Pattern Recognition 135 (2023) 109107

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Detection confidence driven multi-object tracking to recover reliable tracks from unreliable detections

Travis Mandel[a,*], Mark Jimenez[a], Emily Risley[a], Taishi Nammoto[a], Rebekka Williams[a,b], Max Panoff[a,c], Meynard Ballesteros[a], Bobbie Suarez[a]

[a] Univerity of Hawai'i at Hilo, 200 W. Kawili St, Hilo, HI, USA
[b] Univerity of Hawai'i at Mānoa, 2500 Campus Road, Honolulu, HI, USA
[c] University of Florida, Gainesville, FL, USA

ARTICLE INFO

Article history:
Received 22 December 2021
Revised 3 August 2022
Accepted 9 October 2022
Available online 13 October 2022

Keywords:
Multi-object tracking
Model uncertainty
Performance evaluation
Scarce data
Dataset
Marine science applications

ABSTRACT

Multi-object tracking (MOT) systems often rely on accurate object detectors; however, accurate detectors are not available in every application domain. We present Robust Confidence Tracking (RCT), an offline MOT algorithm designed for settings where detection quality is poor. Whereas prior methods simply threshold and discard detection confidence information, RCT relies on the exact detection confidence values to increase track quality throughout the entire tracking pipeline. This innovation (along with some simple and well-studied heuristics) allows RCT to achieve robust performance with minimal identity switches, even when provided with completely unfiltered detections. To compare trackers in the presence of unreliable detections, we present a challenging real-world underwater fish tracking dataset, FISHTRAC. In an large-scale evaluation across FISHTRAC, UA-DETRAC, and MOTChallenge data, RCT outperforms a wide variety of trackers, including deep trackers and more classic approaches. We have publically released our FISHTRAC codebase and training dataset at https://github.com/tmandel/fish-detrac, which will facilitate comparing trackers on understudied problems.

1. Introduction

Multi-object tracking (MOT) is a longstanding computer vision problem in which the goal is to keep track of the identities and locations of multiple objects throughout a video. A popular MOT approach is tracking-by-detection, in which an object detector is first run on every frame, and those detections are fed as input to a MOT algorithm. Convolutional neural networks (CNNs) have led to the creation of highly accurate detectors, thus spurring the development of approaches that rely heavily on these high-quality detections [1].

Training such highly accurate detectors requires significant labeled data. The majority of the MOT literature has focused on tracking pedestrians and vehicles [2,3], two settings in which labeled data is plentiful. However, in specialized tracking scenarios we may have considerably less data; for instance, tracking a new species of insect, or tracking fish off the coast of a tropical island. With limited training data, even the best detectors will have limited performance. Although historically researchers have often separated the problems of tracking and detection (especially for understudied domains such as fish [4] and insects [5]) by assuming access to a perfect detector, reduced detection quality has the potential to compound into extremely poor-quality tracks. In contrast, an ideal tracking algorithm would be able to perform robustly even given an imperfect detector [6], but it is still not clear how to accomplish this.

Most of the recent top performers have been deep tracking approaches, which either learn to associate detections into coherent tracks [7], or learn how to recover tracks from videos in an end-to-end fashion without relying on an external detector [8,9]. However, these systems require training on large amounts of labeled data; in the cases we study, little to no labeled video data is available. Indeed, even properly labeled still image data needed to train an object detector may be fairly scarce, greatly increasing the difficulty of the problem compared to the standard MOT setting.

We have found that even when (pretrained) CNN detectors are trained on little data, they often are still able to predict the general location of objects in the scene, albeit sometimes with very low confidence and many false positives. However, the traditional MOT pipeline discards most of this information, first filtering out the low-confidence detections, and thereafter discarding the de-

* Corresponding author.
E-mail address: tmandel@hawaii.edu (T. Mandel).

https://doi.org/10.1016/j.patcog.2022.109107

tection confidence values [3]. Ideally, a tracker could make use of the full *unfiltered* set of detections to achieve more robust performance. Unfortunately, removing this filtering step greatly increases the computational burden, and requires algorithms to cope with extremely noisy input. Due to these challenges, we are not aware of any tracking-by-detection approaches capable of efficiently recovering high-quality tracks from the unfiltered output of a low-accuracy detector.

Therefore, we present Robust Confidence Tracking (RCT), an algorithm which tracks efficiently and robustly given unfiltered detections as input. The key idea behind RCT is that, instead of discarding detection confidence values, we can use these values to guide the tracking process, using lower-confidence detections only to "fill in gaps" between higher-confidence detections. Specifically, RCT uses detection confidence in three ways: To determine where to best initialize tracks, probabilistically combined with a motion model to optimally extend tracks, and to filter out low-quality tracks. Alongside this novel idea, RCT incorporates some simple and well-studied heuristics, such as a Kalman filter to model motion, a MedianFlow single object tracker (SOT) to incorporate visual information, and greedy track agglomeration to join tracklets into overall tracks. Despite these simple heuristics, RCT's incorporation of detection confidence achieves excellent performance, even compared to more complicated and resource-intensive deep tracking methods. To test trackers such as RCT in challenging scenarios where data is scarce, datasets of common objects do not suffice. Therefore, we present a new, challenging real-world fish tracking dataset, FISHTRAC. We conduct a comprehensive evaluation of RCT and an extremely diverse set of multi-object tracking approaches across both FISHTRAC as well as two more standard datasets when using a low-accuracy detector, UA-DETRAC [3] and MOT17 [2]. The primary contributions of our work are as follows:

- To cope with a lack of high-quality detections, existing approaches that simply threshold the detection confidence do not suffice. Therefore, RCT uses the exact (unfiltered) detection confidence values to increase track quality in three distinct ways (initializing tracks, extending tracks, and filtering tracks). This novel idea, combined with a few simple and well-studied heuristics (Kalman-filter, MedianFlow, and greedy track agglomeration), results in a tracker which outperforms a wide variety of other algorithms in this setting. We show in our experiments that RCT's performance critically relies on this novel method for incorporating detection confidence: Thresholding the confidence at any fixed value worsens performance greatly.
- Through public competitions like MOTChallenge, the computer vision community has an accurate sense of how MOT algorithms perform in the regime of high-quality detections. However, it is unknown how these algorithms perform when provided low-quality detections, a scenario that occurs in many real-world domains. We are the first to perform a systematic comparison of a wide range of trackers in this setting, including classic tracking approaches, single object tracking approaches, specialized fish trackers, and deep tracking approaches (including three top performers on recent challenges). Moreover, this comparison is performed over three distinct domains; we are not aware of any MOT paper that compares a more diverse set of baselines over multiple domains.
- We present a new high-resolution real-world fish tracking dataset, FISHTRAC, which covers the setting of tracking fish in an underwater environment, a scenario not well covered by existing datasets. Real-world fish tracking is a very challenging setting due to complex camera motion, object motion, lighting, and background effects.
- Our codebase has been made publicly available at https://github.com/tmandel/fish-detrac. This codebase is an important contribution, as there do not exist codebases that allow one to compare a wide variety of MOT trackers on *new* datasets.

2. Problem setup

We consider offline multi-object tracking problems within a tracking-by-detection framework, where the goal is to track all objects of a desired class ℓ. Note that ℓ is often known to be of practical importance, but in the settings we consider may be rare enough that accurate detection is difficult. Specifically, we assume there exists a video $\mathcal{V}$ with N frames $v_1, \ldots, v_N$ and a detector $\mathcal{D}$ which outputs detections on each frame $d_1, \ldots, d_N$. Each d_i is a set containing tuples $b = (x, y, w, h, c)$ denoting the detected bounding box and its confidence $0 \leq c \leq 1$ that the box corresponds to an instance of an object of class ℓ (here we use $b \in \ell$ to denote the case that a box is a member of the class ℓ). The goal of the tracking algorithm $\mathbb{T}$ is to produce an optimal set of tracks $T = T_1, \ldots, T_K$ where each track T_j consists of a list of tuples $t_{j,f} = (x, y, w, h, c)$ where $f \in [1, N]$ is the frame number.

3. Related work

Multi-object Tracking Datasets and Codebases There are several public MOT datasets, see Table 1. However, tracking fish in natural underwater scenarios is a challenging and understudied problem which is not well-represented by existing datasets. Although some datasets do include fish data, the video is usually of artificial settings such as aquarium tanks, which greatly simplifies the tracking problem. The one significant exception is Fish4Knowledge/SeaCLEF [10–12], however that dataset suffers from several problems, including low image quality and low FPS (5 FPS). Indeed, most of the datasets with more variety have sacrificed FPS (e.g. 1 FPS for TAO [13]), and some such as TAO also have incomplete annotations, making comprehensive evaluation difficult. Low FPS is a particularly poor choice for fish tracking, since fish move and change direction rapidly. Our FISHTRAC dataset contains high-quality (at least 1920x1080) video of real-world underwater fish behavior, and is completely annotated at 24 FPS. While not as diverse or large as datasets like TAO [13], FISHTRAC fills an important gap by helping shed light on a highly challenging real-world application.

Additionally, there is currently a lack of MOT codebases that facilitate comparison to other methods.[1] Leaderboards such as MOTChallenge [2] are the predominant way to compare trackers, but this does not allow one to compare algorithms on new videos or detections. Running trackers on a new dataset takes substantial implementation time and effort (converting formats, handling very slow trackers, etc.). The UA-DETRAC [3] codebase allows one to compare 8 trackers, but it is intended for only a single dataset and is based on proprietary (paid) software (MATLAB). We present a heavily modified version of the DETRAC code which is adapted to open-source technologies and contains everything needed to run 17 trackers on fish data, car data, pedestrian data, or a new dataset. The code is available online at https://github.com/tmandel/fish-detrac.

Fish tracking Work in real-world fish tracking has been relatively scarce due to lack of suitable datasets. Most attention has focused on artificial settings, such as tracking Zebra Fish in a glass enclosure [16]. One exception is Jäger et al. [11], who developed a custom approach to track fish in real-world scenarios. We compare to this tracker in our experiments.

[1] In the SOT community, substantial efforts have been made to develop publicly available codebases for comparing a wide variety of SOT algorithms on new datasets.

