ChatGPT

LiDAR-Based Penguin Detection Pipeline: Best Practices and Recommendations

1. Methodological Approach – Raster vs. Point-Cloud Detection

Raster-Based (2D Height Map) Approach: Converting the point cloud to raster DEM/CHM layers is popular for small-animal detection because it enables classical image processing (filtering, morphology) on a 2D grid [1]. A fine grid (e.g. 5–10 cm) preserves detail; coarse grids can miss small targets (one study saw error jump from ~2.8% at 0.25 m resolution to >100% at 2 m [1]). Raster CHMs simplify finding local maxima ("peaks") and applying watershed segmentation, which are proven techniques in forestry for separating individual canopies [2]. For penguins, a rasterized Height-Above-Ground (HAG) map is intuitive and allows thresholding by height class. **When appropriate:** Use raster/CHM methods when point density is high enough that a small cell size (< target footprint) can be used without excessive holes. It excels if you require well-defined morphological operations or when many points per animal are available to form a clear "bump" on the HAG surface.

Point-Cloud (3D Segmentation) Approach: Working directly in 3D can avoid interpolation artifacts and use the full precision of the data [3]. For example, a recent UAV LiDAR study segmented cattle by first filtering points above a height threshold then applying 3D clustering (DBSCAN) to group individual animals [4]. This yielded reasonable results (276 of ~400 animals detected, focusing on those standing) [4], confirming the viability of 3D segmentation for individual animals. Point-based detection can be advantageous when point density is low or highly irregular (where rasterizing might drop points/produce voids). It also avoids choosing a raster origin/projection that could slightly shift object positions. **When appropriate:** Use 3D clustering or segmentation when each animal forms a distinct cluster of points separated by gaps. This often suits sparse populations or when the sensor's point spacing is comparable to target size. Modern methods (including learning-based ones) often operate in the point domain for object recognition [3]. **Pitfalls to note:** Point-cloud methods require robust ground removal first – any ground points left can connect clusters. They can be sensitive to clustering distance parameters (needing tuning per density). Very dense colonies pose a problem: points from neighboring animals may merge into one cluster if no clear gap exists. Conversely, raster methods can leverage height "valleys" to separate individuals even if side-by-side, whereas pure distance clustering might merge them. Additionally, 3D approaches may be computationally heavier without specialized spatial indexing or libraries (especially in Python), whereas 2D rasters can leverage fast image ops.

Recommendation: Start with the raster-HAG approach (as you have) for its simplicity and proven track record in ecological surveys. Augment it with 3D point-based checks for edge cases – for instance, a low-density flight or to double-check borderline detections. In dense colonies, a hybrid approach can work: generate a CHM for peak detection, but use the 3D points within each CHM blob to refine exact positions or separate touching individuals. This ensures you get the best of both: the **repeatability and rich toolkit of image processing**, and the **precision of 3D data** when needed.

2. Ground Model Best Practices (Digital Terrain Model generation)

Accurate ground modeling is critical – any bias directly skews HAG values and can hide small penguins. Use **robust ground filtering algorithms** to produce the Digital Terrain Model (DTM). Rather than a simple cell-wise minimum (which can pick up stray low points or burrow artifacts), consider algorithms like **morphological filtering** or the **Cloth Simulation Filter (CSF)** [5]. CSF, for example, drapes a virtual cloth over an inverted point cloud to "feel" the ground surface, needing only a few simple parameters [5]. It effectively distinguishes ground vs. objects and achieved ~4.6% error in benchmarks, on par with state-of-the-art methods [6]. Such methods are robust to outliers and can handle varied terrain. If implementing your own DTM: use a low percentile (e.g. 5th percentile of points in cell) rather than min, to avoid single anomalously low returns (like a part of a penguin or noise) lowering the ground. Then apply a smoothing filter or TIN interpolation to get a continuous terrain. This prevents "bumps" in the ground model under animals.

