Automated Sea Urchin Detection and Tracking for Kelp Forest Restoration Monitoring

UNIVERSITY OF SAN FRANCISCO | MS in Data Science and Artificial Intelligence

Jianing Li[1], Aatish Lobo[1], Annie Taylor[2], Ben Grime[2], Kirk Klausmeyer[2], Cody Carroll[1]

[1]Data Institute, University of San Francisco [2]The Nature Conservancy

The Nature Conservancy | DATA INSTITUTE UNIVERSITY OF SAN FRANCISCO

Background and Problem Overview

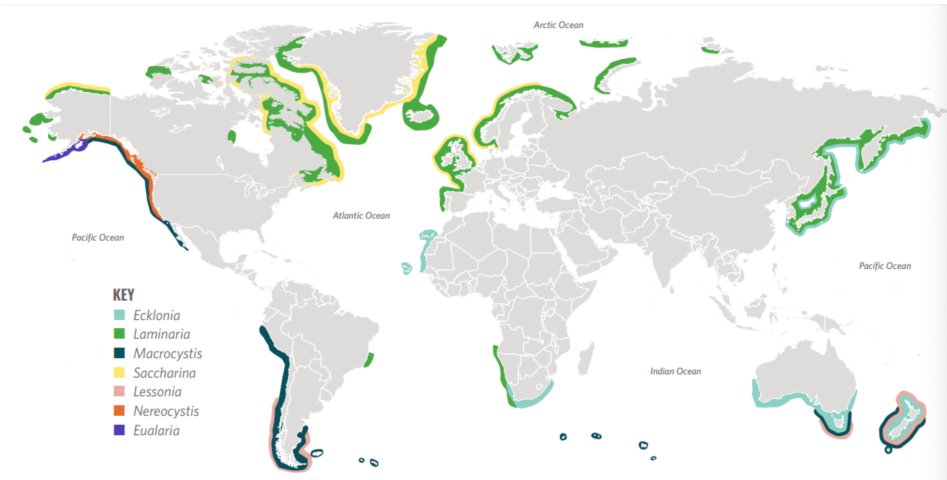

Fig. 1 — *Global distribution of kelp forests. Source: The Nature Conservancy.*

California's North Coast has lost over 96% of its bull kelp canopy since 2015–16, driven in part by dense populations of kelp-grazing purple sea urchins. Restoration teams need reliable estimates of urchin abundance before and after removal, but diver-based monitoring is costly, weather-limited, and difficult to scale.

This project builds a computer-vision pipeline that converts diver-mounted GoPro videos into reproducible urchin counts and density estimates.

North Coast

- 96% kelp loss since 2014–2021
- Multiple stressor events (heatwaves, disease)
- Loss of sunflower sea star
- Persistent urchin barrens with minimal recovery

Fig. 2 — *North Coast kelp forest. Photo: Ralph Pace.*

This project builds an automated pipeline to estimate purple sea urchin abundance from underwater GoPro videos. Using object detection, tracking, and density estimation, the tool helps The Nature Conservancy monitor kelp restoration sites more efficiently than manual diver-based counting alone.

Data Description

We use diver-mounted GoPro videos from TNC kelp restoration transects along California's North Coast. Each video records underwater urchin habitat under varying visibility, lighting, and substrate conditions.

Fig. 3 — *Kelp restoration context. Source: The Nature Conservancy.*

System Architecture

Input: video.MP4

Diver-mounted GoPro video from a 10–30 m transect

Stage 0 — AWS + Box Workflow

S3 video storage → EC2 GPU batch processing → automatic Box upload for team access.

Stage 1 — Detection

Faster R-CNN runs per-frame GPU inference and saves checkpointed detection results every 50 frames.

`detection_result.json`

Stage 2 — Tracking + Counting

Constant-velocity Kalman filter + Hungarian assignment link detections across frames. Tracks with `min_hits` ≥ 50 are counted as confirmed urchins.

`pathway_kalman_summary.json`

Stage 3 — Density Estimation

Density = count / (transect_length × swath_width). Sensitivity range over plausible swath widths.

`estimated_density.json`

Final Output

unique count + estimated density + annotated MP4 uploaded to Box.

Methods

Detection

A PyTorch Faster R-CNN detector, developed by Greenwave Robotics with TNC, runs on every video frame and produces urchin bounding boxes with confidence scores.

Tracking & Counting

A constant-velocity Kalman filter and Hungarian assignment link frame-level detections into tracks. Each track is counted as one unique urchin once it survives at least 50 matched frames.

Density Estimation

The unique urchin count is divided by the surveyed area, defined as transect length × swath width, to estimate density in urchins/m². Density is reported across a sensitivity range of plausible swath widths.

