An Automated Sea Urchin Counting Pipeline for

Kelp Forest Restoration Monitoring

Aatish Lobo[1,2], Jianing Li[1,2],

Annie Taylor[2], Ben Grime[2], Kirk Klausmeyer[2], Cody Carroll[1]

[1] *Data Institute, University of San Francisco*

[2] *The Nature Conservancy*

Introduction

The Problem

Kelp forests are among the most productive and structurally important ecosystems in the ocean. Formed by large canopy-forming species such as bull kelp (*Nereocystis luetkeana*), these forests create underwater habitats that "support more than a thousand marine species" including fish, invertebrates, marine mammals, and seabirds, while also delivering tangible benefits to nearby human communities (KRFS). Healthy kelp forests reduce wave energy, help protect coastlines from erosion, and locally buffer ocean acidification. Despite this importance, kelp forests are particularly vulnerable ecosystems, sensitive to changes in water temperature, nutrient availability, and invertebrate grazing pressure.

Worldwide, this vulnerability has translated into significant losses. Estimates suggest that forty to sixty percent of kelp forests have disappeared in the last fifty years, with some regions experiencing near-total collapse (McPherson et al., 2021). These losses are not simply short-term fluctuations: in many places, kelp has failed to recover even after environmental conditions improved, suggesting that something more fundamental has changed in coastal ecosystems.

Along the Northern California coast, this pattern of collapse and non-recovery is especially clear, drawing TNC's recovery efforts and bringing attention to the factors that underlie the loss (McPherson et al., 2021) document a rapid, coast-wide decline in bull kelp beginning around 2014, coinciding with an intense and prolonged marine heatwave in which reduced nutrient availability weakened kelp growth and reproduction. The article makes clear, however, that warming alone does not explain the scale or persistence of the loss; rather, it describes a "large scale shift in the structure of a kelp forest ecosystem," emphasizing that kelp forests transitioned into a fundamentally different ecological state rather than temporarily declining. During the same period, populations of sunflower sea stars (*Pycnopodia helianthoides*), a key predator of purple sea urchins, dramatically fell due to a wasting disease (McPherson et al., 2021). With predation pressure gone, purple sea urchin populations exploded and began grazing kelp at rates far exceeding its ability to recover. With nutrient density low and invertebrate predators abounding, kelp forests transformed into "urchin barrens," which the authors consider a fundamentally different and self-reinforcing ecological state. Even when temperatures cooled and nutrient conditions improved, kelp often failed to return, because dense urchin populations consumed young kelp before it could reach sexual maturity and reproduce. The authors stress that climate conditions alone cannot allow kelp populations to return to what they once were, and that these ecosystems can remain degraded for years or even decades.

The persistence of urchin barrens carries significant ecological and economic consequences. Kelp forests create vertical structure that supports diverse fish assemblages, particularly juvenile fish that rely on kelp for shelter during vulnerable life stages; as kelp disappears, species that depend on it decline, leading to simplified ecosystems with lower biodiversity and reduced productivity. Kelp forests also provide economic benefits to local fisheries and tourism, and when kelp disappears, fisheries often suffer, not only from habitat loss but also from changes in food-chain dynamics. The decline of kelp has contributed to collapses in red sea urchin fisheries (displaced by purple sea urchin) and has left other species, like abalone, increasingly vulnerable. Red abalone in particular have shown lasting sublethal effects from the kelp collapse: surviving individuals exhibited body condition declines and a 99% drop in mature egg production, demonstrating how the loss of a foundation species can suppress population recovery well beyond the initial die-off (Rogers-Bennett et al., 2021).

Despite widespread recognition of the problem, kelp restoration has historically struggled to keep pace with the scale of loss. Restoration efforts are often small, expensive, and labor intensive, and many do not address underlying factors like grazing pressure. Restoration efforts are often small, expensive, and labor-intensive. TNC has noted that most kelp restoration projects are less than one hectare in size and cost hundreds of thousands of dollars per hectare, limiting their ability to combat large-scale declines (Eger et al., 2022).

Over the last decade, numerous studies have demonstrated that targeted removal of sea urchins is the most effective and widely implemented strategy for restoring kelp forests, with success reported in regions such as California and Australia (Williams et al., 2021, Ling et al., 2009). Across these restoration efforts, removal has primarily relied on subtidal hand-harvesting or culling conducted by divers (Ward et al., 2022; Eger et al., 2022; Miller and Shears, 2023;

McHugh et al., 2025). Since 2020, restoration efforts along the North Coast have focused on reducing urchin population density and lessening the impact of urchins overgrazing recovering kelp (Ward et al., 2022; McHugh et al., 2025). However, this region is remote, rugged, and an exposed open-coast environment characterized by frequent wind and wave events that produce rough oceanographic conditions (Ward et al., 2022). This limits the number of days on which safe diving practices can be conducted, emphasizing the need for novel ways of quantifying urchin removal efforts.

TNC's Attempts at Kelp Recovery

As described in the Interim Report, TNC and its partners are testing a new strategy that directly targets the predation pressure identified by McPherson et al. (2021). The project combines large scale purple sea urchin removal, kelp enhancement techniques, long-term monitoring, and community engagement to encourage kelp growth rather than prioritizing replanting efforts.

Early results from TNC's work so far suggest that focusing on grazer suppression can quickly change ecosystem dynamics. In only one year, targeted urchin removal was associated with substantial increases in kelp canopy cover, and young kelp reached reproductive maturity within months.

While the scale of the problem is daunting, TNC's restoration efforts suggest that kelp recovery is possible if interventions are timely and designed with ecosystem dynamics in mind. Furthermore, finding solutions that work in particular locations but can grow in scale could catalyze worldwide kelp restoration efforts.