Slope and Terrain Breaks: Incorporate slope-based filtering thresholds in your ground classifier. Many algorithms allow a *maximum slope* or *height difference* between neighboring points to still be considered ground. Tune this so that actual slopes are preserved, but a sudden 0.3–0.5 m rise (which could be a penguin or rock) is flagged as non-ground. On very steep terrain, you may allow a higher tolerance, but in relatively flat nesting areas, a lower tolerance catches small standing objects. Ensure the algorithm doesn't "bleed" ground upward onto small objects: for example, iterative filters that gradually remove non-ground should not mistake a cluster of penguin points as a small hill. One strategy is to do a *first coarse DTM* with aggressive filtering (guarantee no penguin-sized object remains), then refine it. You can also add a step to **identify and exclude obvious above-ground clusters** (by height and point count) before final ground interpolation.

Vegetation and Rocks: Penguin colonies might have sparse vegetation or rocks which complicate ground modeling. Large rocks can be tricky – if a boulder is present, should it be "ground" or an object? Generally treat **permanent terrain features (rocks)** as ground if they are broad enough, because penguins standing on them should then still appear as +0.5 m above that "ground". However, if rocks are small (comparable to penguin size) and scattered, the DTM might label them as non-ground to avoid confusion. Consider using *ground class plus "feature" class*: e.g., run ground filter in a "smooth" mode to get a bare-earth surface that ignores sharp upward features. Some workflows classify *"low vegetation"* or *"mesh anomalies"* separately. The goal is a **DTM that reflects the true earth surface or long-term static surface** so that any living animal is clearly above it. If the site has grassy mounds or guano mounds, you might need site-specific handling (e.g., treat guano mounds as ground if they are extensive and static between surveys).

Avoiding HAG Bias: Even a ~10 cm error in DTM can hide a crouching penguin. Ensure ground algorithms are not biased high in areas where penguins cluster. A common failure would be if many returns on a penguin's back are mistaken for ground – the local ground height will be too high, yielding near-zero HAG for that penguin. Prevent this by **excluding points in the penguin height range from DTM extraction** (if you have an idea of target height). For example, one could ignore all returns 20–80 cm above the local minimum when computing the DTM. Also, enforce a **minimum terrain window**: the DTM under a penguin should interpolate from surrounding ground points, not use the penguin itself. In summary, use a modern ground filter (progressive TIN densification, CSF, or similar), tune it with field knowledge (allow minor terrain roughness but filter out penguin-sized bumps), and verify the resulting HAG visually. The ground model step should be consistent across sensors – test on known flat open areas (no penguins) to ensure each

sensor's data produces a flat DTM with minimal noise. This will make your HAG stable and unbiased for the small targets.

3. Parameterization Guidance for Detection

Choosing the right parameters is key for repeatability and cross-site performance:

- **Cell Size (Grid Resolution):** Use the highest resolution that the data supports (without excessive holes). A cell size ~ half the footprint of a penguin or smaller is ideal. For example, if point density allows ~5 cm cells, that would capture a penguin (~0.3 m wide) across multiple cells. Typically 5–10 cm is a good starting point for UAV LiDAR with high density; if using a lower-density sensor/altitude, you might go to 10–15 cm. Research has shown that overly large cells drastically reduce detection accuracy for small objects [1]. **Adaptive resolution** can be considered: if a tile has sparse points, you could temporarily use a larger cell or use a *point density threshold* to flag regions where counts might be low. However, keeping a consistent cell size across the project is better for uniformity – instead, handle sparsity by interpolation or by marking those areas for manual review. In summary, pick the smallest cell size that yields a mostly filled grid (with at most a few gaps that can be closed by morphological filtering).