Table 1
A comparison of public MOT datasets. FPS refers to the annotation FPS.
Comparing UA-DETRAC, TAO, MOT17, YTVIS2021, OVIS, SeaCLEF, and our FISHTRAC data set on number of videos, number of "in the wild" fish videos, annotation FPS, minimum resolution, whether they provide unfiltered detections, whether they are completely labeled, and the number of trackers included in the associated codebase.

Dataset	Num Videos	# "In the wild" fish videos	FPS	Min resolution	Provides unfiltered detections?	Complete labels?	# MOT algs in codebase
UA-DETRAC [3]	100	0	24	960x540	No	No	8
TAO [13]	2907	2	1	640x480	N/A	No	1
MOT17 [2]	14	0	30	640x480	No	Yes	0
YTVIS 2021 [14]	3859	2	5	320x180	N/A	Yes	0
OVIS [15]	901	2	3–6	864 x 472	N/A	Yes	0
SeaCLEF [11,12]	10	10	5	320x240	N/A	No	2
FISHTRAC (Ours)	14	14	24	1920x1080	Yes	Yes	17

Detection Confidence Virtually all tracking-by-detection methods filter detections based on a confidence threshold h and thereafter discard confidence information, e.g. let $d'_f = b$ s.t. $c_b \geq h, b \in d_f$, where c_b denotes the confidence of the box b. Indeed, this is enforced by codebases such as UA-DETRAC [3]. The few exceptions add another threshold to differentiate between high and medium confidence [17], or require modifying detection approaches to expose additional information which may not always be accessible [18,19]. Bayesian approaches like JPDA [20,21] incorporate a fixed detection probability, but do not utilize the individual detection confidences. Approaches such as CMOT and its variants [22] develop a version of track "confidence" that is based on factors such as kinematic plausibility, and/or appearance features over time [23], but do not incorporate the confidence of the underlying object detector. Although such approaches can work well, one limitation is that in cases such as fish tracking, fish tend to dramatically change appearance and motion within a small period of time, for instance when a fish begins to turns around rapidly. These appearance and motion changes cause these approaches to produce many ID switches in this setting (as we see with CMOT in our experiments). We are not aware of any MOT algorithms that make use of the detection confidence values associated with each produced bounding box in a manner more sophisticated than just thresholding them, a procedure which eliminates the more nuanced information contained in these values.

4. Robust Confidence Tracking (RCT)

Our Robust Confidence Tracking (RCT) algorithm contains four components: Initializing tracks based on detection confidence, probabilistically combining detection confidence with motion probability, incorporating a single-object tracker as a fallback when detections alone are insufficient, and track postprocessing. Fig. 1 gives an overview.

4.1. Initializing tracks based on detection confidence

Unlike other MOT algorithms, our RCT algorithm uses the detection confidence as a key to distilling the detections into coherent tracks. For each track T_j, RCT chooses the maximum confidence detection (across all frames) for its initial box I_j ($I_j = \text{argmax}_{b \in d_1 \cup \cdots \cup d_N} c_b$) ; we refer to the frame associated with I_j as $f_{I,j}$. To ensure that this detection does not overlap with a previously-used track, RCT excludes boxes from the max where there exists some track index W such that $t_{W,f_{I,j}} \in d_{f_{I,j}}$ and $|B(t_{W,f_{I,j}}) \cap B(I_j)| > 0$, where the function B returns the set of all pixel coordinates that fall within the box. Also, to avoid edge cases, we do not select detections that are near the edge of the screen (we enlarge I_j by β% and check that it is still onscreen, where β is an RCT parameter). RCT works in a track-wise fashion: once the first track is built (described below), the next highest detection confidence detection is selected as the start frame for the next track, and so on, as long as the initial confidence $c_{I,j}$ is above an RCT threshold parameter h_I. Note that RCT uses detections with confidence $< h_I$ elsewhere.

4.2. Combining detection confidence and motion

Next, RCT initializes a Kalman filter k with the chosen detection I_j. The Kalman filter state s^k is a tuple $(x^k, y^k, v_x^k, v_y^k, w^k, h^k)$ where v_x^k and v_y^k are unobserved (latent) velocities which together form a vector $\vec{v}^k =< v_x^k, v_y^k >$. Let $b^k = (x^k, y^k, w^k, h^k)$ be the box derived from the state.

From the initial box I_j, RCT could extend the track either forward or backward in time, but it does not know which will best help estimate velocity. To handle this, RCT initially tries both options, and selects the option with the best score (as defined below). For clarity we will describe the forward case, the backward case is analogous.

Given a frame f, partial track j and a Kalman filter state $s^k_{f,j}$, RCT must score each detection in d_f to find the best candidate to extend the track. The Kalman filter probability of the box based on the track so far $P(b' = t_{j,f}|t_{j,f-1}, \ldots, t_{j,1})$ can be used as a motion model score. Similarly, the Detector D assigns a probabilistic score $P(b' \in \ell|v_f) = c_{b'}$ reflecting the probability the detector believes this object is of the desired class. Our goal is to find the joint probability $P(b' = t_{j,f}, b' \in \ell|v_f, t_{j,f-1}, \ldots, t_{j,1})$. If we make the simplifying assumption that the class and the track assignment are conditionally independent given the past sequence of boxes and frame image, we have:

$$\begin{aligned} &P(b' = t_{j,f}, b' \in \ell|v_f, t_{j,f-1}, \ldots, t_{j,1}) \\ &\quad = P(b' = t_{j,f}|v_f, t_{j,f-1}, \ldots, t_{j,1}) P(b' \in \ell|v_f, t_{j,f-1}, \ldots, t_{j,1}) \quad (1) \end{aligned}$$

Since our detector gives $P(b' \in \ell|v_f, t_{j,f-1}, \ldots, t_{j,1}) = P(b' \in \ell|v_f) = c_{b'}$, and our Kalman filter assumes $P(b' = t_{j,f}|v_f, t_{j,f-1}, \ldots, t_{j,1}) = P(b' = t_{j,f}|s^k_{j,f})$, the joint probability can be calculated as:

$$P(b' = t_{j,f}, b' \in \ell|v_f, t_{j,f-1}, \ldots, t_{j,1}) = c_{b'} P(b' = t_{j,f}|s^k_{j,f}). \quad (2)$$

The conditional independence assumption is a simplification, but it is justified by the fact that we can determine whether a given box is an extension of the current track based on the object's motion, independent of the object's class. Conversely, we can determine the class of a given box based solely on the visual information contained in the image, independent of the track to which the object is assigned.

RCT uses Eq. (2) to score a detected box based on both detection confidence and motion model score. However, this does not tell us when none of the detected boxes on a certain frame are a reasonable extension of the track - a situation that arises frequently with an imperfect detector. To do so, RCT checks two criteria. First, RCT checks whether the center point of the chosen detection b' is contained within the box derived from the Kalman

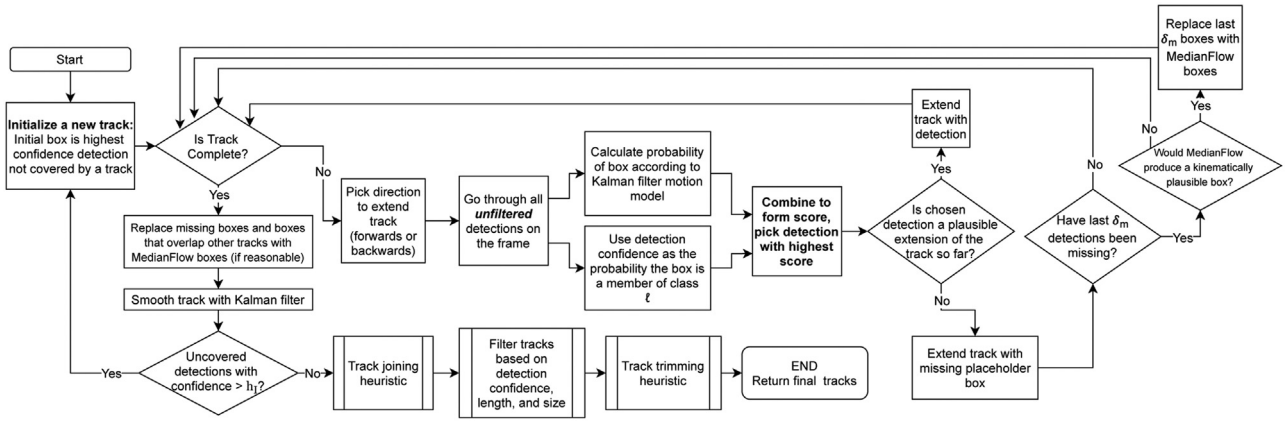

Fig. 1. Process diagram of our Robust Confidence Tracking (RCT) algorithm.
A flowchart showing a visual flow of the RCT algorithm. Covers how RCT initializes tracks, how RCT combines detection confidence and a motion model to extend tracks. The diagram also shows how, if RCT has to resort to a motion model, it first checks if a MedianFlow box is a reasonable extension of the track. The diagram also shows the postprocessing steps (joining, filtering, trimming, etc.) that occur in our algorithm.

filter state, specifically $C(b') \in B(b^k_{j,f})$ where C is a function that returns the geometrical center of the box. If not, it is likely not a kinematically plausible extension of the track.[2] Next, RCT checks if $P(b' = t_{j,f}|s^k_{j,f}) \geq P(b' = t_{j,f-1}|s^k_{j,f-1})$, in other words, whether the previous detection is more likely under the current Kalman filter state than the previous Kalman filter state: if not, it is likely moving in the wrong direction. If the detection is rejected due to either of the above reasons, RCT sets $t_{j,f}$ to a placeholder value indicating a missing observation. Otherwise, RCT sets $t_{j,f} = b'$, and marks b' so that it cannot be re-used in another track.

After δ (δ is an RCT parameter) iterations extending the track in both directions, RCT then switches to a single direction (forward, then backward) as the estimate of the velocity is likely sufficiently accurate. RCT stops this process when the current box is more offscreen than the last box, setting the rest of the t_j to missing since the object is offscreen.