Our Contributions

What follows is a description of the complete system, from the moment a diver uploads footage to Box through to a final density estimate that restoration crews can act on. Rather than presenting the AWS infrastructure and the detection model as two separate efforts, we describe the pipeline as it actually runs end to end, and then walk through the engineering work that has taken it from an initial working prototype to something the team can depend on for ongoing monitoring.

A diver returns from a transect dive with GoPro footage and, lacking AWS access, uploads it to TNC's shared Box folder. From that point on, no human intervention is required. The upload triggers a transfer into an S3 bucket, which fires two events in parallel: an EventBridge notification that boots up (or confirms the status of) a dedicated EC2 processing instance, and a message placed on an SQS queue recording the new file's S3 path. A lightweight Lambda function handles the EventBridge side, validating that the uploaded object is genuinely an MP4 before starting the instance, and leaving it alone if it iss already running, to avoid paying for idle compute resources between dive days.

Once the instance is live, a worker process polls the SQS queue as a to-do list, pulling messages in batches, downloading the referenced files, and discarding duplicate or transient notifications using retry logic with exponential backoff. This mattered more than expected in practice: Box's intermediate renaming and re-uploading of files during transfer produced "ghost" notifications for files that briefly appeared missing, and without the retry logic these would have been dropped entirely. Once a batch of videos is on disk, the instance runs a combined detection-and-tracking

script, originally two separate scripts merged for simpler operation. The detection half runs a pre-trained Faster R-CNN model frame by frame, producing bounding boxes, confidence scores, and class labels for every candidate urchin. The tracking half takes that raw per-frame output and links detections across frames into consistent identities, using IoU-based matching, the Hungarian algorithm for optimal assignment, and a distance-based fallback that can still link a detection to an existing track even when IoU is zero, provided the two fall within a configurable pixel threshold. That fallback was added specifically to fix a fragmentation problem discovered in lower-frame-rate, differently-formatted footage, where individual urchins were being split into several short, unconfirmed track fragments instead of one continuous track.

From there, each tracked count is combined with the corresponding video's coverage area and survey metadata to produce a density estimate, which is what TNC actually needs for prioritizing removal sites, since a raw count means little without knowing how much reef it came from. The pipeline writes its output to three places: a CSV of per-video counts, density estimates, and metadata; the annotated video itself, with tracked bounding boxes burned in for visual QA; and a running master file across every video processed to date. That master file is maintained as JSON rather than CSV, appended to after each video and only converted to CSV once a batch finishes, since EC2 couldn't append rows to a CSV without rewriting the whole file. This was a small fix, but one that meaningfully sped up batch processing once the number of historical videos grew into the hundreds.

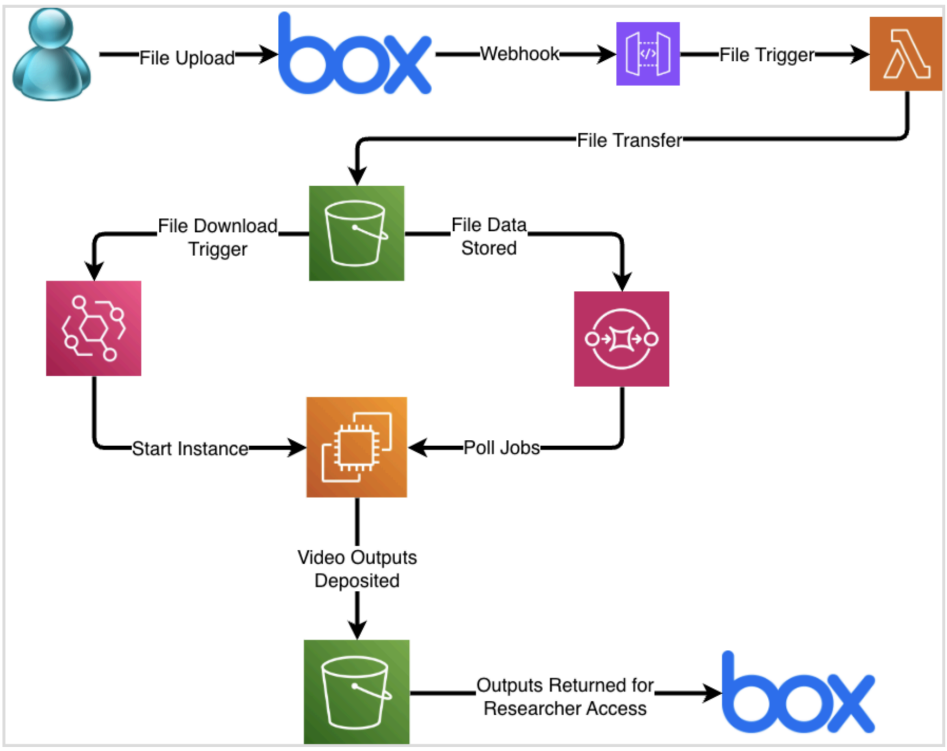

Figure 1. AWS Pipeline Architecture, with stages described above

This section focuses on the algorithmic pipeline for automated urchin counting. We rebuilt the counting workflow as an open Python-based system that separates frame-level detection from multi-object tracking. The detector identifies candidate urchins in each frame, and the pathway script links those detections across frames to produce counts of individual urchins. This design improves transparency and flexibility compared with the original closed-source executable, making the pipeline easier to inspect, tune, validate, and adapt to new transect videos.