- **HAG Thresholds:** Your current 0.2–0.6 m band for "penguin-sized" objects is a reasonable starting range. Penguins (~0.6–0.7 m tall for adults) should protrude above ground at least ~0.5 m, while low artifacts (rocks, driftwood) might be under 0.2 m. It's wise to apply a *lower bound* (e.g. 0.2 m) to eliminate flat ground noise and a *upper bound* to exclude tall objects (if any). For instance, anything above ~0.8 m could be a person or a taller object rather than a penguin. You might refine these per site: if juveniles or chicks (~0.2–0.3 m) need counting, you may lower the threshold. A best practice is to examine HAG histograms from a few sample tiles – often you'll see a cluster of returns in the 0.4–0.7 m range corresponding to penguins. Use such data-driven histograms to fine-tune the threshold for each sensor/site if needed (but keep it within biologically reasonable bounds, to avoid overfitting). The threshold should also account for ground model noise: if DTM has ±10 cm error, you might buffer the lower bound slightly upward to avoid false negatives (missing a 0.2 m penguin because ground was low by 0.1 m). If using intensity or other features later (e.g. thermal in fusion), you might keep more candidates (maybe down to 0.1 m) and let the fusion step discriminate – but for now, a focused band reduces false positives.

- **Minimum/Maximum Object Size:** After thresholding the HAG raster, use connected-component labeling to identify blobs of connected "penguin-height" pixels. Set a **minimum size filter** to remove tiny noise blobs – e.g., an area of a few pixels (equivalent to perhaps <0.02 m^2) might be just a single stray point or a sharp DTM error. A single penguin seen in LiDAR from above might create a blob on the order of, say, 0.1–0.3 m^2 (depending on how many points hit it and the cell size). If your cell is 5 cm, a standing penguin might cover ~20–30 pixels at most. So a reasonable minimum might be ~3–5 connected pixels (to allow partial captures). The **maximum size** is useful to flag clusters that likely contain multiple penguins or non-targets. For example, a blob spanning an area >0.5 m^2 or an absurd number of pixels is probably either multiple penguins merged or a large object (e.g., a seal, a rock outcrop, or a human). Rather than throw those out, you might mark them for *splitting* (see watershed below) or at least for manual inspection. Shape metrics can also be computed: a very elongated shape or large footprint could be a penguin lying down stretched out (unlikely – penguins are upright) or more likely a smear of points (e.g., from motion or a cluster of penguins). Simpler is

to use area and maybe height range: if the blob's internal max HAG is much higher than a penguin (e.g. 1 m), it's not a single penguin. Conversely, if a blob's area is huge but height never exceeds 0.5 m, it might be a low, broad object (not a penguin). Use these criteria to either filter obvious non-penguins or trigger an **instance separation** routine.

- **Local Maxima vs. Connected Components:** There are two ways to pinpoint individual detections: (a) **Connected components** (each contiguous blob = one detection, unless later split), or (b) **Local peak detection** (find local HAG maxima as candidate centers). The connected-component approach is straightforward and finds all above-threshold regions. However, in dense groups one "blob" can contain multiple penguins touching. The local-maxima approach treats each peak (high point) as an individual and can work even if bodies touch, as long as each penguin has a distinguishable top. In forestry, local-max + watershed is a standard to separate tree crowns [2]. For penguins, the difference is scale – peaks might be only a few tens of cm apart in a cluster. We recommend a **hybrid:** identify all connected blobs, then for each blob that is larger than a single-penguin footprint, find internal local maxima. If a blob has two or more distinct high points, likely it's two penguins very close. You can implement a 2D peak finder on the HAG raster (with a window ~0.2–0.3 m) or even simply count how many "cores" a blob has. Research in 3D segmentation of vegetation suggests using local maxima to split clusters improves identification of individual components [7]. **Summary:** Use connected components to get candidate areas (ensuring you catch everything), but use local peak logic to decide if a single area should be counted as multiple. This will increase correctness in dense colonies. Keep in mind the trade-off: local maxima might over-separate if the HAG surface is noisy (two peaks on one penguin's back due to noise). Mitigate that by smoothing the HAG slightly (small Gaussian blur ~1–2 pixels) before peak finding, and perhaps ignore very minor secondary peaks (within say 5 cm of the primary).