The Kalman filter is used to perform one final smooth at the end, letting $t_{j,f} = b^k_{j,f}$ to smooth out any noise in the track and replace missing observations with inferred boxes.[3]

4.3. Incorporating a single object tracker

The aforementioned approach forms the core of the RCT algorithm; however, the Kalman filter assumes linear motion if we do not find a matching detection, which performs poorly when motion is complex. Therefore, RCT uses a SOT algorithm as a fallback option if no reasonable detections can be found. Specifically, we use the MedianFlow tracker [24]: MedianFlow has been successfully used in past MOT approaches [17]; a strength is that it can determine when it has lost track of an object. As with the Kalman filter, we initialize the MedianFlow tracker on frame I_f and update it in both the forwards and the backwards directions.

We observed that the Kalman filter could overcome a short sequence of missing detections or occlusions, while visual information is critical to overcoming a longer sequence of missing detections. Therefore, RCT switches to MedianFlow only if, when detecting on frame f of track j, for all $f' \in \{f, f-1, \ldots, f-\delta_m\}$ $t_{j,f'} \notin d_{f'}$ (i.e. the last δ_m frames also have no valid detections), where δ_m is an RCT parameter. Additionally, we require that the MedianFlow track is plausible according to our Kalman filter, specifically RCT tests that $C(m_{j,f}) \in B(b^k_{j,f})$, where $m_{j,f}$ is the MedianFlow box on frame f of track j. If these conditions are met, and MedianFlow did not report a tracking failure, RCT sets $t_{j,f'} = m_{j,f'}$ for $f' \in \{f, f-1, \ldots, f-\delta_m\}$. In the case where both a MedianFlow box $m_{j,f}$ and an acceptable detected box $b' \in d_j$ are available, and the previous box was MedianFlow ($t_{j,f-1} = m_{j,f-1}$), RCT only sets $t_{j,f} = b'$ if $C(b') \in B(m_{j,f})$ and $C(m_{j,f}) \in B(b')$, which tests whether the detection diverges significantly from the MedianFlow prediction. If it does, it is likely a spurious detection and RCT keeps using the MedianFlow boxes.

To further reduce the reliance on motion, RCT replaces some boxes with MedianFlow after the track is built. If on some track j and frame f, either the detection is a missing placeholder or it overlaps with another track (i.e. there exists some $w \neq j$ such that $t_{w,f} \in d_f$ and $|B(t_{j,f}) \cap B(t_{w,f})| > 0$), we try to see if we can replace $t_{j,f}$ with a better box. First, RCT tries a MedianFlow box: if $|B(m_{j,f}) \cap B(t_{j,f-1})| > 0$, it is a reasonable extension of the track, so we let $t_{j,f} = m_{j,f}$. Else, RCT sets $t_{j,f}$ to indicate a missing observation.

4.4. Track joining, confidence-based filtering, trimming

The approach so far can produce tracks that are fragmented - therefore, RCT joins smaller tracks (tracklets) as a postprocessing step. Instead of computationally expensive matching approaches [25], we use a greedy agglomerative track joining approach based on a simple heuristic that joins two tracks if they are similar enough in terms of time and motion. Specifically, RCT examines the time period in which the tracks switched to purely motion. Without loss of generality, let f_j be the last non-motion-box frame of track j, and f_W be the first non-motion-box frame of track W (we try every possible ordered pairing of tracks). If $f_j \leq f_W$, then RCT computes the temporal distance as $D_{time} = f_W - f_j$ If $f_j > f_W$, then we require there to be at most two frames f where $IoU(t_{f,W}, t_{f,j}) < h_u$, where IoU is the intersection over union function and h_u is an RCT parameter. In other words, the track needs to overlap on almost every frame in which there are detections, if so RCT sets $D_{time} = 0$. RCT only considers joining pairs of tracks where $D_{time} < D_{\max}$, where $D_{\max}$ is an RCT parameter. Next we consider whether the distance reached in that number of frames would be reasonable according to the Kalman filter. Specifically, let

$$v_{max} = \max_{i \in \{w,j\}, f \in \{1,\ldots,N\}} \sqrt{\left(v^k_{i,f,x}\right)^2 + \left(v^k_{i,f,y}\right)^2}, \quad (3)$$

[2] We optimize by only considering detections that overlap $b^k_{j,f}$.

[3] Specifically, RCT runs a (forward-backward) smoothing pass separately on frames $f' < f_{l,j} + \delta$ and $f' > f_{l,j} - \delta$, where the δ extra frames past the start frame are used as context.

which gives the fastest speed it is reasonable for these objects to have under the Kalman filter. Then we test whether

$$d_{euclid}(C(b_{f_w,w}), C(b_{f_j,j})) \leq D_{time} v_{max}, \quad (4)$$

where d_{euclid} gives the Euclidean distance. If not, the distance between tracks is too large to reasonably join them. Additionally, RCT checks that the tracked boxes are moving in the right direction: that is, that a Kalman filter initialized on track w and extended to track j would determine that $P(b_{j,f_j}|s^k_{f_j}) \geq P(b_{j,f_j}|s^k_{f_j-1})$. RCT iterates this process, greedily adding tracks until there are no more pairs that meet our join criteria. After each join, RCT re-smooths the tracks.

Since the detector is low-accuracy, it may be that long sequences of detections occur on objects outside ℓ (e.g. coral instead of fish), necessitating track filtering. RCT again relies on detection confidence to filter tracks: Detections of smaller objects tend to be naturally lower confidence, but if a track is both exceptionally long and contains many large boxes, at least some of the detections should be fairly high confidence if it is truly the target class. To determine if a track T_q qualifies as large and long, we first define a set $\mathcal{T}_l$ consisting of all the high-quality large long tracks. Specifically, $T_j \in \mathcal{T}_l$ if two conditions are met: $c_{l,j} > h_q$, where h_q is an RCT parameter, and $S(T_j) \geq \frac{\sum_{T_i \in T} S(T_i)}{|T|}$, where $S(T_i) = \sum_{f \in \{1,\ldots,N\}} |B(t_{i,f})|$, that is, the total size (as calculated by summing the box sizes across all frames) greater than the mean across all tracks. For each $T_i \in \mathcal{T}_l$, RCT computes $S(T_i)$, producing a set of scalar sizes $\mathcal{S}_l$. RCT then fits a Gaussian distribution to the mean and standard deviation of the elements in $\mathcal{S}_l$, the intuition being that the Gaussian distribution captures what sizes are reasonable for large tracks to have in the dataset. If for the track in question T_q, $S(T_q)$ is above the 95% Gaussian tail, and it is low confidence ($c_{l,q} < h_q$), T_q is removed from the track set. RCT also removes redundant tracks, i.e. where the average IoU between two tracks is greater than RCT parameter h_u.

Finally, RCT trims the ends of tracks (which has a large impact on scores, see our ablation study). Specifically, RCT stops tracking objects when the width of the box is offscreen by more than ω percent of the frame width, and the height of the box is offscreen by more than ω percent of the frame height where ω is an RCT parameter. When an object is moving offscreen, RCT applies constant acceleration of $\alpha \vec{v}^k$ to the Kalman-derived velocity vector $\vec{v}^k$ to move the track swiftly offscreen, where α is an RCT parameter. Additionally, to avoid incorrect extrapolation of the tracks by the Kalman filter, RCT trims all boxes that are based on missing Kalman observations at the tail ends of the tracks as long as there are at least δ_n boxes, where δ_n is an RCT parameter.

5. FISHTRAC Dataset

5.1. A high-resolution MOT fish dataset

Real-world underwater fish tracking is a particularly challenging MOT problem. Fish move unpredictably, change appearance, and are frequently occluded. When video is collected by divers, additional complicated motion and parallax effects arise; additionally, fish often intentionally try to swim away or hide from the diver. And yet fish tracking is an important task in marine science, for instance to aid in studies of fish behavior, and also has recreational applications.

We present FISHTRAC, which is, to our knowledge, the first high-resolution fish dataset designed for multi-object tracking. FISHTRAC contains 14 videos totaling 3449 fully-annotated frames of real-world underwater video. Annotators were instructed that, if a fish is unambiguously identifiable in at least one frame of video, it should be annotated for all frames that it is believed to be within the camera's Field of View (FOV). This results in 131 total individual fish annotated (5–20 per video). Video is in high-resolution 1920x1080 (or higher) format collected at 24 frames per second, see Fig. 2 for an example. The videos were collected off the coast of Hawai'i island, primarily by a SCUBA diver, although we also include a video collected by a snorkeler and a video from a stationary camera. To simulate tracking with scarce data, just 3 videos are designated for training, the other 11 are reserved for testing. Likewise, when training on UA-DETRAC, we use just 3 videos from the train set (MVI_41073, MVI_40732, MVI_40141).

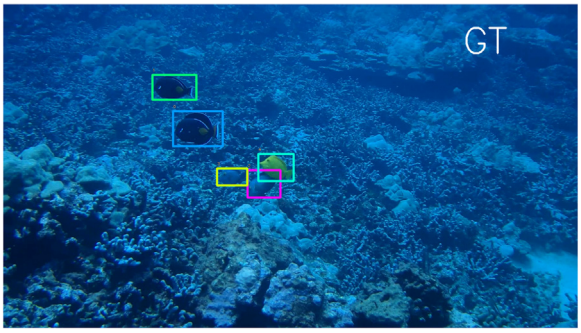

Fig. 2. FISHTRAC frame marked with ground truth (GT).
A frame of real-world underwater video with 5 fish marked, some of which are relatively difficult to see due to the complex coral reef background.

To test generalization, we test trackers on the MOT17 dataset, specifically the seven train videos (the test set annotations have not been publicly released and so the test set cannot be processed through our evaluation pipeline). This is a test of how well trackers can generalize to a different dataset, as we did not examine any pedestrian data during development of RCT, and we likewise do not tune any parameters or train any models on this dataset for the other methods (although some of them examined this dataset during their development, we would expect this to only strengthen their results).[4]

Additionally, we present the FISHTRAC codebase which includes everything needed (conversion/visualization scripts, etc.) to evaluate 17 tracking algorithms on a new MOT problem. Our code is based entirely on free technologies (GNU Octave and Python) and supports Linux. This should enable researchers to more easily compare MOT algorithms on new problems.

