to evaluate how the training data affects the outcome.

6

Conclusion

This thesis found that point tracker models do not demonstrate superior accuracy to object detector models in semi-automatic video annotation given that YOLOv5 outperformed CoTracker and TAPIR across all metrics except one. Therefore, one should consider using YOLOv5 if training data is available and resources are not limited. However, given that CoTracker and TAPIR do not need to be trained before using them for annotation, they can be used when there is no annotated data available.

Moreover, the qualitative properties of videos where point trackers fail to track objects of interest are primarily externally occluding objects, and when the object of interest rotates. The reason for this is that when a query point temporarily disappears, the point trackers find it difficult to relocate the accurate position of the query point thus resulting in lower performance.

Bibliography

Aqel, M. O. A., Marhaban, M. H., Saripan, M. I., & Ismail, N. B. (2016). Review of visual odometry: Types, approaches, challenges, and applications. *Springerplus*, *5*(1), 1897.

Babenko, B., Yang, M.-H., & Belongie, S. (2011). Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(8), 1619–1632. https://doi.org/10.1109/TPAMI.2010.226

Benjumea, A., Teeti, I., Cuzzolin, F., & Bradley, A. (2021). Yolo-z: Improving small object detection in yolov5 for autonomous vehicles. *arXiv preprint arXiv:2112.11798*.

Bolya, D., Foley, S., Hays, J., & Hoffman, J. (2020). Tide: A general toolbox for identifying object detection errors. *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, 558–573.

Brdjanin, A., Dardagan, N., Dzigal, D., & Akagic, A. (2020). Single object trackers in opencv: A benchmark. *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, 1–6.

Chadwick, S., & Newman, P. (2019). Training object detectors with noisy data. *2019 IEEE Intelligent Vehicles Symposium (IV)*, 1319–1325.

Chen, F., Wang, X., Zhao, Y., Lv, S., & Niu, X. (2022). Visual object tracking: A survey. *Comput. Vis. Image Underst.*, *222*(103508), 103508.

Chen, Z., Wu, R., Lin, Y., Li, C., Chen, S., Yuan, Z., Chen, S., & Zou, X. (2022). Plant disease recognition model based on improved YOLOv5. *Agronomy (Basel)*, *12*(2), 365.

Doersch, C., Gupta, A., Markeeva, L., Recasens, A., Smaira, L., Aytar, Y., Carreira, J., Zisserman, A., & Yang, Y. (2022). Tap-vid: A benchmark for tracking any point in a video. *Advances in Neural Information Processing Systems*, *35*, 13610–13626.

Doersch, C., Yang, Y., Vecerik, M., Gokay, D., Gupta, A., Aytar, Y., Carreira, J., & Zisserman, A. (2023). Tapir: Tracking any point with per-frame initialization and temporal refinement. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 10061–10072.

Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazrba, C., Golkov, V., v.d. Smagt, P., Cremers, D., & Brox, T. (2015). Flownet: Learning optical flow with convolutional networks. *IEEE International Conference on Computer Vision (ICCV)*. http://lmb.informatik.uni-freiburg.de/Publications/2015/DFIB15

Dwyer, B., Nelson, J., & Solawetz, J. e. a. (2022). *Roboflow (Version 1.0) [Software]*. Available from. https://roboflow.com

Ericsson. (2020). Eva. https://github.com/Ericsson/eva

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.

Fazeli, S., Sabetti, J., & Ferrari, M. (2023). Performing qualitative content analysis of video data in social sciences and medicine: The visual-verbal video analysis method. *International Journal of Qualitative Methods*, *22*. https://doi.org/10.1177/16094069231185452

Fortun, D., Bouthemy, P., & Kervrann, C. (2015). Optical flow modeling and computation: A survey. *Comput. Vis. Image Underst.*, *134*, 1–21.

Gaur, E., Saxena, V., & Singh, S. K. (2018). Video annotation tools: A review. *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*.

