Tracking Olympiad SS 2024 Report

Martin Reimer martin.reimer@fau.de
Artificial Intelligence in Communication Disorders (anki lab)
Friedrich-Alexander Universität Erlangen

Editor:

Abstract

This paper explores the challenge of tracking small moving objects using Hexbugs, which are electronic devices that mimic insect movements. We investigate whether incorporating body labels, generated using the Segment Anything Model from Meta, enhances the accuracy of head detection. YOLOv8 with transfer learning was employed for detection, and two tracking methods, Euclidean distance-based and Lucas-Kanade optical flow, were evaluated.

Our results indicate that body labels improve detection accuracy, particularly when combined with Lucas-Kanade tracking. Despite solid detection performance, challenges like frequent ID switching and difficult scenarios highlight the need for further refinement. Future work should focus on incorporating color and shape information to improve tracking consistency and handling dynamic object counts in video sequences.

1 Introduction

In this project, we explore the challenge of tracking small moving objects using Hexbugs electronic devices (Hexbug.com (2024)) designed to simulate the erratic movements of insects. This work serves as a simplified model for complex animal tracking applications, where the ability to monitor the movement and interactions of multiple animals provides valuable insights into behavioral studies (Pereira et al. (2022)). Multi-Object Tracking (MOT), the underlying technology for such tasks, plays a critical role in numerous computer vision applications, enabling systems to identify and follow multiple objects over time within a given scene. This capability is essential in fields such as traffic monitoring, autonomous vehicles, medical diagnosis systems, and beyond (Soleimanitaleb et al. (2019)). In particular, animal tracking offers a significant application of MOT, with quantitative measurements of animal motion foundational for understanding behavior in fields like neuroscience and ecology (Pereira et al. (2022)).

MOT typically involves two main components: object detection and object tracking.

Object Detection refers to the process of identifying and locating objects within individual frames of a video. According to Pal et al. (2021), detection methods can be categorized into appearance-based, motion-based, and deep learning (DL)-based approaches. Appearance-based methods rely on image processing techniques to recognize objects directly from images, though they often struggle with occluded objects. Motion-based methods use sequences of images to detect objects but can falter in complex scenes. The most recent advancements are seen in DL-based approaches, which utilize convolutional neural networks

(CNNs) to extract features from images for object classification and bounding box prediction. These DL-based methods are further divided into two-stage detectors, which first propose regions of interest before classifying them, and one-stage detectors, which predict bounding boxes directly, offering faster but sometimes less accurate detections. In this paper, we will be utilizing a DL-based one-stage approach, the You Only Look Once (YOLO) algorithm.

Object Tracking, on the other hand, is the process of maintaining the identity and position of these detected objects across successive frames. Soleimanitaleb et al. (2019) categorizes tracking methods into feature-based, segmentation-based, estimation-based, and learning-based approaches. Feature-based tracking uses unique attributes like color or texture to match objects across frames, while segmentation-based methods isolate foreground objects from the background. Estimation-based methods apply Bayesian frameworks to predict object trajectories, and learning-based methods, including deep learning, adaptively learn the object's appearance and movement patterns over time, allowing for more robust tracking even in challenging conditions such as occlusion or changes in scale. In this study, we will employ feature-based tracking methods, specifically using Euclidean distance-based tracking and the Lucas-Kanade Optical Flow algorithm. Euclidean distance-based tracking matches objects across frames based on their proximity in space, while the Lucas-Kanade method utilizes motion estimates.

Despite advancements in object detection and tracking methods, several challenges persist. These include dealing with illumination variation, background clutter, low resolution, and occlusion, all of which can degrade the performance of tracking algorithms (Soleimanitaleb et al. (2019)). Additionally, fast motion and significant changes in object appearance across frames, such as rotation or deformation, further complicate accurate tracking. These challenges are especially pronounced in scenarios involving small, rapidly moving objects, where even slight inaccuracies in detection or tracking can lead to significant errors.

In this seminar project, we address the challenge of tracking small moving objects using Hexbugs. The specific task was to track the head of these Hexbugs, a problem complicated by their small size, rapid and erratic movements, and the low frame rate of the available videos. Our goal was to develop and compare three different approaches using a dataset of 100 training and evaluation videos and 5 test videos.

The remainder of this report is structured as follows: In Section 2, we describe the methods and algorithms developed for this project, including a details of the experiment setting, data preparation, and both the detection and tracking components. Section 3 presents the results of our experiments, comparing the performance of the different methods. Finally, in Section 4, we discuss the implications of our findings, the challenges encountered, and potential directions for future work.

2 Methods

The primary objective of this work is to accurately track the head of Hexbug devices across videos. In this study, we evaluate two different detection approaches and two tracking algorithms. The detection approaches include using a YOLO model trained on head labels and another trained on both head and body labels. The tracking methods explored are Euclidean distance-based tracking and optical flow-based tracking.