AWS Deployment

Detection runs as batch GPU jobs on AWS EC2, with input videos and output JSON files synced through S3. Final outputs, including summary JSON files and annotated videos, are automatically uploaded to Box for team access.

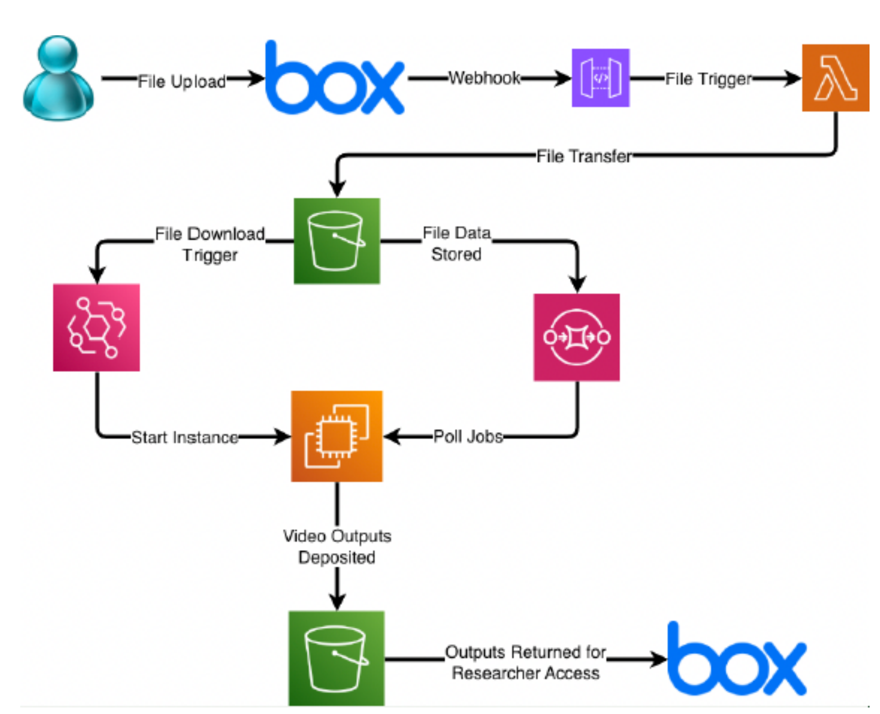

Fig. 4 — *AWS deployment architecture*

Layer 1 — Ingestion

Diver uploads video to Box → Box webhook triggers AWS Lambda → video is streamed into S3 using multipart upload.

Layer 2 — Job Queuing

S3 upload sends a job to SQS and triggers EventBridge to start an EC2 GPU instance.

Layer 3 — GPU Inference

EC2 pulls the latest Docker image from ECR and runs Faster R-CNN detection + Kalman tracking.

Layer 4 — Outputs

Detection JSON, pathway summary, and annotated MP4 are saved to S3, mirrored back to Box, logged in a master CSV, and the EC2 instance shuts down automatically.

Tracking Parameters (OFAT-tuned)