- **Shape and Elevation Metrics:** It can help to compute features for each candidate blob for QC or future classification. E.g., the blob's height above ground (mean and max), area, elongation (eccentricity), and maybe LiDAR intensity statistics. Penguin detections might have a roughly circular shape in the raster and a particular height profile (e.g., a single peak ~0.5 m). Rocks might be more irregular in shape; clusters of penguins might be larger and multi-peaked. Tracking these metrics per detection can assist rule-based filtering (for instance, if something is 0.25 m tall and spans 1 m^2, it's likely a rock ledge with guano rather than individual penguins). These metrics also become useful diagnostics if later you incorporate a machine-learning classifier or want to explain why something was or was not detected.

4. Watershed and Instance Separation Techniques

When multiple penguins huddle together, simple connected-component labeling can under-count (treating a group as one large blob). **Watershed segmentation** (often marker-controlled) is a best-practice to separate touching objects in imagery. The idea is to treat the HAG surface like a terrain and "flood" it so that ridges between peaks form boundaries [8]. In forestry, marker-controlled watershed (using treetops as markers) successfully delineates individual tree crowns without over-segmenting [2] [8]. We recommend a similar approach: use the local peaks (as "markers" for penguin locations) and perform a watershed segmentation on the inverted HAG raster (so water flows into low areas between penguins). Ensure you impose minima at the marker points (to prevent spurious extra basins). This will split broad clusters at the lowest saddle points between peaks.

Tuning Watershed: Key parameters include the degree of smoothing and the threshold for what constitutes a separate individual. If you over-smooth the HAG, you risk under-segmentation (merging two penguins into one mound). If you under-smooth (too noisy), watershed can over-segment (creating divisions within one penguin). A good practice is to apply a light Gaussian blur to the HAG before peak detection and watershed – this removes high-frequency noise but preserves the broader shape. Next, enforce a minimum valley depth between peaks: for example, only split a blob if the valley between two peaks drops by >10–20 cm relative to the shorter peak. This prevents watershed from splitting for tiny height differences. In a dense colony, penguins might actually be in physical contact, so the "valley" could be very shallow (perhaps a few cm dip). You may need to accept some under-segmentation in the most extreme cases (those can be noted for manual count adjustment if needed).

When Watershed Helps vs. Not: Watershed (or any instance separation) is most helpful when individuals have distinct peaks. If penguins cluster tightly in a circle (e.g., protecting chicks), from above their LiDAR returns might form one blob with no clear internal peak difference – in such truly fused cases, even watershed may not partition them correctly (under-segmentation). Conversely, if one penguin stands slightly taller or they're back-to-back, watershed will likely separate them (each has its own peak). **Failure Modes:** Over-segmentation can occur if the HAG has multiple peaks on one animal – e.g., a flipper or head separate from body creating two peaks. This could cause a false split. Tuning the peak detection to require a certain minima distance or area can alleviate this. Another failure mode is under-segmentation in **dense rows** (penguins lining a slope tightly). In such cases, consider a higher-resolution analysis or even a manual assist. **Watershed markers:** We strongly suggest **marker-controlled watershed** – i.e., identify likely penguin tops first (markers) and use those to guide the segmentation [8]. This prevents the classic over-splitting where plain watershed finds every local bump (even noise) as a separate basin. By restricting basin centers to your validated peaks, you get a one-to-one correspondence: each peak yields one segmented region.

Instance Counting in Dense Colonies: Even with watershed, extremely dense colonies might yield slightly off counts. It's important to flag these situations (perhaps based on average spacing or an unusually low per-penguin area in that region) for special handling. If thermal imagery will later be fused, it might help separate heat signatures even if LiDAR blobs are merged. In this LiDAR-only stage, do what's *deterministic and documentable*: e.g., "if one blob contains multiple peaks above 0.4 m, count each peak as an individual." That rule, once tuned, should remain fixed for all sites to ensure consistency. Document such rules clearly in metadata (see QA/QC section) so that if there's an obvious under- or over-count in a dense cluster, reviewers know what the algorithm decided (and can override if needed).