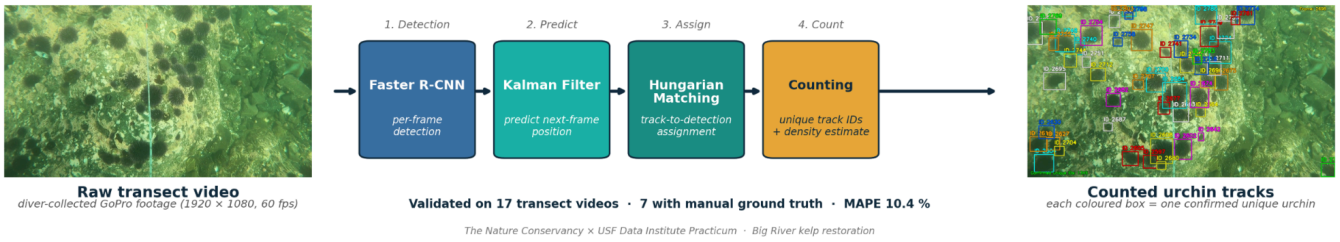

Figure 2. End-to-end pipeline architecture. A raw transect frame (left) is passed through Faster R-CNN detection, Kalman filtering, Hungarian multi-object tracking, and a counting step to produce an annotated transect frame (right) along with per-video urchin counts and density estimates.

Methods

Data, preprocessing, and workflow motivation

The primary data for this project are underwater transect videos collected by The Nature Conservancy. Each video is processed frame by frame so that an object detector can be applied to individual images. For each frame, the detector produces bounding boxes, confidence scores, and class labels for visible urchins. These frame-level detections are stored in JSON format and then passed to the pathway script for filtering, tracking, and counting.

We began by examining the existing Greenwave executable and the outputs it generated on a set of test videos. This inspection showed that the executable produced frame-level urchin detections, but that the detection output alone was not sufficient to generate a final aggregated count. The counting step required a separate tracking procedure to link detections across frames and identify which detections corresponded to the same individual urchin.

We therefore reimplemented the workflow as two Python scripts. The first script runs the

detection model and writes frame-level detection results. The second pathway script processes these detections, tracks bounding boxes across frames, and produces a final count of unique sea urchins. Because detection and aggregation are separated, each stage can be inspected, debugged, tuned, and modified independently for videos with different resolutions, frame rates, and visual conditions.

Detection stage

The detection stage produces frame-level predictions for each video. For every detected urchin, the model outputs a bounding box, a confidence score, and a class label, which are stored in JSON format and associated with the corresponding frame number. This structure is consistent with standard object-detection frameworks. Two-stage detectors such as Faster R-CNN performs inference on individual frames or images; they identify where objects appear, but they do not by themselves determine whether detections across multiple frames belong to the same individual object (Girshick, 2015; Ren et al., 2015; Redmon & Farhadi, 2018). The resulting detection JSON is then passed to the pathway script, where detections are filtered and linked across frames to produce final per-individual urchin counts.

Multi-object tracking: tracking-by-detection

The pathway script follows a tracking-by-detection approach, a common framework in multi-object tracking when frame-level detections are already available. In this approach, the detector identifies where objects appear in each frame, while the tracker links detections across consecutive frames and decides which detections belong to the same individual object. This structure is also the basis of SORT-family trackers, including SORT and DeepSORT (Bewley et

al., 2016; Wojke et al., 2017).

In practice, the tracker maintains a set of active tracks. For each new frame, it predicts the next position of every active track using a motion model, compares those predicted tracks with new detections, builds an assignment cost matrix, and uses the Hungarian algorithm to find the best matching between tracks and detections. Matched tracks are updated with new detections, unmatched tracks are aged, and unmatched detections can start new tracks.

Pre-tracking filtering

Before tracking begins, the pathway script refines the detector outputs. Detections with confidence scores below a chosen score_threshold are removed. The remaining detections are then filtered with non-maximum suppression (NMS), which removes duplicate or highly overlapping bounding boxes based on intersection-over-union (IoU). Confidence thresholding and NMS reduce false positives and duplicate detections before downstream analysis (Redmon et al., 2016; Zhou et al., 2024).

Motion model: constant-velocity Kalman filter

After filtering, detections are linked across frames using track-to-detection matching. For each frame, the tracker compares the predicted bounding box of each active track with each new detection using IoU. These IoU values are converted into a cost matrix, where lower cost indicates a better match.

The Hungarian algorithm is then used to find the assignment between tracks and detections that minimizes total matching cost (Kuhn, 1955). A matched pair is accepted only if its IoU exceeds the configurable match_iou threshold; otherwise, the track and detection remain unmatched. This

combination of Kalman prediction, IoU-based costs, and Hungarian assignment follows the structure of SORT-style tracking (Bewley et al., 2016).

Track confirmation and aging

Raw tracking can produce more tracks than the final number of true urchins because short-lived false positives or unstable detections may briefly appear as separate tracks. To reduce this overcounting, the pathway script applies track confirmation and aging rules.

A track is counted only if it accumulates at least min_hits matched detections across the video. Tracks that remain unmatched for more than max_age consecutive frames are removed from the active set. Together, these parameters control the main trade-offs in the tracker: lower min_hits may admit more false positives, while higher min_hits may exclude real urchins that appear only briefly; higher max_age can bridge short detector misses, while lower max_age reduces the chance of keeping stale tracks alive. Along with score_threshold, these parameters were the main levers used during tuning.