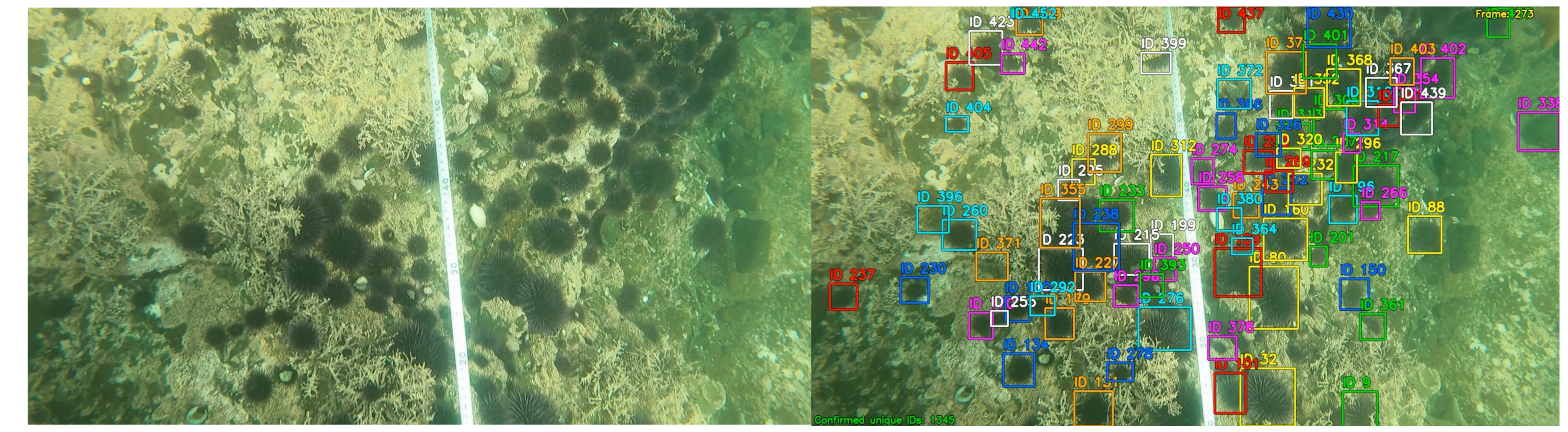

Fig. 5 — *Example original vs annotated frame with confirmed urchin tracks.*

Parameter	Default	Tuned	Effect
score_thresh	0.30	0.20	*Recovers small / low-contrast urchins*
min_hits	50	50	*Filters short-lived false detections*
match_iou	0.30	0.30	*Links boxes across small frame shifts*
max_age	6	6	*Allows brief occlusion recovery*
max_step	80	80	*Prevents unrealistic track jumps*
distance_fallback	off	on	*Recovers matches on 30 fps footage*

Validation Results

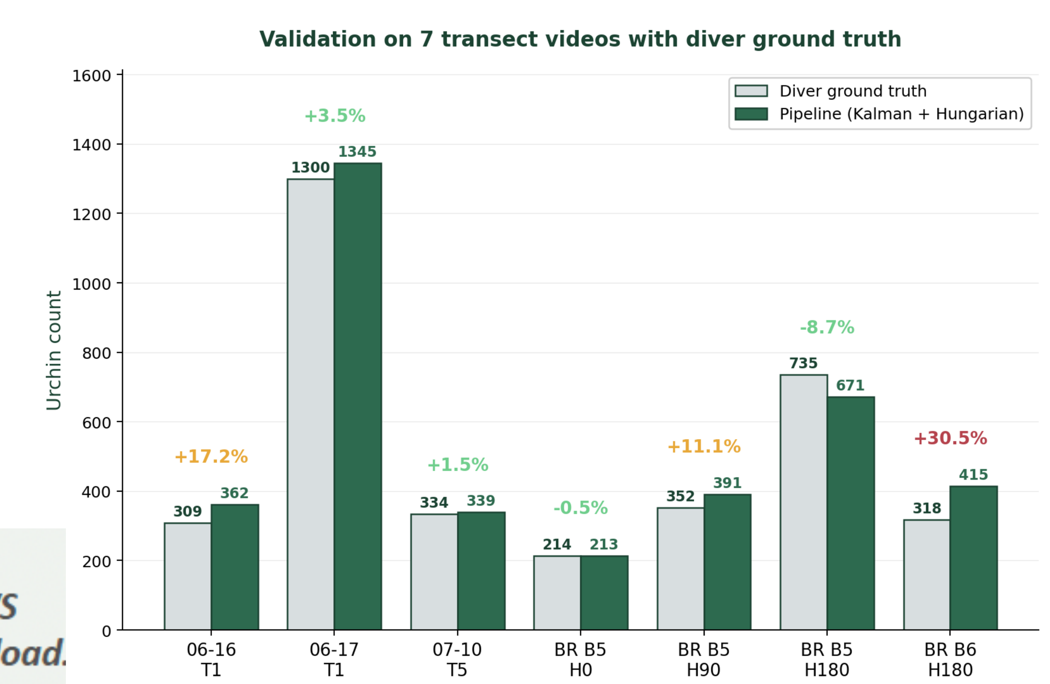

Fig. 6 — *Pipeline vs. diver counts across 7 videos.*

Headline metrics

- MAPE = 10.4 %
- MAE = 43 urchins
- Mean bias = +25 urchins (conservative overcount)
- Best: BR B5 H0 (−0.5 %)
- 5 of 7 videos within ±15 %

Conclusions & Impact

- 10.4 % MAPE on 7 diver-validated transects; matched or outperformed the vendor executable on all 4 head-to-head videos.
- Distance-based fallback closes the out-of-distribution gap on 30 fps footage without retraining the detector.
- AWS deployment lets a diver's raw footage go from Box upload to an audited density estimate without anyone running scripts by hand.
- Enables TNC to monitor more sites at a higher cadence than diver-based counting alone can support.

References

[1] Ren et al. (2015). Faster R-CNN. NeurIPS.

[2] Bewley et al. (2016). Simple online and realtime tracking. ICIP.

[3] Kuhn (1955). The Hungarian method for the assignment problem. Naval Research Logistics Quarterly.

4] Rogers-Bennett & Catton (2019). Marine heatwave and multiple stressors tip bull kelp forest to sea urchin barrens. Sci. Rep.

[5] Eger et al. (2022). Global kelp forest restoration. Biological Reviews.

Code: github.com/tnc-ca-geo/urchin