5. Determinism and Reproducibility

For a pipeline that runs across many flights and sites (and will be extended with sensor fusion), determinism is crucial. **Ensure the pipeline yields identical results given the same input, every time.** Here are best practices to achieve that:

- **Avoid Non-Deterministic Operations:** If using any functions that leverage randomness (e.g., a random sample consensus) or parallelization, ensure you fix random seeds and/or enforce a deterministic order. In Python/Numpy, most operations are deterministic by default, but beware of parallel processing or multi-threaded BLAS libraries that could sum in different orders (causing tiny float differences). If you chunk the data (say, process each tile separately or in parallel), be mindful that floating-point summations or morphological operations at tile edges could differ compared to

processing a merged tile. It's often best to process each tile with a consistent padding/overlap and a fixed order to avoid race conditions.

- **Stable Sorting and Indexing:** When outputting detection centroids, always sort them (for example, by easting/northing or by tile and ID) so that output order doesn't depend on quirks of memory or hash ordering. This ensures that if nothing changes in the input, the "Nth detection" remains the same across runs. This can also support generating stable **IDs** for detections (e.g., combine tile ID and sorted index) – which is useful if later you want to track the same detection through different pipeline versions or compare outputs.

- **Guardrail Unit Tests:** Establish some "guardrail" tests using known data. For example, take a small LiDAR subset with a few dummy targets (even a synthetic scenario or a single flight from last year) and run your pipeline end-to-end. Save the outputs (centroids, count, even the intermediate raster if possible). On each code update or different environment, run it again and diff the results. The results should be *exactly* the same. If differences occur (aside from intentional improvements), that flags an issue. This approach will catch things like if a library update changed how a function behaves (e.g., a morphology function rounding differently). Because floating-point differences can accumulate, consider adding tolerances for intermediate results if needed (e.g., two centroids within 1e-9 might be considered the same for test purposes).

- **Floating-Point Precision:** Use double precision (64-bit) for critical calculations like computing coordinates, DEM interpolation, and HAG values. This reduces numeric differences across platforms. If you use any GPU acceleration or lower-precision libraries, be extra cautious – stick to CPU double precision in this pipeline, as speed is not the primary concern, but consistency is. Also, be careful with any summing or averaging (though in counting penguins, you might not have much of that) – if you do something like compute an average height for each blob, the order of accumulation could matter. Use numpy's stable functions or sort inputs before summation for reproducibility.

- **Environment & Version Pinning:** Document and, if possible, lock the versions of core libraries (laspy, numpy, scipy, scikit-image, etc.). Floating point algorithm changes between versions can yield slight differences. You might use Docker or a conda environment file to ensure the field deployment uses the exact versions you tested. If upgrading any library, re-run the guardrail tests to ensure no change. Similarly, differences in hardware (e.g., different CPU vector instruction sets) are usually minor, but if running on heterogeneous hardware, test on each.

- **Chunking and Edge Effects:** If processing in spatial tiles or chunks, always include a buffer zone and then clip to tile boundary in final outputs. This ensures consistency whether you process one big area or each tile separately – the results in the overlap region will be identical. For example, if using a 1 m buffer around each tile for the DEM and blob detection, two adjacent tiles will produce overlapping detections in the buffer area. You must have a deterministic rule to keep or discard those (e.g., maybe always discard the ones in the right or bottom buffer to let the left/top tile own them, or merge and unique them post-hoc). Define this rule clearly so that running the pipeline as a whole vs. tile-by-tile yields the same final set. This is crucial for reproducibility because field teams might re-process a single tile for QA – they should get the same detections as in the full batch run.