The chapter is organized into subsections covering the experiment setting and dataset, data preparation, detection algorithms, tracking algorithms, tested approaches and evaluation methods for assessing performance.

2.1 Setting

Dataset The dataset comprises 105 videos, with 100 videos used for training and validation, and 5 reserved exclusively for testing (shown in figure 1). The videos in the dataset vary in terms of the number of Hexbugs present, their movement patterns, and environmental conditions (e.g., lighting, background). The dataset also includes scenarios where Hexbugs are in challenging positions, such as lying on their side or back, as well as instances of low-light conditions and color variations, with some colors like black being underrepresented. This variability is crucial for developing a detection model that generalizes well across different scenarios.

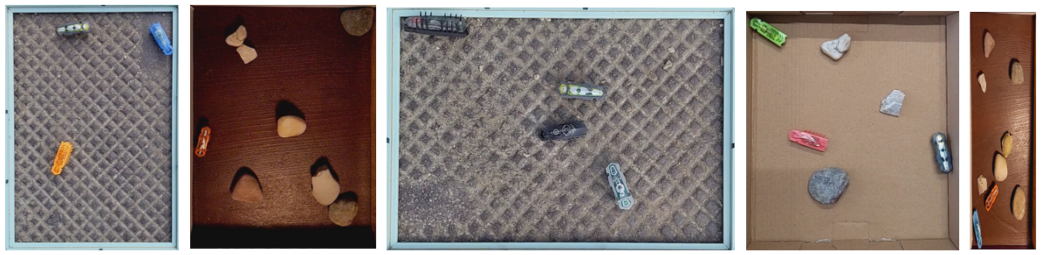

Figure 1: Each image represents the initial frame of one test video, showcasing the variability and context of the test scenarios

Experimental Environment All experiments were conducted using Python. Resource-intensive computations, particularly those involving model training and data processing, were performed using Google Colab Pro with access to an A100 GPU. This setup provided the necessary computational power for generating the body labels and for training the detection models.

2.2 Data Preparation

2.2.1 New Labeling Using Adjusted Tool

While the original 100 videos were already labeled, we employed the TRACO Labeling Tool to annotate an additional 7 videos. This tool, optimized for Hexbug labeling, allows for precise tracking of Hexbug heads by selecting a center point on the head in each frame. The labeled data can be exported in a CSV format, including the tracked IDs for further processing. However, to tailor the tool to our specific needs, we made several adjustments:

- We added functionality to export data in YOLO bounding box format. After labeling the center point of a Hexbug head, the user is prompted to select the approximate size of the bounding box. This size is then applied uniformly across all labeled frames, allowing direct usage with YOLO-based detection models.

- Given that our dataset consists of videos with low frame rates (fps), we incorporated an option to adjust the fps during labeling. This ensures that the new data remains consistent with the original dataset's characteristics.
- We improved the tool's UI by displaying annotation texts with corresponding IDs alongside the labeled points and simplified the workflow for more efficient labeling.

An illustration of this labeling tool is in figure 5.

The additional 7 videos were labeled to address underrepresented scenarios in the original dataset, such as Hexbugs lying on their side or back, variations in color (e.g., black Hexbugs), and low-light conditions.

2.2.2 Label Generation

Given that our initial labels only provided center points for Hexbug heads, we needed to generate bounding boxes for both the heads and bodies.

Head Labeling For head detection, we defined a bounding box with a fixed width and height of 80 pixels around the labeled head center points. This size was chosen after experimenting with different bounding box dimensions on the training videos to ensure optimal detection performance.

Body Labeling Although our primary goal is to track the Hexbug heads, we hypothesized that incorporating body labels might improve the model's detection accuracy. We believe that detecting the body is not only simpler than predicting the head—since the head can be easily mistaken for the tail—but also that body detection could enhance overall detection performance. To address the labor-intensive and time-consuming task of manual labeling, we applied a novel approach using the Segment Anything Model (SAM) developed by Meta.

SAM is a versatile segmentation model that generates object masks from input prompts like points or boxes (Kirillov et al. (2023)). We utilized the "vit_h" variant, the largest model. SAM was employed to generate segmentation masks for all objects in each frame, and then we refined the selection to determine the most appropriate mask for each Hexbug body. The underlying process is illustrated in figure 2.

Since multiple segmentation boxes could be generated for a single pixel, we implemented refinement criteria to select the most suitable box. We chose the smallest segmentation box that covered at least 50% of the YOLO head bounding box area and ensured it was at least twice the size of the head bounding box to encompass the entire body.

On the A100 GPU, processing each frame took approximately 8 seconds, which included generating and refining segmentation masks. The entire operation, spanning the processing of all frames for all 107 videos, took around 25 hours.