Distance-based association fallback

The IoU-based matching step works well when detections of the same urchin overlap across consecutive frames. However, this assumption can fail when the camera moves quickly, when the video has a lower frame rate, or when the detector produces small shifts in bounding-box position. In these cases, two detections may belong to the same urchin but have little or no bounding-box overlap. Because IoU becomes zero, the tracker may treat them as unrelated objects, causing a single urchin pathway to break into multiple short track fragments.

To reduce this fragmentation, we added a distance-based fallback to the assignment step. The

tracker still prioritizes IoU-based matches when bounding boxes overlap sufficiently. When IoU is too low, however, the fallback allows a match if the predicted track center and the detected box centers are within a configurable distance threshold (max_step). This gives the tracker a second way to preserve continuity when overlap-based matching fails.

This modification was especially useful for out-of-distribution footage, including 30 fps Big River videos, where frame-rate and motion differences made adjacent-frame overlap less reliable. By recovering plausible center-based matches, the fallback improved track continuity and reduced the number of fragmented pathways before final counting.

Aggregate counting and density estimation

After tracking, the pathway output contains bounding boxes associated with consistent track identifiers. Confirmed tracks are then counted as individual sea urchins. These aggregate counts can be combined with video coverage area and severe metadata to estimate urchin density across transects.

This project focuses primarily on producing reliable automated counts. However, the same outputs can support future density estimation. In future work, the counting pipeline could be extended with spatial sampling methods or hierarchical models to account for detection uncertainty and produce more robust ecological estimates (Marques et al., 2013; Williams et al., 2017; Kéry & Royle, 2016).

Results and Interpretations

We ran the pipeline on 17 TNC transect videos in total, covering North Coast 2025 surveys, Big River 2026 pre-removal blocks, and Noyo 2026 post-intervention surveys. These videos

included both in-distribution footage recorded at 1920 × 1080 resolution, 16:9 aspect ratio, 60 fps, and out-of-distribution Big River footage recorded at 5312 × 4648 resolution, 8:7 aspect ratio, 30 fps.

Of the 17 videos, seven had manual reference counts available in a form suitable for quantitative validation. The remaining ten videos were still processed by the pipeline and used for workflow testing or qualitative review, but they did not have comparable manual reference counts, so they could not be included in the calculation of MAE, MAPE, mean bias, or Bland-Altman agreement.

Video	Profile	Ground truth	Pipeline	Δ (%)
06-16-25 Transect 1	In-dist (default)	309	362	**+17.2 %**
06-17-25 Transect 1	In-dist (default)	1,300	1,345	**+3.5 %**
07-10-25 Transect 5	In-dist (default)	334	339	**+1.5 %**
BR Block-5 Heading-0	Big River profile + dist-fb	214	213	**−0.5 %**
BR Block-5 Heading-90	Big River profile + dist-fb	352	391	**+11.1 %**
BR Block-5 Heading-180	Big River profile + dist-fb	735	671	**−8.7 %**
BR Block-6 Heading-180	Big River profile + dist-fb	318	415	**+30.5 %**

Table 1. Per-video validation results across seven ground-truthed transect videos.

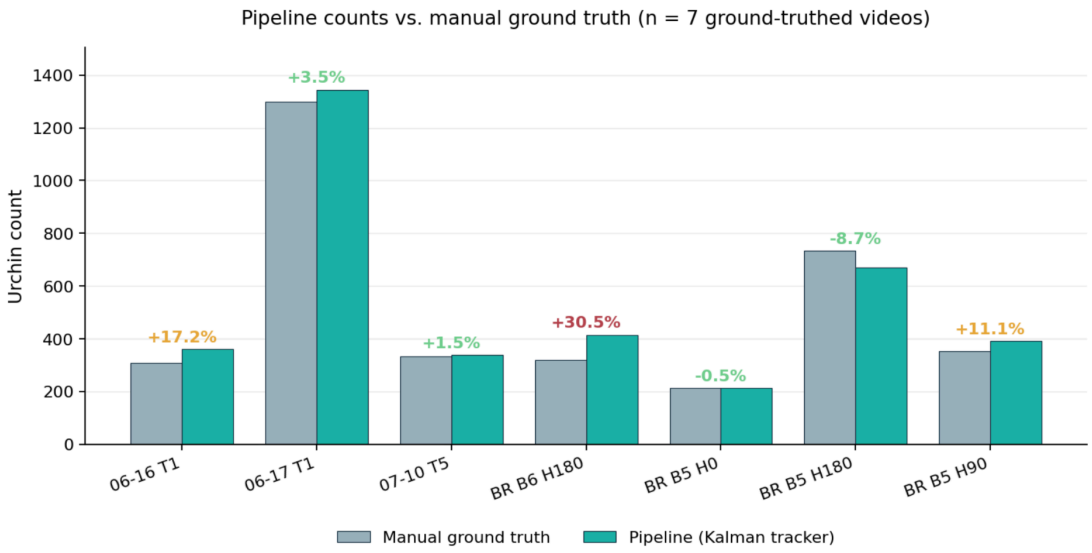

Figure 3. Pipeline counts vs. manual ground truth on the seven videos for which multi-counter ground truth is available. Annotated percentages show the per-video signed error. Best result: BR B5 H0 within 0.5 %; five of seven within 15 %.