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *Int. J. Rob. Res.*, *32*(11), 1231–1237.

Greff, K., Belletti, F., Beyer, L., Doersch, C., Du, Y., Duckworth, D., Fleet, D. J., Gnanapragasam, D., Golemo, F., Herrmann, C., et al. (2022). Kubric: A scalable dataset generator. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 3749–3761.

Harley, A. W., Fang, Z., & Fragkiadaki, K. (2022). Particle video revisited: Tracking through occlusions using point trajectories. *European Conference on Computer Vision*, 59–75.

Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2012). Exploiting the circulant structure of tracking-by-detection with kernels. In *Computer vision – ECCV 2012* (pp. 702–715). Springer Berlin Heidelberg.

Hosang, J., Benenson, R., & Schiele, B. (2017). Learning non-maximum suppression. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ince, K. G., Koksal, A., Fazla, A., & Alatan, A. A. (2021). Semi-automatic annotation for visual object tracking. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*.

Jocher, G., Chaurasia, A., & Qiu, J. (2023, January). *Ultralytics YOLO* (Version 8.0.0). https://github.com/ultralytics/ultralytics

Jocher, G., Stoken, A., Borovec, J., NanoCode012, ChristopherSTAN, Changyu, L., Laughing, tkianai, Hogan, A., lorenzomammana, yxNONG, AlexWang1900, Diaconu, L., Marc, wanghaoyang0106, ml5ah, Doug, Ingham, F., Frederik, ... Rai, P. (2020, October). *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements* (Version v3.1). Zenodo. https://doi.org/10.5281/zenodo.4154370

Karaev, N., Rocco, I., Graham, B., Neverova, N., Vedaldi, A., & Rupprecht, C. (2023). Cotracker: It is better to track together. *arXiv preprint arXiv:2307.07635*.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Le Ludec, C., Cornet, M., & Casilli, A. A. (2023). The problem with annotation. human labour and outsourcing between france and madagascar. *Big Data Soc.*, *10*(2).

Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., et al. (2022). Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*.

Lin, J., Gan, C., Wang, K., & Han, S. (2021). Tsm: Temporal shift module for efficient and scalable video understanding on edge device. *arXiv preprint arXiv:2109.13227*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Computer vision – ECCV 2014* (pp. 740–755). Springer International Publishing.

Loshchilov, I., & Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*, 91–110.

Mao, X., Chen, Y., Zhu, Y., Chen, D., Su, H., Zhang, R., & Xue, H. (2023). Coco-o: A benchmark for object detectors under natural distribution shifts. *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., & Brox, T. (2016). A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4040–4048. https://doi.org/10.1109/CVPR.2016.438

Nassauer, A., & Legewie, N. M. (2021). Video data analysis: A methodological frame for a novel research trend. *Sociological methods & research*, *50*(1), 135–174.

Nazir, A., & Wani, M. A. (2023). You only look once - object detection models: A review. *2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, 1088–1095.

Neoral, M., erch, J., & Matas, J. (2024). Mft: Long-term tracking of every pixel. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 6837–6847.

Olorunshola, O. E., Irhebhude, M. E., & Evwiekpaefe, A. E. (2023). A comparative study of YOLOV5 and YOLOV7 object detection algorithms. *Deleted Journal*, *2*(1), 1–12. https://doi.org/10.33736/jcsi.5070.2023

Peng, C., Xiao, T., Li, Z., Jiang, Y., Zhang, X., Jia, K., Yu, G., & Sun, J. (2018). Megdet: A large mini-batch object detector. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 6181–6189.