After generating the body labels, we observed that some segmentation boxes were inaccurate which can be seen in figure 3. This was partly due to issues with the box selection algorithm, which sometimes selected incorrect segments, such as shadows or background structures, instead of the Hexbug bodies. Correcting all such errors manually was infeasible; thus, we focused on easily identifiable and correctable bounding boxes. Specifically, we filtered out boxes that were disproportionately large in width, height, or overall area, removing bounding boxes in around 100 frames where these errors were most evident. While

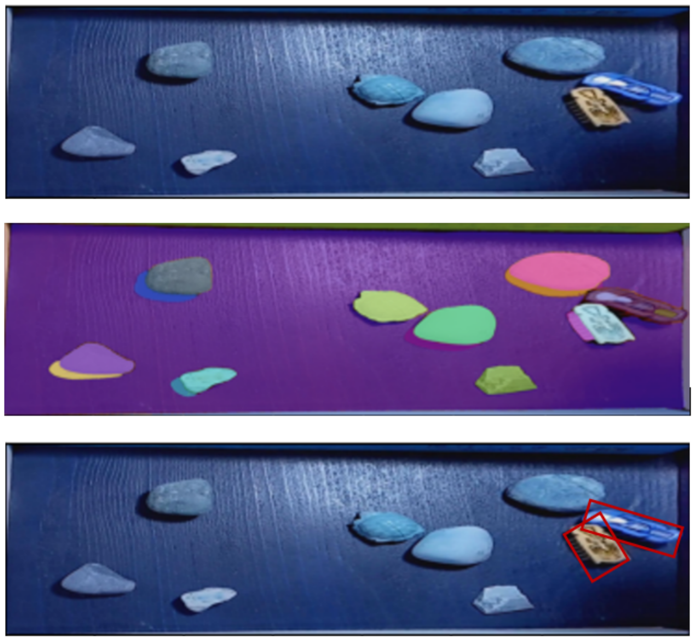

Figure 2: The first image shows the original frame. The second image displays the frame with all segmentation outputs from the Segment Anything Model. The third image illustrates the final bounding boxes, selected based on the head label and post-processing criteria.

erroneous bounding boxes still exist in the dataset, we hypothesize that the detection model will be robust enough to handle these noisy labels and still effectively learn to identify the Hexbug bodies.

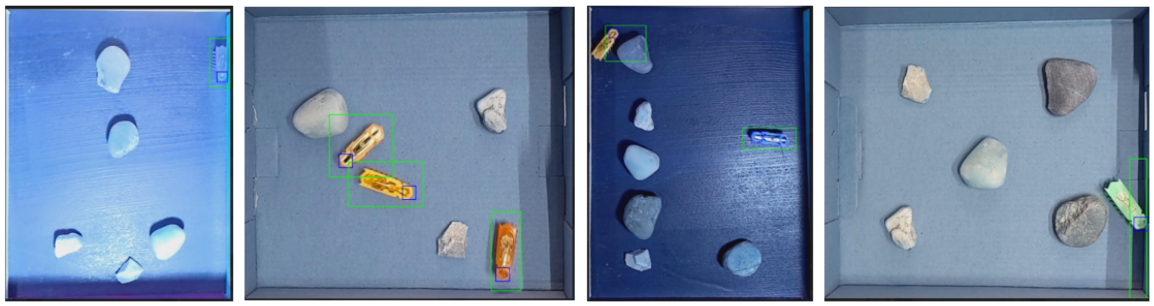

Figure 3: The first two images show generated body labels without errors and other two include both one false detection each.

2.2.3 Data Splitting

To ensure a comprehensive evaluation of our detection models, our team manually split the data, distributing different scenarios (e.g., side-lying objects, large objects) evenly between the training and evaluation sets. The specific video IDs used for each set are as follows:

- Training Set: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 88, 89, 95, 101, 102, 103, 104, 105, 107, 108, 109, 111, 112, 114, 116}
- Validation Set: {77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 117}
- Test Set: Five videos were reserved exclusively for testing.

2.2.4 Data Augmentation

To further enhance the model's robustness, we applied various data augmentation techniques using the Ultralytics library (github.com (2024)). Data augmentation is crucial for improving model generalization, particularly when training data is limited or when the data includes variations such as stretched videos.

In addition to standard augmentation techniques, we introduced a custom `StretchImage` class that applies stretching transformations to some images (see implementation in Appendix B). This was designed to simulate scenarios where the proportions of width and height are altered. Since stretched images were present in the test data but not in the training videos, this augmentation helps bridge the gap between training and test conditions by artificially introducing these variations during training.