5.2. FISHTRAC object detection

In order to run a tracking-by-detection MOT pipeline on FISHTRAC, we need to train an object detector; however, this requires significant training data (even after pretraining the network on a general-purpose dataset like ImageNet [26]). Although manually annotating images is one option, that is time-consuming and expensive, and for many applications significant training data is available in large public datasets. Therefore, to generate training data for our FISHTRAC detectors, we scraped all human-annotated bounding boxes labeled "fish" from Google Open Images Dataset [27], one of the largest bounding box datasets. However, this resulted in only 1800 images (many of which were not in real-world underwater environments), which is more limited than the data usually used to train a deep learning model.

Next we examine different object detection approaches - although we wish to study cases where detections are low-quality, it is important to select a detection pipeline that maximizes the

[4] For methods that we trained a separate model for FISHTRAC and DETRAC (DAN and TransCenter), we tried both and selected the one with best performance on the MOT17 data.

Table 2
Precision, recall and mean average precision (mAP) of RetinaNet compared to YOLOv4 on FISHTRAC train.
RetinaNet is compared to YOLOv4 at 1024x1024 and 608x608, as well as YOLOv4 Tiny at 608x608. Columns are precision at a 0.5 confidence threshold, Recall at a 0.5 confidence threshold, and the mAP. RetinaNet achieves the best on all three measure, with 85.83% precision, 42.25% recall, and 60.41 mAp.

Algorithm	Precision @ 0.5	Recall @ 0.5	mAP
RetinaNet	**85.83**	**42.25**	**60.41**
YOLOv4 - 1024x1024	76.90	16.30	25.11
YOLOv4 - 608x608	83.89	18.51	30.40
YOLOv4 Tiny - 608x608	69.94	23.64	39.32

quality of detections given our limited training data. Therefore, we compared state-of-the-art detectors on our FISHTRAC set and selected the one with the best performance. Specifically, we compared the RetinaNet [28] architecture to variants of YOLOv4 [29]. For RetinaNet, we selected a ResNet50 backbone pretrained on ImageNet [26], and trained it for 10 epochs. For YOLOv4, we used the officially published model architectures (both full size and tiny variants).

We followed the official guide in the GitHub repo[5] to train the YOLO models on our custom objects, using the pretrained MSCOCO weights. RetinaNet rescales images to between 800–1000 pixels on each side, whereas YOLOv4 by default rescales its input to 608x608, so we also tried a variant of YOLOv4 with a larger input data size (1024 x 1024).

We evaluated the various object detection models on the FISHTRAC train dataset, using an IoU threshold of 0.5. As one can see from the results in Table 2, RetinaNet significantly outperforms the more recent YOLOv4 model on FISHTRAC data. Hence we use RetinaNet as our detector in all further experiments. Still, the resulting RetinaNet detector still has mediocre performance on FISHTRAC train: at a 0.5 confidence threshold, it has 85.53% precision and just 42.25% recall (60.4 mAP).

Detection for UA-DETRAC: For DETRAC, one can train an accurate detector given the ubiquity of vehicle data; but we intentionally trained on limited data to realistically simulate low-quality detections. Specifically, we trained RetinaNet with 200 car images from Open Images. We then trained the same RetinaNet architecture used in FISHTRAC. Unsurprisingly, this resulted in mediocre performance on our DETRAC train set: just 62% precision and 46% recall (50.3 mAP).

Detection for MOT17: For MOT17, one can likewise train an accurate detector given the ubiquity of pedestrian/person data, but we instead trained RetinaNet with 1800 person images from Open Images. This resulted in mediocre performance the MOT17 train set at 91% precision but only 24% recall (58.9 mAP).

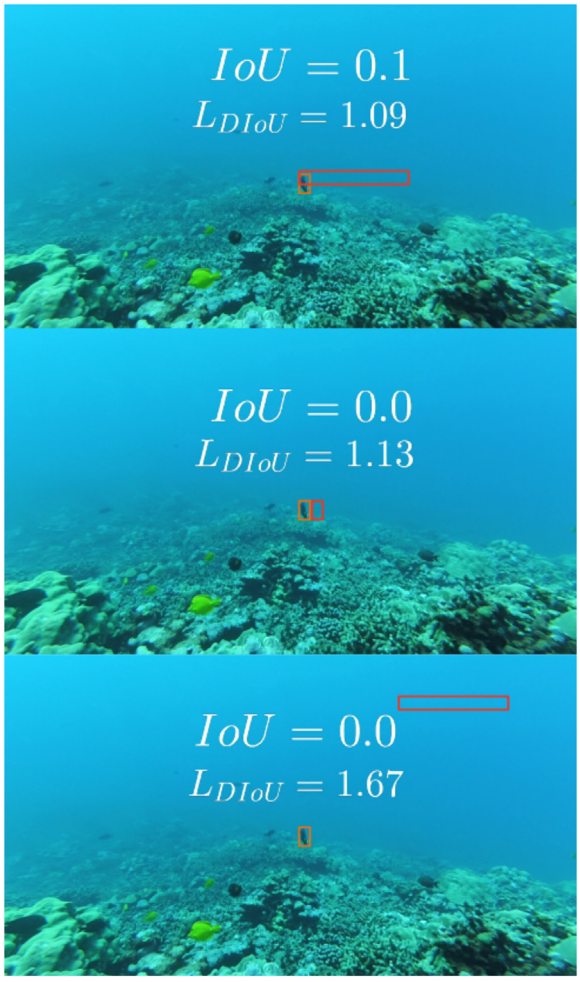

Fig. 3. The scores of DIoU and IoU in a real fish tracking scenario. The orange box is the ground truth and the red is the predicted (tracked) box. The middle image seems best from an end-use perspective, but using IoU there is no way to differentiate it from the bottom image, which is clearly much worse. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
Three scenarios are shown vertically, with IoU and DIoU scores posted for each scenario. In all three scenarios, a small ground truth box is correctly marked around a fish in the center of the screen. In the topmost scenario, we see a much larger predicted box, which is in the wrong position but overlaps slightly with the ground truth box. In this case, the IoU is 0.1 and the DIoU is 1.09. In the middle scenario, we see that the predicted box is the right shape and size but slightly to the right of the ground truth box, so they don't overlap. The IoU is 0.0 and the DIoU is 1.13. In the bottom secnario, the predicted box is the completely wrong size, shape, and location - the IoU is still 0.0 but the DIoU is now 1.67, providing a way to differentiate it from the middle scenario.

6. Experiment setup

6.1. Evaluation metrics

As a primary metric we use the recent HOTA metric [30], which has gained popularity due to its strong performance in user studies [30]. But we also report more classic CLEAR MOT metrics like MOTA [31] as secondary metrics.

However, one limitation of these MOT evaluation metrics is that they are both based on the IoU (intersection over union) between each box in the predicted track and each box in the ground truth track. MOTA uses a fixed threshold on IoU to determine whether a ground truth and a predicted box are close enough to match, whereas recent improvements like HOTA take the average score over many possible thresholds. However, regardless of the exact threshold, if the predicted and the ground truth track do not overlap, the IoU value is zero. So a predicted box which does not overlap any ground truth box will count as a false positive. Indeed, in such situations the tracker would have received a better score if it had simply not tracked the object at all. The rationale behind this approach is that a tracker which completely loses track of the object should detect that it has failed and not issue a prediction so as not to confuse the downstream pipeline.

Although at first this seems reasonable, in our setting we found that this produced highly counter-intuitive results. Consider Fig. 3 for instance. The middle image is clearly better than the top image at giving the user a sense of where the fish is: it has roughly the right size box in roughly the right location, whereas with the

[5] https://github.com/AlexeyAB/darknet

top image the location and size of the target are both completely inaccurate. Yet with a low enough IoU threshold, the top image will count as a matched detection due to the overlap, while the middle image never will no matter the IoU threshold. Additionally, with the IoU metric, there is no way to differentiate the middle image from the bottom image, even though the middle prediction is clearly much more useful to an end-user than the bottom prediction. With low-accuracy detections, situations similar to the middle image will happen a lot: especially when targets become small (such as fish swimming away from the camera) the tracker may have to rely on a motion model rather than visual information to determine where the object is. In these situations, as long as the tracker produces a track that is "close" to the original it will still be helpful for downstream applications even if there is no overlap, especially for small targets.

Therefore, we instead use Distance-IoU (DIoU) [32], a recent metric that combines IoU with the normalized distance between the boxes to give more "partial credit" to non-overlapping detections. Specifically, DIoU computes

$$DIoU(b_1, b_2) = 1 - IoU(b_1, b_2) + \frac{d^2_{euclid}(C(b_1), C(b_2))}{g^2(b_1, b_2)}, \quad (5)$$

where g is the diagonal length of the smallest box enclosing the two boxes and C is the center point operator. Intuitively, this combines IoU with the normalized distance between the boxes. Since DIoU contains both distance and IoU as components, it is suitable for a wide range of different scenarios and domains, allowing the practitioner to select a threshold that best meets the needs of their setting.

DIoU ranges from between 0 to 2 (note that, unlike IoU, lower is better), and since in our setting detections are low-quality and the tracker may have to interpolate between sequences of missing detections or occlusions, we select a threshold greater than 1 but less than 2 to admit boxes that may not overlap but are still relatively close. Specifically, if two equally-sized boxes barely touch at a corner, they will have DIoU 1.25, so this is the value we use to initialize the track. DETRAC's CLEAR MOT implementation originally allowed 20% variability in the threshold to allow for more leeway while tracking the object; we followed this approach, allowing the DIoU to rise up to 1.5 while tracking an object. To ensure our HOTA and MOTA metrics were considering a similar range of DIoU values, we modified HOTA to integrate over DIoU values between 1.25 and 1.5. The resulting scores on the training set of FISHTRAC more closely matched our intuition than the IoU-based scores, for instance DIoU with these thresholds would successfully count the middle scenario in Fig. 3 as a match while excluding the much worse bottom scenario.

6.2. Evaluation protocol

Trackers fed low-accuracy detections might take an extremely long time, or might fail to produce any results. To handle the time issue, our code kills the tracker after 30 minutes have passed on a single video - this is recorded as a **timeout**. In contrast, a **failure** occurs when a tracker fails to produce any results at all for an entire video, usually due to an assumption in the original code not being met - e.g. assuming that there are detections on every frame.