Perazzi, F., Pont-Tuset, J., McWilliams, B., Gool, L. V., Gross, M., & Sorkine-Hornung, A. (2016). A benchmark dataset and evaluation methodology for video object segmentation supplemental material. https://api.semanticscholar.org/CorpusID:260972929

Pereira, N. (2022). Pereiraaslnet: Asl letter recognition with yolox taking mean average precision and inference time considerations. *2022 2nd International Conference on Artificial Intelligence and Signal Processing (AISP)*, 1–6. https://doi.org/10.1109/AISP53593.2022.9760665

Perrigo, B. (2022). Inside Facebooks African Sweatshop. https://time.com/6147458/facebook-africa-content-moderation-employee-treatment/

Pont-Tuset, J., Perazzi, F., Caelles, S., Arbeláez, P., Sorkine-Hornung, A., & Van Gool, L. (2017). The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*.

Raji, F., Ke, L., Tai, Y.-W., Tang, C.-K., Danelljan, M., & Yu, F. (2023). Segment anything meets point tracking. *arXiv preprint arXiv:2307.01197*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., & Li, F. (2015). ImageNet Large Scale Visual Recognition Challenge. *International journal of computer vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sand, P., & Teller, S. (2006). Particle video: Long-range motion estimation using point trajectories. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, *2*, 2195–2202. https://doi.org/10.1109/CVPR.2006.219

Shah, S. T. H., & Xiang, X. (2021). Traditional and modern strategies for optical flow: an investigation. *SN applied sciences/SN Applied Sciences*, *3*(3). https://doi.org/10.1007/s42452-021-04227-x

Teed, Z., & Deng, J. (2020). Raft: Recurrent all-pairs field transforms for optical flow. *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, 402–419.

Terven, J., Córdova-Esparza, D.-M., & Romero-González, J.-A. (2023). A comprehensive review of YOLO architectures in computer vision: from YOLOV1 to YOLOV8 and YOLO-NAS. *Machine learning and knowledge extraction*, *5*(4), 1680–1716. https://doi.org/10.3390/make5040083

Tsai, Y.-H., Yang, M.-H., & Black, M. J. (2016). Video segmentation via object flow. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3899–3908. https://doi.org/10.1109/CVPR.2016.423

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace.

Vondrick, C., Patterson, D. J., & Ramanan, D. (2012). Efficiently scaling up crowdsourced video annotation. *International journal of computer vision*, *101*(1), 184–204. https://doi.org/10.1007/s11263-012-0564-1

Wang, B.-L., King, C.-T., & Chu, H.-K. (2018). A semi-automatic video labeling tool for autonomous driving based on multi-object detector and tracker. *2018 Sixth International Symposium on Computing and Networking (CANDAR)*, 201–206. https://doi.org/10.1109/CANDAR.2018.00035

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2023). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 7464–7475.

Wang, Q., Chang, Y.-Y., Cai, R., Li, Z., Hariharan, B., Holynski, A., & Snavely, N. (2023). Tracking everything everywhere all at once. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 19795–19806.

Wu, Y., Lim, J., & Yang, M.-H. (2013). Online object tracking: A benchmark. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2411–2418. https://doi.org/10.1109/CVPR.2013.312

Xu, R., Lin, H., Lu, K., Cao, L., & Liu, Y. (2021). A forest fire detection system based on ensemble learning. *Forests*, *12*, 217. https://doi.org/10.3390/f12020217

Yuen, J., Russell, B., Liu, C., & Torralba, A. (2009). Labelme video: Building a video database with human annotations. *2009 IEEE 12th International Conference on Computer Vision*, 1451–1458. https://doi.org/10.1109/ICCV.2009.5459289

Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Zheng, Y., Harley, A. W., Shen, B., Wetzstein, G., & Guibas, L. J. (2023). Pointodyssey: A large-scale synthetic dataset for long-term point tracking. *ICCV*.

Zhiqiang, W., & Jun, L. (2017). A review of object detection based on convolutional neural network. *2017 36th Chinese Control Conference (CCC)*, 11104–11109. https://doi.org/10.23919/ChiCC.2017.8029130

Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). Object detection in 20 years: A survey. *Proceedings of the IEEE*, *111*(3), 257–276. https://doi.org/10.1109/JPROC.2023.3238524