The used augmentation techniques are:

- `StretchImage`: Custom augmentation to simulate video stretching.
- `CopyPaste`: Combines objects from different images to create new training examples.
- `RandomPerspective`: Applies random perspective transformations to images.
- `MixUp`: Combines two images with a random factor to enhance training data diversity.
- `Albumentations`: A suite of transformations including random cropping, color adjustments, and more.
- `RandomHSV`: Adjusts the hue, saturation, and value of images to simulate different lighting conditions.
- `RandomFlip`: Flips images horizontally or vertically to increase variability.

Mosaic augmentation was considered but ultimately removed from the pipeline. It was incompatible with the `StretchImage` transformation, as its implementation required images to have similar width and height, which conflicted with the purpose of stretching the images.

2.3 Detection Algorithm

Object detection is a crucial step in our methodology, and for this purpose, we employed the YOLOv8 (You Only Look Once version 8) model. YOLO is a family of models known for balancing accuracy and inference speed, making it suitable for real-time object detection tasks. Both detection approaches in this study are based on YOLOv8.

2.3.1 YOLO

The YOLO framework revolutionized object detection by processing images in a single pass through the network, in contrast to traditional methods that used sliding windows or region proposal networks followed by classification (Terven et al. (2023)). Over the years, YOLO has undergone multiple iterations, from YOLOv1 to YOLOv8 and further. Each version brought enhancements in performance.

YOLOv8, released by Ultralytics in January 2023, is one of the latest iteration in the YOLO series and supports multiple vision tasks such as object detection, segmentation, pose estimation, tracking, and classification (Terven et al. (2023)). YOLOv8 is offered in five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra large).

In our study, we used the smallest available pre-trained model, `yolov8n.pt`, due to its balance of computational efficiency and detection performance. Although we experimented with the small and medium versions (`yolov8s.pt` and `yolov8m.pt`), we did not observe significant improvements in performance. Therefore, we chose to proceed with the nano version.

2.3.2 Detection Approaches

Two detection approaches were tested in our study:

- YOLO with Head Labels: In this approach, we trained the YOLO model using only head labels. The class label for the head is defined as `class=0`.
- YOLO with Body and Head Labels: In this approach, the YOLO model was trained using both head and body labels. Two class labels were defined: `class=0` for the head and `class=1` for the body.

The used configurations and hyperparamters are illustrated in Appendix C.

2.4 Tracking Algorithm

In this section, we describe the tracking algorithms employed in our study. After obtaining the detections from YOLO, two different tracking methods are used: tracking via minimal Euclidean distance and tracking via the Lucas-Kanade optical flow method.

2.4.1 Tracking Assumptions

Both tracking algorithms rely on several key assumptions to ensure their effectiveness:

- The algorithm assumes a fixed number of objects throughout a given video, simplifying tracking.

- The algorithm assumes that the movement of the hexbugs is constrained to a certain distance between frames.
- The algorithm assigns the first detection it iterates over to the nearest object, assigning remaining detections to the next closest object, for the case that multiple head detections are closest to the same object.
- If fewer detections than expected are present in the initial frame, the algorithm artificially creates a detection at the top-left corner.
- If detections are missing in the current frame, the algorithm assumes the object hasn't moved and retains its last known position.

2.4.2 Detection Analysis and Confidence Level Selection

Before applying the tracking algorithms for a video, we analyze our detections over all video frames to select the optimal detection inference confidence level. This involves:

- Utilizing only detections above a specified confidence level.
- For the body-head approach, body detections are preferred due to their alleged higher reliability, while for the head-only approach, head detections are used.
- We test different confidence levels $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]$, computing the number of detections per frame and calculating the standard deviation.
- The confidence level with the minimal standard deviation is selected, and the median number of detected objects at that confidence level is used for the entire video.

2.4.3 Body-Head Detection Approach

In the body-head detection approach, the head is detected based on the body detection:

- Detect_Head_from_Body:
  - The YOLO inference results are obtained for the body class using the selected confidence level and a predefined Intersection over Union (IoU) threshold of 0.4.
  - For the head class, a lower confidence threshold of 0.1 is used to generate a larger number of head detections.
  - We iterate through the body predictions and filter head detections by requiring that each body detection contains at least one head detection within its bounding box.
  - If multiple heads are detected within a single body detection, the head with the highest confidence is selected.
  - If no head is detected within a body bounding box, an artificial head detection is added.

2.4.4 Head-Only Detection Approach

In the head-only detection approach:

- Detect_Head:
  - The YOLO inference results are obtained directly for the head class using the selected confidence level and a predefined IoU threshold of 0.4.