Additionally, each tracker other than RCT requires setting h, the threshold on detection confidence. To compare to the strongest possible baselines, we tune this separately for each tracker. A robust tracker should never timeout or fail, so we first select the threshold(s) that minimize the sum of timeouts and failures over the DETRAC and FISHTRAC train sets. In the case of ties, we select based on average HOTA. We then use this threshold on the test videos.

After this process, all trackers had zero timeouts/failures on the training data, except for the three slowest methods (D3S, GMMCP, and IHTLS), which often timed out.

6.3. RCT and baseline implementations

RCT Implementation Details: Like most other MOT algorithms, RCT has a number of parameters. In our case, other than $h_l = 0.5$ which was set purely based on intuition, we set the other 10 parameters to maximize MOTA and qualitative performance on the 6 FISHTRAC/DETRAC training videos. This resulted in the following settings: $\beta = 50\%$, $\delta = 4$, $\delta_m = 2$, $h_u = 0.3$, $D_{\max} = 20$, $h_q = 0.8$, $h_f = 0.2$, $\omega = 1\%$, $\alpha = 1.1$, and $\delta_n = 5$. The majority (7 of 10) of these parameters control the trimming and joining heuristics (see ablation study for the impact of these components). A list of the RCT parameters used in our experiments are provided in Table 3. In terms of Kalman filter parameters, we set the transition and observation covariance matrices to standard 1-diagonal form (with 0 elements for the velocity observations since they are unobserved), although we did a small amount of tuning on the diagonal velocity transition elements (which were set to 0.2), and the diagonal position observation elements (which were set to 0.5).

Classic and Specialized Baselines: In total, we compare RCT to 16 trackers. We compare to four classic trackers from the original DETRAC set (**GOG** [33], **CMOT** [34], **RMOT** [35], and **IHTLS** [36]). To this we add **GMMCP** [25], a tracker used in recent video-based person re-identification systems [37]. We compare two related improvements of the IOU tracker [1], **KIOU** [48] (which uses a Kalman filter) and **VIOU** [17] (which uses MedianFlow). We compare **JPDA_m** [21], an optimization of the classic JPDA approach [20] that, like RCT, incorporates motion model probability. We also compare to Visual Fish Tracking (**VFT**) [11], which is specially designed to track fish in real-world video.

Deep Baselines: We compare to **DAN** [7], which has exceptional performance on UA-DETRAC and MOT17. We fine-tuned DAN on our train set (using the provided pretrained pedestrian model, which outperformed the base model) to maximize performance. We also compare **AOA** [38], which won the recent 2020 ECCV TAO challenge and uses an improved version of the popular DeepSORT [39] algorithm. Finally, we compare to **TransCenter** [9], a recent deep transformer-based approach which (as of June 2022) ranks in the top 20 in the MOT17 challenge and the top 10 in the MOT20 challenge. TransCenter trains its own (essentially end-to-end) object detection and tracking pipeline based on the DETR [40] object detector. We fine-tuned TransCenter on our train set (using the provided pretrained base model, which outperformed the pedestrian model), after which it achieved a near-perfect 92 HOTA on our training data.

SOT Baselines: Comparing to SOT approaches is unusual in the MOT literature; however, SOT approaches rely less on detection quality and thus may be a viable approach in this setting. We adapt these approaches to the MOT setting in a way that mirrors RCT: we initialize the tracker on the highest confidence detection that does not overlap previous tracks, and run the tracker forward and backward from that frame; continuing to add tracks while there are still uncovered detections. As SOT trackers, we try **MedianFlow** [24] and **KCF** [41], which have shown good performance in MOT pipelines [17]. For KCF, we first downscale the video to 300x300 pixels as we observed that this improved performance, whereas for MedianFlow we use the original video resolution to mirror the way it is used inside RCT. We also compare **D3S** [42], a deep segmentation approach which is one of the top performers on the recent real-time VOT-RT 2020 challenge [43]. However, even "real-time" SOT trackers may be slow when applied to the more complex MOT task. Therefore, we compare to **GOTURN** [44],

Table 3
RCT parameters and meaning.
Eleven parameters of RCT are listed, along with their meanings.

Parameter	Meaning
h_l	Threshold for track initialization - we will not initialize tracks on a detection with lower confidence than this.
β	Percent box is enlarged to check if it is sufficiently far from image edge.
δ	Number of previous frames used to calculate approximate position and velocity using Kalman filter.
δ_m	Number of boxes needed to justify switching from Kalman filter to median flow.
h_u	IoU threshold to determine if two detections are potentially on the same object
$D_{\max}$	Maximum number of frames with missing detections to permit joining two tracklets.
h_q	Confidence threshold used to separately analyze "high-quality" tracks during the track filtering step.
h_f	IoU threshold used to filter redundant tracks.
ω	Percentage offscreen an object must be in order to trim its track.
α	Acceleration factor when objects are moving offscreen.
δ_n	Number of frames of missing detections needed before deciding to trim them from track.

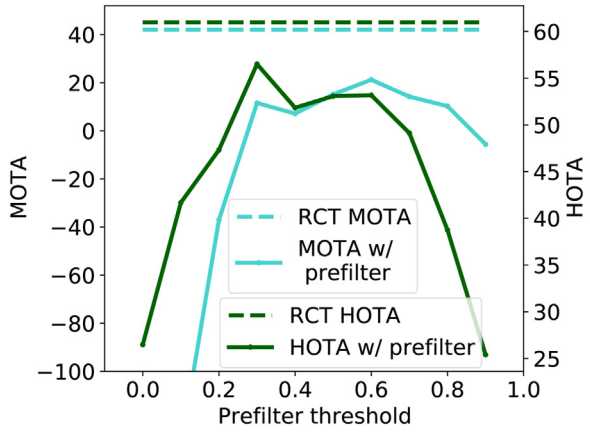

Fig. 4. RCT on FISHTRAC train with/without a prefilter.
A graph is shown - the x axis varies the prefilter threshold between 0 and 1, the left y axis shows the MOTA score, and the right y axis shows the HOTA scores. Four lines are shown: RCT MOTA (constant), RCT MOTA with prefilter, RCT HOTA (constant), and RCT HOTA with prefilter. The prefilter curves are roughly inverted U shapes, with values between 0.2 and 0.7 working best, but even the best values are still roughly 10 points worse in MOTA/HOTA than the unmodified RCT.

a deep tracker which ranked #1 in terms of speed and #6 of 39 in accuracy on the large-scale GOT-10k benchmark [45].

7. Results

7.1. RCT performance analysis

One of the key aspects of RCT is its use of the exact detection confidence, instead of "prefiltering" the detections by a fixed confidence threshold, and then discarding the confidence. Fig. 4 shows a comparison of RCT to a variant with an initial prefilter. No matter how we set the threshold, we cannot reach the original performance, showing the benefit of utilizing the exact detection confidence when tracking.

RCT is capable of efficiently searching through an unfiltered set of detections to produce an effective track. Table 4 shows that this is usually not the case for other trackers. Several methods could not cope with the large number of unfiltered detections, being unable to return a result even after three days.[6] In contrast, RCT was able to quickly produce accurate tracks in this setting, while the methods that completed generally showed poor performance. For instance, one of the fastest methods, MEDFLOW, is 3x slower than RCT and generates significantly negative MOTA scores. TransCenter is one of the only other methods that is able to quickly achieve good results in the UA-DETRAC case; however, this due to the fact that it largely discards the original detections in favor of running its own detector (the provided detections are used only as a filter on where to start a track). While this works well on UA-DETRAC since we trained TransCenter on our DETRAC train set, TransCenter cannot generalize to an environment different from what it was trained on, as shown by the near-zero HOTA and MOTA scores on pedestrian data. In contrast, RCT is able to quickly produce accurate tracks in both settings.

Given that our method contains several non-essential components, we ran an ablation study to determine the impact of each factor on performance as well as on speed. The results are shown in Table 5. From the first five rows of the table, we observe that removing any of the various features of RCT does result in a decrease in the training data HOTA and MOTA, and typically an increase in ID switches. Additionally, we observe that the majority of the time spent is on the MedianFlow tracker, with a significant percentage of that being loading the images (MedianFlow is the only aspect of RCT that uses visual information). This could likely be optimized, for instance by lowering the input resolution or loading images in parallel. We did find it surprising that the precise method of track trimming had such a large impact on performance, as shown in the last three rows of the table - for discussion of this, see Section 8.3.

7.2. Test results

We ran all 17 trackers across all 11 FISHTRAC test videos, the 40 UA-DETRAC test videos and the 7 MOT17 videos for which ground truth was available. We followed good practice regarding test data, in particular, we did not in any way evaluate RCT on the test videos during its development, with the exception of optimizing and reformatting the code in ways that did not affect the tracks produced. Our objective is the same as it was when selecting thresholds: we wish to minimize timeouts and failures for reliability, and then to maximize the combined HOTA score. FISHTRAC comprises 50% of the combined score (since it is a more realistic example of having to cope with low-quality detections compared to the other two settings), and the remaining 50% is split evenly between UA-DETRAC and MOT17.

Test results are shown in Table 6. Our main result is that, of the trackers which successfully produced results for every sequence (i.e. no timeouts or failures), our RCT algorithm has the best combined HOTA across the three datasets. Additionally, RCT achieves by

[6] The methods were either run on a state-of-the-art high-performance computing cluster, or on a modern GPU-capable server, depending on their hardware and software requirements.

Table 4
Performance when trackers are fed unfiltered detections for one DETRAC video (MVI_40752, 2025 frames) and one MOT17 video (MOT17-04, 1050 frames).
All 17 trackers compared on the DETRAC and MOT17 sequence given unfiltered detections, showing time taken, HOTA, MOTA, and ID Switches. RCT takes 2 minutes on the DETRAC sequence and 4 minutes on the MOT17 sequence. TRANSCTR takes similar amounts of time (4 min on UA-DETRAC and 3 min on MOT17), and even has better HOTA than RCT on the UA-DETRAC sequence (65.0 vs 51.44), but performs very poorly on the MOT17 sequence (HOTA of 10.13 and MOTA of 0.9). The next fastest tracker is MEDFLOW, but it takes three times the time of RCT (6 minutes on DETRAC and 12 min on MOT17) with many more identity switches. The next fastest tracker is KCF with 57 minutes on DETRAC and 105 minutes on MOT17. It only gets worse from there, with VFT, RMOT, IHTLS, CMOT, and GMMCP all taking more than three days to return results on the DETRAC sequence. This shows that existing trackers are not capable of efficiently extracting useful data from unfiltered detections.