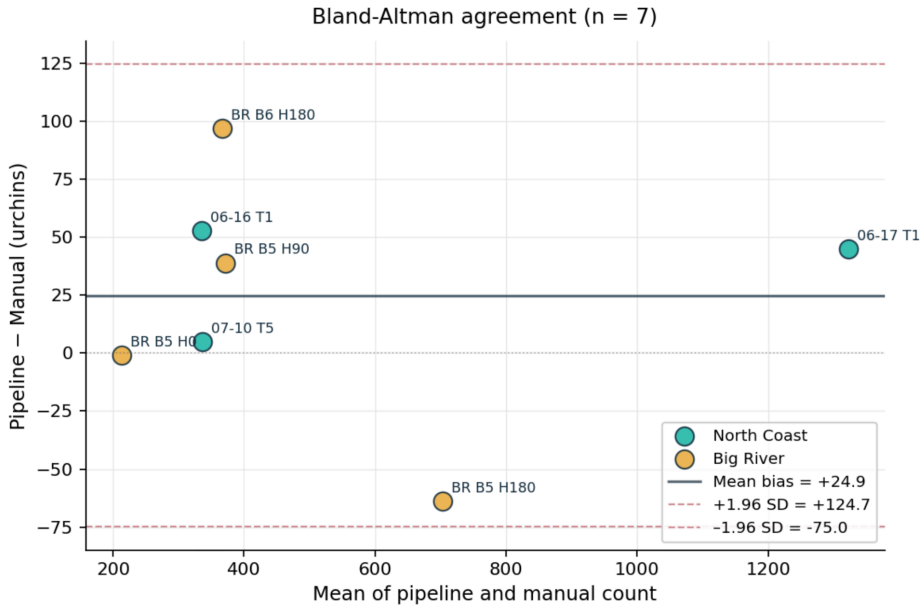

Figure 4. Bland-Altman agreement between pipeline and manual ground truth across the seven ground-truthed videos. Solid line marks the mean bias (+25); dashed lines mark the 1.96 SD limits of agreement. Points are colored by site.

Across the seven validated videos, the pipeline achieved a MAE of 43.4 urchins, a MAPE of 10.4%, and a mean bias of +25 urchins. The Bland-Altman limits of agreement were approximately -75 to +125 urchins. Because the validation set is small, these limits should be interpreted as provisional rather than as a final estimate of long-term performance.

The most accurate result was on Big River Block-5 Heading-0, where the pipeline counted 213 urchins compared with a manual reference count of 214, an error of -0.5%. The largest percentage deviation was on BR Block-6 Heading-180, where the pipeline counted 415 urchins compared with a manual reference count of 318, an overcount of +30.5%. The densest in distribution video, 06-17-25 Transect 1, was much closer: 1,345 pipeline-counted urchins compared with a manual reference count of 1,300, an overcount of +3.5%.

Overall, the pipeline tended to overcount rather than undercount. This positive bias reflects a conservative tuning choice. For TNC's monitoring use case, undercounting is often more problematic than overcounting because an underestimated urchin density could make a restoration site appear less urgent than it is. Therefore, the tracker was tuned to reduce the risk of missing real urchins, even if that meant accepting some false positives.

Comparison with the Greenwave executable

We also compared our pipeline with the Greenwave Robotics executable on the four videos for which all three comparable outputs were available: manual reference counts, our pipeline counts, and Greenwave executable counts. Because Greenwave outputs were not available in a directly comparable form for every video, this should be interpreted as a limited baseline comparison rather than a full 17-video evaluation. Table 2 summarizes this head-to-head comparison.

Video	Manual ground truth	Our pipeline (count)	Our pipeline error	Greenwave EXE (count)	Greenwave EXE error
06-16-25 Transect 1	309	362	+17.2 %	386	+24.9 %
06-17-25 Transect 1	1300	1345	+3.5 %	1034	−20.5 %
07-10-25 Transect 5	334	339	+1.5 %	153	−54.2 %
BR B5 Heading-180	735	671	−8.7 %	1294	+76.1 %

Table 2. Head-to-head comparison of our pipeline against the Greenwave Robotics EXE baseline across the four videos for which manual ground truth, pipeline output, and executable output are all available.

Across these four videos, the Greenwave executable showed larger and less consistent errors than our pipeline. Its error ranged from -54.2% on 07-10-25 Transect 5 to +76.1% on BR B5 Heading-180. In contrast, our pipeline errors ranged from -8.7% to +17.2% across the same four videos. The executable undercounted 07-10-25 Transect 5 substantially, counting 153 urchins compared with a manual reference count of 334, while our pipeline counted 339. It also overcounted BR B5 Heading-180, counting 1,294 urchins compared with a manual reference count of 735, while our tuned pipeline counted 671.

This limited comparison suggests that the open Python pipeline was more consistent and more adaptable than the fixed executable, especially when parameter tuning and tracker improvements were needed. Unlike the executable, our pipeline exposes the main parameters and intermediate outputs, making each run easier to inspect, reproduce, and audit.

Effect of parameter tuning and out-of-distribution diagnosis

The most challenging videos were the Big River videos, which differed from the in-distribution footage in resolution, aspect ratio, and frame rate. The in-distribution videos were recorded at

1920 × 1080, 16:9, and 60 fps, while the Big River videos were recorded at 5312 × 4648, 8:7, and 30 fps. Because of these differences, the default in-distribution tracking profile did not transfer well to the Big River footage.

The main failure was track fragmentation rather than complete detector failure. Annotated review videos showed that the detector was still producing many candidate urchin boxes, but the tracker often split the same urchin into several short tracks. Since tracks must survive long enough to pass the min_hits confirmation rule, many of these short fragments were removed before final counting, producing severe undercounts.

Big River Block-5 Heading-180 illustrates this problem most clearly. Under the default in distribution profile, the pipeline counted only 276 urchins compared with a manual reference count of 735, an error of -62.4%. FPS-scaled tracking parameters alone improved the count to 404, reducing the error to -45.0%. The fully tuned Big River profile, which combined FPS-scaled parameters, a lower score threshold, and the distance-based fallback, increased the final count to 671, reducing the error to -8.7%.