2.4.5 Tracking via Minimal Euclidean Distance

The minimal Euclidean distance tracking method operates under the following assumptions: The tracking process involves iterating through frames:

- In the first frame, detections are initialized with an ID. If fewer detections are found than expected, artificial detections are added at the top-left corner.
- For consecutive frames:
  - A distance matrix is created between tracked objects from the last frame and new detections from YOLO output.
  - A list is maintained to keep track of already assigned IDs.
  - Each YOLO head detection is matched with the nearest tracked object, assuming that the nearest object is less than 300 pixels away.
  - If multiple head detections are closest to the same object, the first detection is assigned, and the others are matched to the next nearest object.
  - Any missing detections are filled in using the position from the last frame.

2.4.6 Tracking via Lucas-Kanade Optical Flow

The tracking process involves iterating through frames:

- In the first frame, detections are initialized with an ID. If fewer detections are found than expected, artificial detections are added at the top-left corner.
- For consecutive frames:
  - Points from the last frame are used to predict their next positions using Lucas-Kanade optical flow. The used implementation here is the `cv.calcOpticalFlowPyrLK()` function from the OpenCV library (docs.opencv.org (2024)).
  - New points are iterated over, and their corresponding YOLO head detections are matched by minimal distance, provided the head is not already assigned to another point.
  - Missing detections are filled in using the position from the last frame.

2.5 Approaches

In this study, we evaluated the performance of four different approaches for detecting and tracking Hexbugs. Each approach combines different detection and tracking techniques to assess their effectiveness in various scenarios. The following approaches were tested:

- **Body-Head Detection + Euclidean Distance Tracking**: We trained on both head and body labels and utilized both predictions in the final tracking.
- **Body-Head Detection + Lucas-Kanade Tracking**: We trained on both head and body labels and utilized both predictions in the final tracking.
- **Head Detection + Euclidean Distance Tracking**: We trained the model using only head labels and applied the predictions for tracking.
- **Head Detection + Lucas-Kanade Tracking**: We trained the model using only head labels and applied the predictions for tracking.
- **Head(-Body) Detection + Euclidean Distance Tracking**: In this approach, we trained on both head and body labels but only utilized the head predictions for tracking.

We tested these combinations to evaluate the impact of different detection and tracking algorithms on performance. Specifically, we included the Head Detector trained on both head and body labels to determine if incorporating body information improves head detection accuracy. The other combinations were tested to understand how detectors trained with different label sets perform with both tracking algorithms. This approach helps us identify which combinations provide the best results and why certain label sets or tracking methods may enhance detection and tracking performance.

2.6 Evaluation Method

To evaluate the performance of the tested approaches, we utilized the TRACO leaderboard evaluation server. This server provides a framework for assessing tracking algorithms using a set of ground truth labels. The evaluation process involves the following steps:

- Uploading Results: Tracking results for all 5 test videos are uploaded to the TRACO evaluation server. The server then compares these results against the ground truth labels to calculate performance scores.
- Evaluation Metrics: The performance score is determined based on several factors, including: accuracy of the center point predictions, maintaining the correct id over all the frames, detection of the correct number of Hexbugs and several other factors.
- Scoring System: The evaluation produces two scores: an absolute value and a percentage value. The absolute value provides a raw score reflecting the model's performance, while the percentage value ranges from 0% to 100%, with 0% representing the lowest possible performance and 100% indicating perfect performance. While the percentage value helps in comparing models, it may not fully reflect the qualitative aspects of model performance.

- Visual Inspection: In addition to the scores, we perform a visual analysis of the detections and tracking results. This involves examining video frames to identify instances of ID switching, assessing performance across different scenarios, and understanding where each model excels or encounters issues.

3 Results

The evaluation results for the five different approaches are summarized in the tables below. Each approach was assessed based on the score and percentage provided by the evaluation server.

Table 1: Comparison of Tracking Approaches

Approach	**TRACO-Score**	**Percentage**
Body-Head Detection + Euclidean Distance Tracking	106,924	29.27%
Body-Head Detection + Lucas-Kanade Tracking	114,134	31.24%
Head Detection + Euclidean Distance Tracking	96,067	26.30%
Head Detection + Lucas-Kanade Tracking	106,752	29.22%
Head(-Body) Detection + Euclidean Distance Tracking	96,387	26.39%

The results indicate that the Body-Head Detection + Lucas-Kanade Tracking approach achieved the highest performance, with a TRACO-Score of 114,134, corresponding to 31.24%. On the other hand, the Head Detection + Euclidean Distance Tracking approach showed the lowest performance, with a TRACO-Score of 96,067 and a percentage of 26.30%.

4 Discussion

In this discussion, we will analyze the TRACO scores and tracking visualizations, followed by an outlook on potential improvements.

4.1 Discussion of TRACO Scores

These are the key observations given the performance results of the approaches:

Performance of Tracking Algorithms Across all detection methods, the Lucas-Kanade tracking algorithm consistently outperforms Euclidean distance tracking. For instance, when combined with Body-Head Detection, Lucas-Kanade achieves a TRACO-Score of 114,134 (31.24%), compared to 106,924 (29.27%) for Euclidean distance tracking. This trend is also evident with Head Detection.