	DETRAC sequence				MOT17 sequence			
Tracker	**Time**	**HOTA**	**MOTA**	**ID switches**	**Time**	**HOTA**	**MOTA**	**ID switches**
RCT	2 min	51.44	36.78	8	4 min	34.08	26.08	26
TRANSCTR	4 min	65.00	62.89	139	3 min	10.13	0.90	0
MEDFLOW	6 min	45.47	−8.21	88	12 min	40.49	-19.47	37
KCF	57 min	44.89	2.13	307	105 min	38.20	29.63	688
DAN	61 min	13.38	−757.63	843	28 min	30.47	11.13	1578
VIOU	155 min	12.49	−984.70	219	58 min	15.49	−249.50	170
KIOU	334 min	6.06	−4126.46	330	43 min	14.11	−875.87	181
GOG	452 min	15.40	−828.10	367	119 min	20.31	−144.03	434
GOTURN	524 min	2.84	−5499.44	189	816 min	9.47	−1905.40	127
AOA	677 min	6.74	−2219.49	572	149 min	12.69	−907.29	262
D3S	1010 min	10.09	−951.38	125	130 min	38.20	−18.01	33
$JPDA_m$	2483 min	9.50	−1688.34	835	2 min	4.50	1.28	1
VFT	> 3 days	–	–	–	> 3 days	–	–	–
RMOT	> 3 days	–	–	–	> 3 days	–	–	–
IHTLS	> 3 days	–	–	–	> 3 days	–	–	–
CMOT	> 3 days	–	–	–	Failed	–	–	–
GMMCP	> 3 days	–	–	–	Failed	–	–	–

Table 5
Ablation study. HOTA and MOTA are averaged over the two train datasets; ID switches are summed. The first 5 rows explore the impact of excluding components of RCT on performance and time, while the last three rows show the impact of trimming on HOTA and MOTA scores.
Ablation study, showing modifications of RCT, their HOTA and MOTA scores, ID switches, and FPS. We test removing features from RCT and find that they hurt the scores but improve the speed. The original average HOTA score is 60.61 and FPS is 23.37. Exlcuding loading time keeps the HOTA the same but increases the FPS to 30.51. Excluding MedianFlow drops the HOTA to 55.93 but raises the FPS to 60.20. Excluding MedianFlow+ Track joining drops the HOTA to 51.76 but raises the FPS to 64.57. Excluding MedianFlow+ Track joining + long-large filtering drops the HOTA to 50.27 and increases the FPS to 65.11. The final three rows show the impact of trimming: three trimming types are tested (Not trimming when box is offscreen, Trimming as soon as box touches offscreen, Not trimming when box is fully onscreen) and all three significantly drop the HOTA and MOTA scores.

Variation	Avg HOTA	Avg MOTA	Total ID Switches	FPS
Unmodified	60.61	45.86	32	23.37
Excluding image loading from timing	60.61	45.86	32	30.51
No MedianFlow	55.93	41.24	39	60.20
No MedianFlow + no track joining	51.76	39.37	44	64.57
No MedianFlow + no track joining + not filtering long, large, low confidence tracks	50.27	25.86	83	65.11
Not trimming when box is offscreen	25.61	−608.37	38	24.31
Trimming as soon as box touches offscreen	57.06	36.65	23	24.35
Not trimming when box is fully onscreen	58.37	32.91	34	24.38

far the lowest number of total identity switches of methods that completed on the majority of videos.

8. Discussion

8.1. RCT performance

The fact that the RCT algorithm has the best combined HOTA of all methods without timeouts or failures demonstrates the robust performance of the algorithm. Many other trackers were not nearly as robust - for instance, while CMOT has an impressive HOTA score on the FISHTRAC dataset, it cannot cope with the longer DETRAC sequences, resulting in 5 timeouts and 1 failure. This is particularly notable for three reasons:

1. RCT was developed based on examining its performance on just 6 videos, in contrast to other methods where development typically involved a much larger set of videos (such as the full 60-sequence DETRAC train set).
2. For all other algorithms, which filter detections by a threshold, we adjusted the threshold so as to maximize performance on our train set. RCT must process all unfiltered detections, and the threshold used for the initial box of a new track (h_l) is set purely based on intuition.
3. Although RCT uses classic techniques for many components (MedianFlow for visual object tracking, Kalman filters for the motion model, greedy track joining), it can outperform more sophisticated classic approaches as well as more recent deep learning methods such as DAN, AOA, and TransCenter. The TransCenter results exemplify the challenges deep trackers face in this setting. TransCenter performed near-perfectly on the train data but poorly on the test data, in spite of employing mechanisms to combat overfitting (reserving validation data, etc.). While deep trackers perform very well in cases where large amounts of training data are available, our results show that they may not do so in cases where data is scarce. In constrast, RCT's use of detection confidence allows it to recover high-quality tracks without needing to learn a specialized model.

Table 6
Test set results for FISHTRAC (shorthand: Fish), DETRAC (shorthand: Car), and MOT17 (shorthand: Ped). Timeouts and Failures are summed across the datasets, while the combined HOTA and FPS are computed as a weighted average with 0.5 wt for FISHTRAC and 0.25 each for the other two datasets. The table is sorted first by the sum of timeouts and failures, and second by the combined HOTA. Bolded values indicate the best scores of trackers that produced results on all sequences.
This table shows all 17 trackers in rows with columns Timeouts, Failures, Combined HOTA, Total ID Switches, Fish HOTA, Fish MOTA, Fish ID switches, Car HOTA, Car MOTA and Car ID switches, Pedestrian HOTA, Pedestrian MOTA and Pedestrian ID switches. KIOU, VIOU, JPDA_m, GOTURN, CMOT, RMOT, VFT, D3S, GMMCP, and IHTLS all timed out or failed on at least one video. Of the trackers that succeeded on all videos, RCT has the best combined HOTA (42.67), total ID switches (741) Fish HOTA (49.67), Fish MOTA (45.97), Fish ID switches (47), and Car ID switches (506). KCF has the best Car HOTA (55.87) and MOTA (47.86). MEDFLOW has the best Pedestrian HOTA (40.68). GOG has the best Pedestrian MOTA (36.40) and Combined FPS (103.81). TransCenter has the fewest pedestrian ID switches (53), but this is likely because it produced almost no tracks, as shown by the very low scores on that dataset (HOTA of 7.43 and MOTA of 0.36). Other trackers typically have many more ID switches than RCT, for instance the second-ranked in terms of combined HOTA is DAN, but it has 18,954 ID switches, compared to 741 for RCT.

Tracker	Timeouts	Failures	Comb. HOTA	Total ID Sw	Fish HOTA	Fish MOTA	Fish ID Sw	Car HOTA	Car MOTA	Car ID Sw	Ped HOTA	Ped MOTA	Ped ID Sw	Comb. FPS
RCT	**0**	**0**	**42.67**	**741**	**49.67**	**45.97**	**47**	39.49	29.6	**506**	31.84	27.21	188	11.39
DAN	**0**	**0**	41.23	18,954	44.24	42.05	361	39.8	35	16,892	36.64	32.19	1701	3.63
MEDFLOW	**0**	**0**	39.65	973	32.03	-58.51	108	53.87	33.67	692	**40.68**	33.28	173	26.12
KCF	**0**	**0**	37.81	4355	30.45	27.75	884	**55.87**	**47.86**	2679	34.50	34.45	792	17.18
GOG	**0**	**0**	37.61	17,764	37.85	45.08	414	40.98	39.42	15,459	33.76	**36.40**	1891	**103.81**
AOA	**0**	**0**	35.69	23,592	39.28	13.79	593	31.14	3.6	20,255	33.04	30.81	2744	11.10
TRANSCTR	**0**	**0**	21.24	6431	20.21	21.11	174	37.12	38.94	6204	7.43	0.36	**53**	8.45
KIOU	0	1	45.7	5685	49.47	46.72	119	43.81	31.64	4990	40.04	36.78	576	174.49
VIOU	0	1	44.53	3009	48.91	46.44	51	42.81	35.39	2717	37.49	34.67	241	4.41
$JPDA_m$	0	1	35.17	1701	34.11	35.75	77	41.47	32.99	1280	31.00	28.69	344	16.98
GOTURN	0	2	20.73	1230	19.53	-281.09	114	21.41	-88.88	988	22.48	-21.09	128	7.65
CMOT	5	1	47.42	6487	54.4	50.3	110	38.82	12.71	5621	42.05	41.40	756	2.50
RMOT	6	0	36.16	1210	39.74	40.21	133	36.82	25.77	944	28.34	18.34	133	4.57
VFT	9	0	23.94	5723	30.73	33.93	449	16.25	16.45	4808	18.03	13.23	466	12.81
D3S	26	1	38.05	150	54.72	23.2	33	13.5	2.15	49	29.27	9.13	68	0.81
GMMCP	36	12	16.07	138	29.76	31.1	114	4.75	0.4	24	0	0	0	0.16
IHTLS	52	1	6.8	242	13.6	-2.54	242	0	0	0	0	0	0	0.08