The same Big River-specific profile was then applied across the validated Big River videos. It substantially improved performance on several headings, including BR Block-5 Heading-0 (-0.5%), BR Block-5 Heading-90 (+11.1%), and BR Block-5 Heading-180 (-8.7%), although BR Block-6 Heading-180 still overcounted by +30.5%. This suggests that the tuned profile reduced the major out-of-distribution tracking failure, but did not eliminate all video-specific errors.

This diagnosis also motivated the diver filming protocol included as one of the final deliverables. Because changes in resolution, aspect ratio, frame rate, and camera motion directly affected tracking stability, future videos should be collected as consistently as possible. Standardizing

capture settings and diver movement will reduce unnecessary domain shift and make automated counts more comparable across surveys.

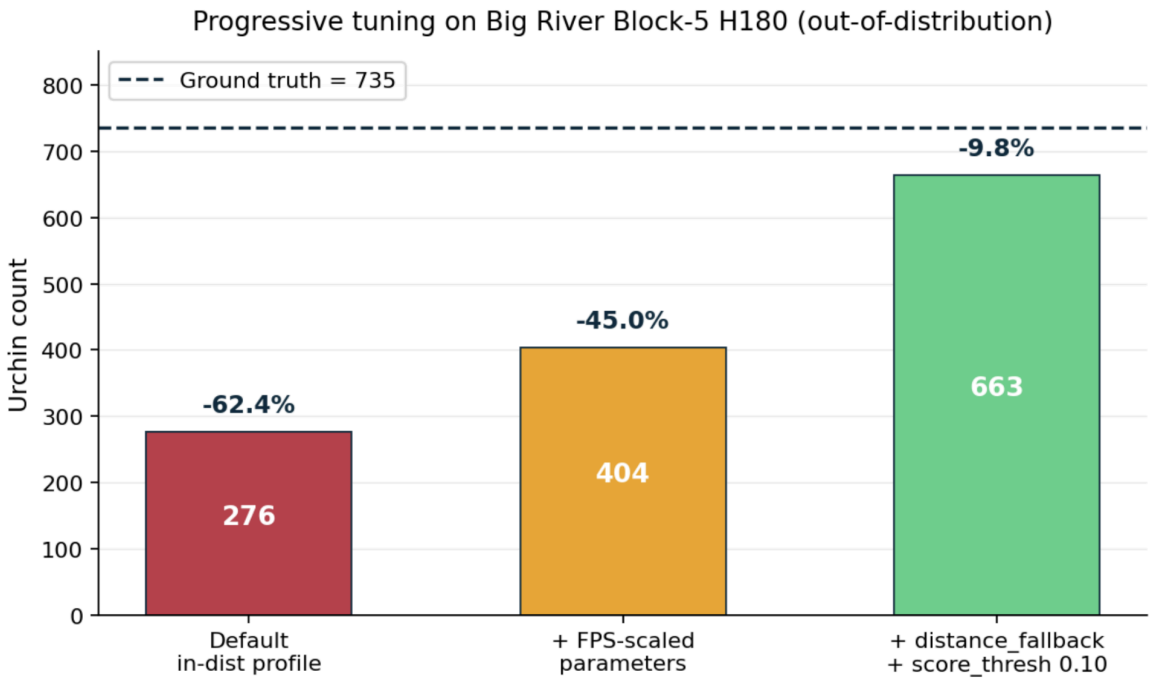

Figure 5. Progressive recovery on Big River Block-5 Heading-180 (out-of-distribution). Default in-distribution parameters undercount the diver estimate of 735 by 62%. FPS-scaled parameter rescaling closes the gap to -45%. Adding the distance fallback and lowering the score threshold to 0.10 closes the remaining gap to -10%.

Discussion

Contribution and importance

Part of the contribution of this practicum is an open, inspectable, and reproducible pipeline for counting sea urchins from underwater transect videos. Compared with the previous vendor supplied executable, our pipeline exposes the major steps of the workflow: detection outputs, pathway generation, tracker parameters, intermediate JSON files, and annotated review videos.

This transparency is important for ecological monitoring because count differences between surveys should reflect real environmental change, not hidden changes in the counting tool.

A second contribution is the distance-based association fallback. Although the change is small, it had a meaningful effect on the out-of-distribution. This suggests that targeted improvements to the tracking stage can recover substantial performance when a detector is already available but the footage differs from the original video format.

Limitations and future work

Several limitations remain. First, only seven videos had manual reference counts suitable for quantitative validation, so the reported MAE, MAPE, mean bias, and Bland-Altman limits of agreement should be treated as preliminary. A larger validation set is needed before making strong statistical claims about performance across sites and video formats.

Second, the detector was treated as a fixed component. We did not retrain it on Big River footage or other out-of-distribution videos, so small, distant, low-contrast urchins remain a likely source of error. Future work should include detector retraining or fine-tuning using annotated frames from Big River and other challenging survey conditions.

Third, the current pipeline produces counts, but density estimation still depends on survey metadata and assumptions about video coverage area. Future work could connect the count outputs more directly to survey-level density estimates, including uncertainty intervals rather than only point estimates.

Conclusion

In this work we developed an automated Python pipeline for counting sea urchins from underwater transect videos. The pipeline combines frame-level detection with Kalman filter-based tracking, Hungarian assignment, track confirmation, and a distance-based fallback for cases where IoU matching fails. Across the seven videos with manual reference counts, the pipeline achieved a MAPE of 14.2%, with the best result within 0.5% of the manual count. On out-of-distribution Big River footage, the distance fallback helped reduce severe undercounting without requiring detector retraining.