Performance of Detection Algorithms When comparing approaches that use both head and body labels (Body-Head Detection) to those that use only head labels (Head Detection), the Body-Head detection approaches tend to perform better overall. This is seen in the higher scores for Body-Head Detection + Lucas-Kanade (114,134) compared to Head Detection + Lucas-Kanade (106,752). Interestingly, the Head(-Body) Detection approach, where the model was trained on both labels but used only head predictions, performs similarly to the standard Head Detection method. This implies that while the detection model benefits from being trained on both head and body labels, the actual tracking performance may not significantly improve unless both predictions are used during the tracking process.

General Observations The data suggests that training using both head and body labels paired with a more advanced tracking method such as Lucas-Kanade, results in the best performance. This combination likely provides the model with richer information and a more resilient tracking mechanism.

4.2 Discussion of Tracking Visualizations

To gain deeper insights into the performance of the different tracking approaches, we conducted a visual analysis of the tracking results across five test videos. Below, we detail the key observations for each test video (see figure 1 for illutrations of the initial video frames):

Video 1 In this video, all tracking algorithms exhibited a single ID switch. This indicates a generally stable tracking performance.

Video 2 The tracking performance was nearly perfect in this video. Only two frames presented issues where the detection failed in a dark area, leading the algorithm to rely on the detection from the previous frame. This demonstrates that the detection models handle well-lit scenarios effectively but struggles with poor lighting conditions.

Video 3 This video posed significant challenges for the tracking algorithms, largely due to frequent collisions between the Hexbugs. The result was a high number of ID switches. Additionally, all detectors struggled with Hexbugs that were lying on their sides or backs. An interesting observation was that the Body-Head detection models could detect the black Hexbugs, which were underrepresented in the training data, whereas the Head-only detectors rarely identified them. However, the Body-Head models still struggled with detecting Hexbugs in lying positions, with only about 50% of these scenarios being detected correctly.

Video 4 All algorithms performed well in this video, with only 1-2 ID switches observed.

Video 5 This video presented a complex and stretched scenario, leading to interesting observations. The Body-Head detection models outperformed the Head-only models in detecting Hexbugs. In the Head-only models, detection was inconsistent throughout the video, with many frames having no detections at all. The Body-Head models, however, were able to detect Hexbugs in almost all frames, though the accuracy and consistency were suboptimal. The `analyze_detections` method, which identifies the most consistent interval by analyzing the median number of objects per frame, found that the Head-only detectors most consistently predicted zero detections per frame, with only occasional detections of one or two heads. As a result, the method recommended tracking 0 Hexbugs for the Head-only detector. This led to no points being awarded for head detector-based trackings in this video. In contrast, the Body-Head detectors consistently predicted two bodies per frame, even though they struggled to accurately predict the head position, often defaulting to using the center of the body as the head.
Despite the fact that our labels were sometimes noisy or not always available, YOLO was still able to learn the body labels effectively. Additionally, we did not observe any instances of incorrect head detections that were outside the body of the Hexbug, further supporting the conclusion that the inclusion of body labels contributed positively to the overall

detection and tracking process. This example highlights that in complex and stretched scenarios, Body-Head detection models are more effective, though they still face challenges, particularly in accurate head prediction.

General Observations Overall, the tracking algorithms generally performed well across the five videos, with detection issues arising primarily in scenarios involving frequent collisions, lying Hexbugs (which were underrepresented in the training data), and stretched video conditions (see figure 4). The most significant tracking issues were observed in complex scenarios, such as those with numerous collisions, where the algorithms struggled to maintain consistent IDs. Additionally, some unexpected behaviors were noted in the tracking, indicating that further pre-testing and fine-tuning of the tracking algorithms may be necessary to improve performance.

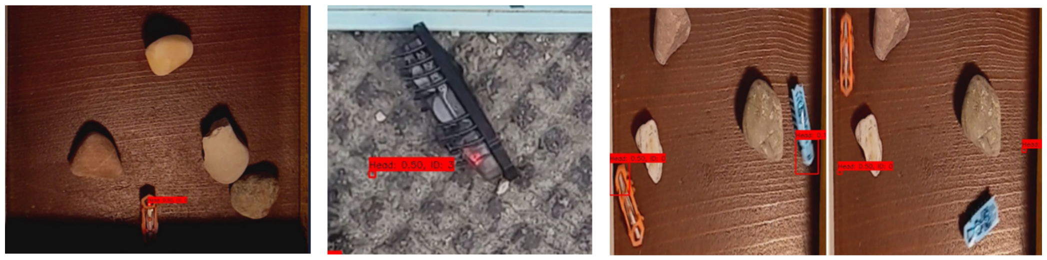

Figure 4: Illustration of the challenges faced by the detection and tracking approaches. The first image shows a missed detection due to poor lighting conditions. The second image depicts a lying Hexbug that wasn't detected, causing the tracking to rely on the last known detection coordinates. The third image presents a two-frame sequence from a video where the detection struggled, often relying on detections from earlier frames.