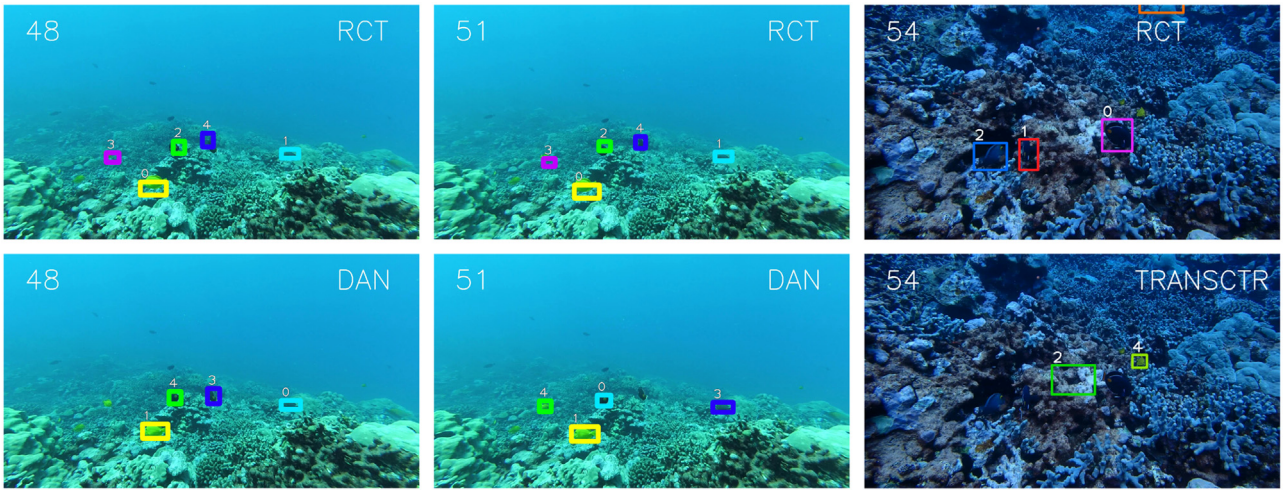

Fig. 5. The left four images show the performance of RCT and DAN on two frames of a FISHTRAC training video, while the right two images show the performance of RCT and TransCenter on a FISHTRAC test video. RCT shows good performance in both cases, whereas DAN causes several identity switches and TransCenter struggles to reliably localize the fish. (Please see the online version for a color version of this figure, which may assist with the interpretation.)
A 3x2 array of images is shown. The first two images in the first row show RCT's performance on frames 48 and 51 of a FISHTRAC Train video. The first two images in the second row shows DAN's performance on the same two frames. RCT has no identity switches here, but DAN has swapped almost all identities, for instance taking a fish with label 0 on frame 48 and changing it to label 3 on frame 51. The final column of images shows RCT vs TransCenter on a FISHTRAC test video. RCT has correctly marked three large fish, but TransCenter has not marked these fish and instead marked one very small fish and one large box that is completely over coral.

One of the most notable features of our RCT algorithm is how it achieves just 741 identity switches across all 58 test videos - the only algorithms with fewer are D3S, GMMCP, and IHTLS, algorithms that simply did not produce any tracks for the majority of videos. The other multi-object trackers have an order of magnitude more total identity switches, even algorithms such as DAN, VIOU, and KIOU which achieve good HOTA. The reduction in ID switches makes a significant visual difference in the quality of the produced tracks - see Fig. 5 for an example. The small number of ID switches is due to RCT's ability to fuse low-confidence detections, a motion model, and a single object tracker to rapidly produce high-quality continuous tracks even when high-confidence detections are sparse. Minimizing ID switches is very important for practical applications - for instance, we intend to use RCT to help divers keep track of individual fish while underwater. Numerous ID switches are likely to confuse the diver and cause them to follow the incorrect fish. In these types of applications, we would much rather miss some objects, but ensure the tracks we do provide are high-quality, with little to no identity switches, even in the face of unreliable detections; we expect RCT to excel in these situations.

8.2. Single object trackers

Most multi-object trackers do not compare to single object trackers as the tasks are quite different - SOT approaches are evaluated on different metrics and make different assumptions. How-

ever, we found that, surprisingly, SOT approaches perform quite well here. Our adaptation of the KCF single-object tracker achieves the best HOTA score on the DETRAC dataset, and our adaptation of the MEDFLOW tracker achieves the best HOTA score on MOT17. The fact that KCF and MEDFLOW perform so well in these settings highlights the importance of comparing to SOT algorithms even when attempting to solve a MOT problem. The reduced reliance on detection quality compared to MOT algorithms helps significantly when objects are visually distinct. Moreover, our results highlight the importance of comparing trackers on multiple datasets: The SOT approaches perform much worse on FISHTRAC, which confirms our intuition that fish are more difficult to track based purely on visual information, since they can change appearance dramatically from frame to frame.

It is also surprising that KCF and MEDFLOW, which are typically thought to be quite low baselines when it comes to SOT algorithms, outperform stronger (deep) single object trackers such as GOTURN and D3S. Our experiments indicate that stronger SOT trackers like D3S (despite being "real-time") are too computationally expensive to run on MOT problems with low-quality detections - in fact, D3S timed out on almost half of the test videos. This is consistent with past work which has observed that deep SOT approaches have insufficient speed when tracking multiple objects simultaneously [46]. GOTURN is an exception as it has sufficient speed, but performs poorly, in part due to not adequately handling MOT-specific issues such as track termination.

8.3. Limitations of HOTA scores

Our main evaluation metric is HOTA scores, but a close examination of Tables 5 and 6 reveals some potential weaknesses in the scores. First, Table 5 shows that the precise method of track trimming (that is, stopping tracks when an object goes offscreen or is not visible) has a major impact on the HOTA and MOTA scores of our method. For instance, the scores penalize our tracker greatly if it does not make a track disappear in the middle of the screen when the tracker loses track of an object. But in a real-world scenario such as fish tracking, we know an object cannot instantly disappear and so it may be more useful to give the end user an estimate of its position even if that estimate is imperfect. Additionally, Table 6 shows that algorithms with a very large number of ID switches can still achieve high HOTA and MOTA scores (e.g. on our test data, DAN with 18,954 ID switches has a better HOTA score than MEDFLOW which has just 973 ID switches). As such, although we concur with Luiten et al. [30] that HOTA represents a significant improvement over MOTA, we feel that it is important to create better MOT metrics that match human intuition regarding MOT results.

8.4. Limitations

A seeming limitation of RCT is that it is outperformed by MEDFLOW on the MOT17 dataset: MEDFLOW has a similar number of ID switches but a significantly higher HOTA score on that dataset. However, MEDFLOW has abysmal performance on the FISHTRAC dataset, as evidenced by the fact that the MOTA score goes significantly negative in this case. The underlying reason for this is that MEDFLOW's performance relies on the ability to distinguish foreground from background, which is easier in the car and pedestrian cases (constant lighting, fixed camera, etc.). Qualitatively, MEDFLOW often tracks background objects in FISHTRAC (i.e. coral instead of fish). This motivates RCT's careful use of MEDFLOW to achieve good results on FISHTRAC - and while the performance on MOT17 is still quite strong, our results suggest that RCT may be able to be improved further in the case of high target density.

Although RCT makes use of the full detection confidence when assigning detections to tracks, it still thresholds detections by $h_l = 0.5$ in order to determine where to initialize tracks. This is a limitation, as for instance, if a detector somehow produced only detections below h_l, no tracks would be produced. This is currently necessary to ensure that we only start tracking objects that are more likely than not to belong to the target class, thereby avoiding spurious tracks. It would be interesting to investigate alternative methods, for instance, the tracker could make the decision whether to accept or reject a track based on the relationship between track size, length, and average detection confidence, in a similar manner to our long-large filtering step, although this might increase the computational expense.

While RCT does achieve good HOTA and low ID switches, it does miss some tracks. This is supported by the precision and recall results (found alongside the full set of MOTA/HOTA metrics at https://github.com/tmandel/fish-detrac) - RCT consistently leans towards high precision over high recall, achieving 84% precision (with 57% recall) on FISHTRAC test data, 94% precision (with 32% recall) on DETRAC test data, and 93% precision (with 30% recall) on the MOT17 data. This matches with the fact that our intended application domains, high precision and low ID switches are more important than high recall - we want to produce high-quality tracks that do not confuse a diver, even if that means a few small fish may be missed. Nevertheless, improving the recall of our system (for instance by the aforementioned method of initializing tracks on lower-confidence detections) is a direction for future work.

We focus on the problem of offline multi-object tracking; this is a limitation as in many real-world applications, online tracking is more useful. We feel that understanding how to perform offline MOT in the presence of unreliable detections is an important goal in and of itself, as well as a first step to solving online MOT problems in this setting. We also note that standard MOT testbeds like UA-DETRAC focus on the offline setting, and most of the algorithms we compare to are offline trackers. We believe it is possible to adapt RCT to the online setting, for instance by applying the motion model update only in a single direction, but this is left for future work.

9. Conclusion

We have studied the problem of offline multi-object tracking-by-detection with unreliable detections. To illustrate this, we presented a new MOT dataset, FISHTRAC, with high-resolution videos of underwater fish behavior. We also present RCT, which takes a different approach than other MOT algorithms, using the detection confidence to guide an efficient search through a completely unfiltered set of input detections. Despite the fact that RCT relies on simple heuristics such as greedy track agglomeration, we find that RCT outperforms baselines (including the 2020 TAO challenge winner, top performer on the VOT-RT 2020 challenge, and a recent top performer on the MOT17 and MOT20 challenges), tracking objects accurately with very few ID switches and no timeouts or failures. In addition to releasing our FISHTRAC data, we will release the code to our test harness, allowing other MOT researchers to easily compare different trackers on new datasets.