The final output is not only a counting model but a reusable workflow: source code, parameter profiles, detection and pathway JSON outputs, annotated review videos, and a diver filming protocol. This provides a foundation that future TNC teams can inspect, tune, and extend for restoration monitoring.

Getting the pipeline to this stable state took more debugging than the architecture above suggests. The most disruptive issue was a bug in how the pipeline retrieved credentials from AWS Secrets Manager: under certain conditions, separate components of the pipeline ended up wiping each other's cached tokens, causing authentication failures partway through a batch run with no single obvious point of failure. Tracking it down required tracing the credential lifecycle across the Lambda function, the EC2 worker, and the detection script independently, since each had its own approach to refreshing and caching tokens, and those approaches turned out to conflict. Since fixing it, the team has treated the authentication chain as high-risk territory: changes there are now made conservatively and reviewed carefully, since a small oversight can silently break the entire pipeline mid-run rather than failing loudly at startup.

Parameter tuning was the other major piece of recent work. Because the detector and tracker interact in non-obvious ways (a stricter confidence threshold reduces noise but risks losing faint urchins, while a looser matching IoU reduces fragmentation but risks merging distinct individuals), the team ran a 256-combination grid search across the main tracking parameters (score threshold, NMS IoU, matching IoU, MAX_AGE, MIN_HITS) to identify settings that performed acceptably across both the original in-distribution footage and the structurally different Big River out-of-distribution videos. Our updated Kalman tracking script, incorporating the distance-based fallback described above, was integrated into the production pipeline following this tuning work. Re-validating the previously reported performance metrics (MAE, MAPE, bias) against these new defaults remains an open item before those figures can be considered final.

Finally, as the pipeline moved from a working prototype toward something the team would depend on operationally, the AWS resources themselves were renamed from ad hoc development names to a consistent production naming scheme, and previously scattered regional resources were consolidated into a single region to simplify monitoring and reduce cross-region data transfer costs.

A handful of standing operational rules came out of this hardening process and are now followed on every deployment:

1. the `*DELETE_LOCAL_ANNOTATED_VIDEO_AFTER_UPLOAD*` flag is always passed on rebuild and push commands, to keep the EC2 instance's local disk from filling up with annotated videos that have already been uploaded to S3;

2. the instance is always stopped, removed, and re-run rather than restarted in place, since docker restart was found to leave stale state behind; and

3. NVML driver errors are resolved by rebooting the instance outright, since in-place fixes rarely worked.

Taken together, these pieces mean a diver's raw footage can go from a Box upload to a reproducible, audited density estimate without anyone on the team manually running a script, reformatting an output file, or babysitting an EC2 instance. That reliability is what makes the pipeline useful for TNC's actual restoration decisions rather than a one-off research demonstration: removal crews and the Oceans Recovery Team can treat its outputs as a steady, ongoing stream of monitoring data instead of something that needs to be re-run by hand every time new footage comes in.

Together, the AWS infrastructure and the tracking algorithm make this project an operational monitoring pipeline rather than a one-off model. The AWS side automates the workflow from Box upload to S3, EC2 processing, output storage, and annotated video generation. The algorithmic side makes the counts traceable and tunable by linking frame-level detections into persistent urchin tracks and allowing parameters to be adjusted when video conditions change.

This combination gives TNC a repeatable way to process diver footage, review automated counts, and produce density estimates for restoration decisions. Future work should expand validation, improve detector performance on out-of-distribution footage, and add uncertainty aware density estimates. Even now, the pipeline provides a practical foundation for faster, more consistent kelp restoration monitoring.

Code and Data Availability

The pipeline source code, parameter profiles, validation data, and the figure-generating scripts used to produce all tables and figures in this report are released as a reproducible artefact. The full Python source including detection.py (Stage 2), pathway_kalman_aws.py (Stage 3), the figure-rendering scripts under /code/paper_figures/, and the AWS deployment configuration are available at https://github.com/tnc-ca-geo/urchin.

References

McPherson, M.L., Finger, D.J.I., Houskeeper, H.F., Bell, T.W., Carr, M.H., Rogers-Bennett, L., & Kudela, R.M. (2021). Large-scale shift in the structure of a kelp forest ecosystem co-occurs with an epizootic and marine heatwave. *Communications Biology*, 4(1), 298. https://doi.org/10.1038/s42003-021-01827-6

Ward, M., McHugh, T. A., Elsmore, K., Esgro, M., Ray, J., Murphy-Cannella, M., Norton, I. and Freiwald, J. Restoration of North Coast Bull Kelp Forests: A Partnership Based Approach. Reef Check Foundation, Marina del Rey, CA, April 2022.

McHugh, T.A., Grime B., Hamilton, S., Downie G., Hughes B., Leuschel, M., Freiwald J., Springborn M., Smith, J. 2025. Accelerating North Coast Kelp Recovery. The Nature Conservancy, Mendocino, California. February 2025.

Miller K.I., Shears, N.T., (2023) The efficiency and effectiveness of different sea urchin removal methods for kelp forest restoration. Restoration Ecology, 31(1): e13754.

Rogers-Bennett, L., Klamt, R., & Catton, C. A. (2021). Survivors of climate driven abalone mass mortality exhibit declines in health and reproduction following kelp forest collapse. *Frontiers in Marine Science*, 8, 725134. https://doi.org/10.3389/fmars.2021.725134

Ling, S.D., Johnson, C.R., Frusher, S.D., & Ridgway, K.R. (2009). Overfishing reduces resilience of kelp beds to climate-driven catastrophic phase shift. *Proceedings of the National Academy of Sciences*, 106(52), 22341–22345.