4.3 Outlook for Future Work

While the detection performance is generally satisfactory, there is room for improvement, particularly in handling difficult scenarios. Acquiring more training data that includes varied and complex situations, such as collisions, lying Hexbugs, and stretched scenes, would likely enhance the model's robustness and accuracy.

Tracking, on the other hand, presents significant opportunities for improvement. One of the main challenges observed was ID switching, which is an area where human observers excel due to their ability to utilize color and shape differences. This suggests that integrating methods that leverage color and specific shape features could improve ID consistency. Traditional techniques such as color histograms, as well as deep learning approaches like Siamese networks to compute similarities between object IDs, could be explored to enhance tracking performance. Combining these with optical flow methods might provide a more robust tracking solution.

Additionally, our current approach makes several assumptions, such as the presence of a fixed number of objects throughout the video. This is not always realistic, as Hexbugs can enter or exit the scene. Future work should explore how to effectively handle dynamic scenarios where objects may appear or disappear, ensuring the tracking algorithm can adapt to these changes without losing accuracy.

Appendix A. Adjusted Labeling Tool

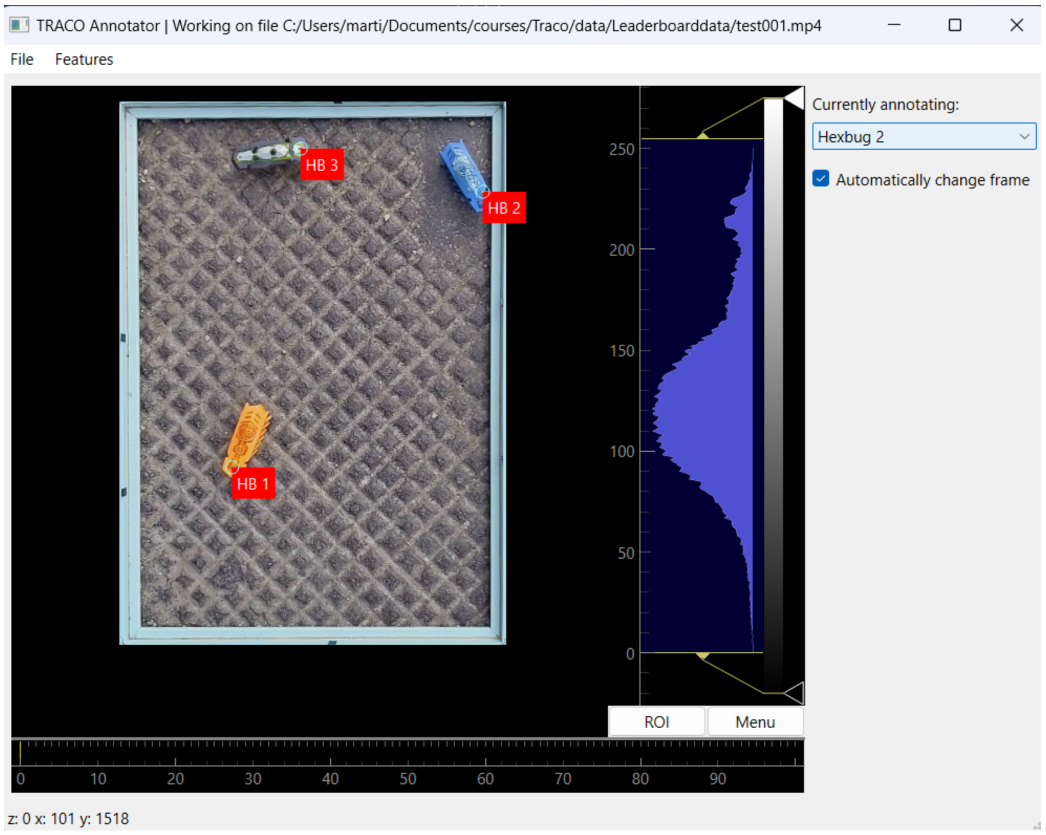
TRACO Annotator | Working on file C:/Users/marti/Documents/courses/Traco/data/Leaderboarddata/test001.mp4
File
Features
HB 3
HB 2
HB 1
250
200
150
100
50
0
ROI
Menu
0
10
20
30
40
50
60
70
80
90
Currently annotating:
Hexbug 2
Automatically change frame
z: 0 x: 101 y: 1518

Figure 5: The adjusted labeling tool used in our study.