One practical benefit of RCT is that it does not use a GPU, which in edge settings may be fully utilized by the detection network - future work includes implementing and evaluating an online version of RCT in these settings. Also, while RCT does probabilistically integrate motion and detections, performance in high-density settings could likely be further improved by probabilistically incorporating appearance information in a Bayesian fashion [47]. Additionally, we found many of the top-ranked MOT methods work poorly with a low-quality detector; so it would be interesting to

explore an adaptive approach which analyzes detection quality and adapts the tracker behavior accordingly. Pursuing these directions will help MOT algorithms be more easily deployed to solve a diverse set of real-world problems.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We gratefully acknowledge the assistance of Jennifer Nakano, Timothy Kudryn, Ilya Kravchik, Dr. Timothy Grabowski, Christopher Hanley, and Sebastian Carter on this research project. This work was supported by NSF CAREER Award #HCC-1942229 and NSF EPSCoR Award #OIA-1557349.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patcog.2022.109107

References

[1] E. Bochinski, V. Eiselein, T. Sikora, High-speed tracking-by-detection without using image information, International Workshop on Traffic and Street Surveillance for Safety and Security at IEEE AVSS 2017, Lecce, Italy, 2017. http://elvera.nue.tu-berlin.de/files/1517Bochinski2017.pdf
[2] P. Dendorfer, A. Osep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth, L. Leal-Taixé, MotChallenge: a benchmark for single-camera multiple target tracking, Int J Comput Vis 129 (4) (2021) 845–881.
[3] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, S. Lyu, Ua-detrac: a new benchmark and protocol for multi-object detection and tracking, Comput. Vision Image Understanding 193 (2020) 102907.
[4] C. Spampinato, S. Palazzo, D. Giordano, I. Kavasidis, F.-P. Lin, Y.-T. Lin, Covariance based fish tracking in real-life underwater environment, in: VISAPP (2), 2012, pp. 409–414.
[5] X. Cao, S. Guo, J. Lin, W. Zhang, M. Liao, Online tracking of ants based on deep association metrics: method, dataset and evaluation, Pattern Recognit 103 (2020) 107233.
[6] F. Solera, S. Calderara, R. Cucchiara, Towards the evaluation of reproducible robustness in tracking-by-detection, in: 2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), IEEE, 2015, pp. 1–6.
[7] S. Sun, N. Akhtar, H. Song, A. Mian, M. Shah, Deep affinity network for multiple object tracking, IEEE Trans Pattern Anal Mach Intell 43 (1) (2019) 104–119.
[8] X. Zhou, V. Koltun, P. Krähenbühl, Tracking objects as points, in: European Conference on Computer Vision, Springer, 2020, pp. 474–490.
[9] Y. Xu, Y. Ban, G. Delorme, C. Gan, D. Rus, X. Alameda-Pineda, TransCenter: Transformers with dense representations for multiple-object tracking, 2021, arXiv:2103.15145
[10] J. Jäger, E. Rodner, J. Denzler, V. Wolff, K. Fricke-Neuderth, SeaCLEF 2016: Object proposal classification for fish detection in underwater videos, in: CLEF (working notes), 2016, pp. 481–489.
[11] J. Jäger, V. Wolff, K. Fricke-Neuderth, O. Mothes, J. Denzler, Visual fish tracking: combining a two-stage graph approach with cnn-features, in: OCEANS 2017-Aberdeen, IEEE, 2017, pp. 1–6.
[12] I. Kavasidis, S. Palazzo, R. Di Salvo, D. Giordano, C. Spampinato, An innovative web-based collaborative platform for video annotation, Multimed Tools Appl 70 (1) (2014) 413–432.
[13] A. Dave, T. Khurana, P. Tokmakov, C. Schmid, D. Ramanan, TAO: A large-scale benchmark for tracking any object, in: European Conference on Computer Vision, Springer, 2020, pp. 436–454.
[14] L. Yang, Y. Fan, N. Xu, Video instance segmentation, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 5188–5197.
[15] J. Qi, Y. Gao, Y. Hu, X. Wang, X. Liu, X. Bai, S. Belongie, A. Yuille, P.H. Torr, S. Bai, Occluded video instance segmentation: a benchmark, Int J Comput Vis 130 (8) (2022).
[16] M. Pedersen, J.B. Haurum, S.H. Bengtson, T.B. Moeslund, 3D-ZEF: A 3D zebrafish tracking benchmark dataset, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2426–2436.
[17] E. Bochinski, T. Senst, T. Sikora, Extending IOU based multi-object tracking by visual information, in: 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), IEEE, 2018, pp. 1–6.
[18] R.C. Verma, C. Schmid, K. Mikolajczyk, Face detection and tracking in a video by propagating detection probabilities, IEEE Trans Pattern Anal Mach Intell 25 (10) (2003) 1215–1228.
[19] M.D. Breitenstein, F. Reichlin, B. Leibe, E.K. meier Luc Van Gool, C.V. Laboratory, Robust tracking-by-detection using a detector confidence particle filter, ICCV, 2009.
[20] T. Fortmann, Y. Bar-Shalom, M. Scheffe, Sonar tracking of multiple targets using joint probabilistic data association, IEEE J. Oceanic Eng. 8 (3) (1983) 173–184.
[21] S.H. Rezatofighi, A. Milan, Z. Zhang, Q. Shi, A. Dick, I. Reid, Joint probabilistic data association revisited, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 3047–3055.
[22] S.-H. Bae, K.-J. Yoon, Confidence-based data association and discriminative deep appearance learning for robust online multi-object tracking, IEEE Trans Pattern Anal Mach Intell 40 (3) (2017) 595–610.
[23] X. Lin, C.-T. Li, V. Sanchez, C. Maple, On the detection-to-track association for online multi-object tracking, Pattern Recognit Lett 146 (2021) 200–207.
[24] Z. Kalal, K. Mikolajczyk, J. Matas, Forward-backward error: Automatic detection of tracking failures, in: 2010 20th International Conference on Pattern Recognition, IEEE, 2010, pp. 2756–2759.
[25] A. Dehghan, S. Modiri Assari, M. Shah, GMMCP tracker: Globally optimal generalized maximum multi clique problem for multiple object tracking, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 4091–4099.
[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2009, pp. 248–255.
[27] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, et al., The open images dataset v4, Int J Comput Vis 128 (7) (2020) 1956–1981.
[28] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár, Focal loss for dense object detection, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2980–2988.
[29] A. Bochkovskiy, C.-Y. Wang, H.-Y.M. Liao, Yolov4: optimal speed and accuracy of object detection, arXiv preprint arXiv:2004.10934 (2020).
[30] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, B. Leibe, HOTA: A higher order metric for evaluating multi-object tracking, Int J Comput Vis (2020) 1–31.
[31] K. Bernardin, R. Stiefelhagen, Evaluating multiple object tracking performance: the CLEAR MOT metrics, EURASIP J Image Video Process 2008 (2008) 1–10.
[32] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, D. Ren, Distance-IoU loss: Faster and better learning for bounding box regression, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 12993–13000.
[33] H. Pirsiavash, D. Ramanan, C.C. Fowlkes, Globally-optimal greedy algorithms for tracking a variable number of objects, in: CVPR 2011, IEEE, 2011, pp. 1201–1208.
[34] S.-H. Bae, K.-J. Yoon, Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1218–1225.
[35] J.H. Yoon, M.-H. Yang, J. Lim, K.-J. Yoon, Bayesian multi-object tracking using motion context from multiple objects, in: 2015 IEEE Winter Conference on Applications of Computer Vision, IEEE, 2015, pp. 33–40.
[36] C. Dicle, O.I. Camps, M. Sznaier, The way they move: Tracking multiple targets with similar appearance, in: Proceedings of the IEEE international conference on computer vision, 2013, pp. 2304–2311.
[37] X. Jiang, Y. Qiao, J. Yan, Q. Li, W. Zheng, D. Chen, SSN3D: Self-separated network to align parts for 3D convolution in video person re-identification, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021.
[38] F. Du, B. Xu, J. Tang, Y. Zhang, F. Wang, H. Li, 1st place solution to ECCV-TAO-2020: detect and represent any object for tracking, arXiv preprint arXiv:2101.08040 (2021). https://arxiv.org/abs/2101.08040
[39] N. Wojke, A. Bewley, Deep cosine metric learning for person re-identification, in: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2018, pp. 748–756.
[40] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, J. Dai, Deformable DETR: Deformable transformers for end-to-end object detection, in: International Conference on Learning Representations, 2020.
[41] J.F. Henriques, R. Caseiro, P. Martins, J. Batista, High-speed tracking with kernelized correlation filters, IEEE Trans Pattern Anal Mach Intell 37 (3) (2014) 583–596.
[42] A. Lukezic, J. Matas, M. Kristan, D3S - a discriminative single shot segmentation tracker, CVPR, 2020.
[43] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, J.-K. Kamarainen, L. Čehovin Zajc, M. Danelljan, A. Lukezic, O. Drbohlav, L. He, Y. Zhang, S. Yan, J. Yang, G. Fernandez, et al., The eighth visual object tracking VOT2020 challenge results, European Conference on Computer Vision, 2020.
[44] D. Held, S. Thrun, S. Savarese, Learning to track at 100 FPS with deep regression networks, in: European Conference on Computer Vision, Springer, 2016, pp. 749–765.
[45] L. Huang, X. Zhao, K. Huang, Got-10k: a large high-diversity benchmark for generic object tracking in the wild, IEEE Trans Pattern Anal Mach Intell (2019).
[46] L. Vaquero, V.M. Brea, M. Mucientes, Tracking more than 100 arbitrary objects at 25 fps through deep learning, Pattern Recognit 121 (2022) 108205.
[47] D.Y. Kim, B.-N. Vo, B.-T. Vo, M. Jeon, A labeled random finite set online multi-object tracker for video data, Pattern Recognit 90 (2019) 377–389.
[48] S. Chen, C. Shao, Efficient Online Tracking-by-Detection With Kalman Filter, IEEE Access 9 (2021) 147570–147578, doi:10.1109/ACCESS.2021.3124705.

Travis Mandel is an associate professor of computer science at University of Hawai'i at Hilo. He received a Ph.D. in computer science & engineering in 2017 from University of Washington, and a BS in computer science in 2011 from Carnegie Mellon University. His research interests include human-in-the-loop AI and computer vision.

Mark Jimenez is currently a production engineer working in the software industry. He obtained his BS in computer science from the University of Hawai'i at Hilo in 2022. He received a BA in economics and political science from the University of Arizona, Tucson, in 2018.

Emily Risley is currently a software engineer working in industry. She obtained her BS in computer science in 2020 from the University of Hawai'i at Hilo.

Taishi Nammoto is currently a software developer working in industry. He obtained his BS in Physics from the University of Hawai'i at Hilo in 2020.

Rebekka Williams is currently a Master's student studying communicology at the University of Hawai'i at Mānoa. She obtained her MAT in Secondary Mathematics Education from Mount St Mary's University in 2022 and her BA in Mathematics from the University of Hawai'i at Hilo in 2020.

Max Panoff is a Ph.D. student in electrical and computer engineering at the University of Florida, working in the Security in Silicon Laboratory. He obtained his BS in computer science from Stevens Institute of Technology in 2019. His research interests include computer vision and security.

Meynard Ballesteros is currently an undergraduate student in computer science at the University of Hawai'i at Hilo. His research interests include human-in-the-loop AI and computer vision.

Bobbie Suarez is a MS student in the Tropical Conservation Biology and Environmental Science program at University of Hawai'i at Hilo. He received a BS in Marine Biology from Texas A&M University in 2017. His research interests include quantitatively studying coral reef fish populations.