Williams, J.P., Claisse, J.T., Pondella, D.J. II, Williams, C.M., Robart, M.J., Scholz, Z., Jaco, E.M., Ford, T., Burdick, H., & Witting, D. (2021). Sea urchin mass mortality rapidly restores kelp forest communities. *Marine Ecology Progress Series*, 664, 117–131.

Eger, A.M., Marzinelli, E.M., Christie, H., Fagerli, C.W., Fujita, D., Gonzalez, A.P., Hong, S., Kim, J.H., Lee, L.C., McHugh, T.A., Nishihara, G.N., Tatsumi, M., Steinberg, P.D., & Vergés, A. (2022). Global kelp forest restoration: past lessons, present status, and future directions. *Biological Reviews*, 97(4), 1449–1475.

Miller, K.I., & Shears, N.T. (2023). The efficiency and effectiveness of different sea urchin removal methods for kelp forest restoration. *Restoration Ecology*, 31, e13754.

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *Proceedings of the IEEE International Conference on Image Processing* (ICIP) (pp. 3464-3468). IEEE.

Bland, J. M., & Altman, D. G. (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *The Lancet*, 327(8476), 307-310.

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision* (ICCV) (pp. 1440-1448). IEEE.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1), 35-45.

Kery, M., & Royle, J. A. (2016). Applied hierarchical modeling in ecology: Analysis of distribution, abundance and species richness in R and BUGS, Volume 1. Academic Press.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2), 83-97.

Marques, T. A., Thomas, L., Martin, S. W., Mellinger, D. K., Ward, J. A., Moretti, D. J., Harris, D., & Tyack, P. L. (2013). Estimating animal population density using passive acoustics. *Biological Reviews*, 88(2), 287-309.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779-788). IEEE.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint* arXiv:1804.02767.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems* (NeurIPS) (Vol. 28).

Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. *Proceedings of the IEEE International Conference on Image Processing* (ICIP) (pp. 3645-3649). IEEE.

Zhou, X., Yi, J., Xie, G., Jia, Y., Xu, G., & Sun, M. (2024). Underwater object detection: Architectures and algorithms - A comprehensive review. *Multimedia Tools and Applications*, 83(2), 6259-6300.

Appendix: Dive-Camera Field Protocol

To support future data collection, our dive-camera field protocol is included as Supplementary Appendix A. The protocol gives dive teams a one-page reference for GoPro settings, transect filming technique, filename conventions, and non-standard format handling. It was added because the Big River results showed that inconsistent resolution, aspect ratio, frame rate, and camera motion can introduce domain shift and increase the need for parameter tuning.

Dive Camera Protocol for Automated Urchin Counting

Below is a one-page reference for divers collecting GoPro footage to be processed by the automated urchin counting pipeline. Following this protocol substantially improves count accuracy and reduces the need for video-by-video parameter tuning.

Quick reference

Setting	Value
Resolution	1920 × 1080
Aspect ratio	16 : 9
Frame rate	60 fps
Hover at start of transect	~ 2 seconds, camera steady
Hover at end of transect	~ 2 seconds, camera steady
Movement	Slow, steady forward only and no backtracking

1. Camera settings (GoPro)

Set the camera before each dive:

1. **Resolution: 1920 × 1080**
2. **Aspect ratio: 16 : 9** *(standard widescreen, not 8:7 / SuperView)*
3. **Frame rate: 60 fps**

2. Field technique

Before swimming the transect

Hover at the starting point for about 2 seconds, with the camera steady and the substrate in view. Begin recording, then count "one Mississippi, two Mississippi" before moving forward.

During the transect

1. **Swim forward only, along the transect tape, at a slow, steady pace** (roughly 0.25 m/s, a comfortable underwater walk).
2. **Keep the camera at approximately constant altitude above the substrate.** Please try not to bob up and down.
3. **Keep the transect tape visible in the frame throughout the swim for length verification.**
4. **Do not stop and restart mid-transect, and do not backtrack.** If you reverse and re-cross a region, urchins there will be counted twice.

At the end of the transect

Hover at the endpoint for about 2 seconds before stopping the recording. Continue holding the camera steady, with the substrate in view, for two more counts of "one Mississippi" before pressing stop.

3. Filename convention

After offloading the video, rename the file to:

YYYY-MM-DD_<Site>_<Transect-or-Block>_<length>m.MP4 Examples:

2026-04-15_BigRiver_Block5_10m.MP4

2025-06-16_NorthCoast_Transect1_30m.MP4

The pipeline extracts the transect length (e.g. 10m, 30m) directly from the filename. Misspelled or non-conforming names will fail this auto-extraction and require manual entry.

4. What if you can't shoot in 1080p / 16:9 / 60 fps?

Some scenarios force a different format:

- Camera defaulting to 8:7 (GoPro Hero 12 / 13)
- Low-light conditions requiring 30 fps for higher sensitivity
- Higher-resolution captures (5.3K) for archival purposes

In these cases the pipeline can still process the video, but a video-format-specific parameter profile is required (see METHODS.md §6.b for the Big River profile as an example). Contact the data team and provide:

- The video's resolution, aspect ratio, and frame rate
- A representative sample (~30 s) of footage
- Notes on any unusual conditions (visibility, kelp coverage, etc.)

5. Expected improvement

Surveys collected according to this protocol are expected to:

- Reduce boundary-effect undercount by ~5–10 %,
- Reduce between-survey variability introduced by inconsistent camera handling, and
- Eliminate the need for per-video parameter tuning on in-distribution footage.