Appendix B. Python Code for Data Augmentation: StretchImage

```python
import cv2
import numpy as np

class StretchImage:
    def __init__(self, ratios=None, p=1.0) -> None:
        """
        Initializes the StretchImage class with a list of possible ratios
    and a probability of applying the transformation.

        Args:
            ratios (list, optional): A list of float values representing the
     possible width-to-height ratios.
                                    Default is a predefined list of ratios.
            p (float, optional): The probability of applying the
    transformation. Must be between 0 and 1. Default is 1.0.
        """
        self.p = p
        assert 0 <= p <= 1.0, "Probability must be between 0 and 1."
        if ratios is None:
            self.ratios = [0.2, 0.12, 3.8, 2.5, 0.43, 3.6, 5.8, 6.7, 4.2,
    0.55, 0.36, 6.2]
        else:
            self.ratios = ratios
        self.current_ratio = 1.0  # Default ratio

    def __call__(self, labels):
        if random.random() < self.p:
            self.current_ratio = random.choice(self.ratios)
            img = labels["img"]
            orig_w, orig_h = img.shape[:2]

            if self.current_ratio > 1:  # Widen the image
                new_w = orig_w
                new_h = int(orig_h / self.current_ratio)
            else:  # Heighten the image
                new_w = int(orig_w * self.current_ratio)
                new_h = orig_h

            # Resize the image
            new_image = cv2.resize(img, (new_w, new_h))

            labels["img"] = new_image
        return labels
```

Listing 1: Python code for the StretchImage augmentation class

Appendix C. Configurations and Hyperparameters used for the detection model training

```
# YOLOv8 Training Configuration for Class Head
model: /content/yolov8n.pt
epochs: 15
batch: 16
imgsz: 640
pretrained: true
optimizer: auto
iou: 0.7
max_det: 300
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 7.5
cls: 0.5
```

```
dfl: 1.5
pose: 12.0
kobj: 1.0
label_smoothing: 0.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
auto_augment: randaugment
erasing: 0.4
crop_fraction: 1.0
stretch_ratios: 0.2, 0.12, 3.8, 2.5, 0.43, 3.6, 5.8, 6.7, 4.2, 0.55, 0.36, 6.2
stretch_image_prob: 0.15

# YOLOv8 Training Configuration for Classes Body & Head
model: /content/yolov8n.pt
epochs: 15
batch: 16
imgsz: 640
pretrained: true
optimizer: auto
iou: 0.7
max_det: 300
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.0
box: 7.5
cls: 0.5
dfl: 1.5
pose: 12.0
```

```
kobj: 1.0
label_smoothing: 0.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 0.0
mixup: 0.0
copy_paste: 0.0
auto_augment: randaugment
erasing: 0.4
crop_fraction: 1.0
stretch_ratios: 0.2, 0.12, 3.8, 2.5, 0.43, 3.6, 5.8, 6.7, 4.2, 0.55, 0.36, 6.2
stretch_image_prob: 0.15
```

References

docs.opencv.org. Opencv - optical flow. `https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html`, 2024. Accessed: 2024-08-09.

github.com. Ultralytics library. `https://github.com/ultralytics/ultralytics/tree/main/ultralytics`, 2024. Accessed: 2024-08-09.

Hexbug.com. Hexbug nano. `https://www.hexbug.com/nano.html`, 2024. Accessed: 2024-08-09.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, October 2023. doi: 10.1109/iccv51070.2023.00371. URL `http://dx.doi.org/10.1109/ICCV51070.2023.00371`.

Sankar K. Pal, Anima Pramanik, Jhareswar Maiti, and Pabitra Mitra. Deep learning in multi-object detection and tracking: state of the art. *Applied Intelligence*, 51:6400 – 6429, 2021. URL `https://api.semanticscholar.org/CorpusID:234827020`.

Talmo Pereira, Nathaniel Tabris, Arie Matsliah, David Turner, Junyu Li, Shruthi Ravindranath, Eleni Papadoyannis, Edna Normand, David Deutsch, Z. Wang, Grace McKenzie-Smith, Catalin Mitelut, Marielisa Castro, John D'Uva, Mikhail Kislin, Dan Sanes, Sarah Kocher, Samuel Wang, Annegret Falkner, and Mala Murthy. Sleap: A deep learning system for multi-animal pose tracking. *Nature Methods*, 19:1–10, 04 2022. doi: 10.1038/s41592-022-01426-1.

Zahra Soleimanitaleb, Mohammad Ali Keyvanrad, and Ali Jafari. Object tracking methods:a review. *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 282–288, 2019. URL `https://api.semanticscholar.org/CorpusID:210930009`.

Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. 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, November 2023. ISSN 2504-4990. doi: 10.3390/make5040083. URL `http://dx.doi.org/10.3390/make5040083`.