- **Output Signature:** Consider creating a simple "fingerprint" of each run's output for verification. For instance, compile a list of all detection coordinates (sorted) and hash them (MD5 or similar). Store

this hash with the output. This way, if someone re-runs the pipeline on the same input, they can quickly see if anything differed (if the hash changes, something is off). This can be part of your QC log. Even printing a summary like "Tile X: 15 detections (hash=abc123)" can aid quick comparisons. Over time, these signatures ensure no unintended drift.

Finally, **document all parameters and algorithm steps in the output metadata** (see section 6) – this makes the pipeline's behavior transparent and thus more defensible. If someone questions a count, you can show it was produced with a fixed set of rules applied uniformly.

6. QA/QC Artifacts for Field Validation

Providing rich QA/QC outputs will build trust with field teams and scientists reviewing the results. We recommend the pipeline produce the following artifacts and visualizations for each batch (or tile):

- **HAG Maps and Overlays:** Generate a colorized Height-Above-Ground raster (DEM of just the targets). For instance, create a GeoTIFF or PNG where each pixel's intensity reflects height above ground (capped at say 1 m for contrast). Overlay detection markers (e.g., red dots or circles for each detected centroid). This gives an intuitive map for field QC – reviewers can see if detections align with visible "mounds" on the HAG. It can highlight if some obvious mounds were missed (suggesting a threshold issue) or if noise was picked up (false positive blob with no real height bump). Ideally, also overlay ground truth points or field-count locations if available for comparison.

- **Candidate Blob Outlines:** For deeper QC, output an image (or a vector layer) showing the outline of each detected blob on top of, say, an intensity image or HAG image. This could be a PNG where each detection's region is contoured. Field staff can then see the approximate "footprint" the algorithm identified for each penguin. This helps in diagnosing splits/merges (e.g., if two penguins are in one outline, you know watershed missed a split).

- **Histograms and Distribution Plots:** Provide histograms of point heights, HAG values, and blob areas for each tile or flight. For example, a histogram of all point heights after ground removal might show a distinct peak corresponding to penguin heights. If the threshold is correctly set, that peak should be well-captured. If there's a secondary peak in HAG at, say, 0.1 m (perhaps rocks or noise) that got filtered out, it's good to show that as validation that your filter is excluding those. Additionally, histogram of blob sizes can show if many very small blobs were detected (could indicate false positives due to noise or partial returns). These distributions can be compiled per site to track consistency.

- **Spatial Density or Heatmaps:** Create a low-res heatmap showing detection density across the area. This could simply be a grid count of detections per 5 m cell, visualized as a shaded map. It helps identify colony clusters and any unexpected detections far from colony centers (which might be false positives on random objects). If you have known colony regions, you expect high density there; a random high density spot elsewhere could indicate an issue (or an undocumented sub-colony!).

- **QA Plots per Tile:** For each LiDAR tile, consider outputting a small multi-panel QA image (e.g., a PDF or PNG). For example: Panel 1 – DSM (first return surface) grayscale; Panel 2 – DTM (ground) model; Panel 3 – HAG raster (color); Panel 4 – HAG thresholded blobs (binary image) with outlines; Panel 5 –

final detections overlaid on an ortho photo or intensity image if available. This might seem a lot, but automation can produce these, and they are extremely informative for quick scanning of results. Field teams can flip through these "report cards" for each tile to spot anomalies.

- **Embedded Metadata:** Every output file (GeoJSON, GeoTIFF, etc.) should contain metadata fields describing how it was produced. For example, in a GeoJSON properties or an accompanying JSON/YML: algorithm version, date, thresholds used, cell size, ground method, sensor name, coordinate system, etc. This is critical for defensibility – months later, you want to know exactly which parameters were used for a given result. If using GeoPackage or shapefile, use the metadata or description fields. If nothing else, include a README with the output data that lists all runtime parameters and any site-specific settings. Also record the LAS file source and timestamp. Essentially, someone else should be able to take the same LAS input and these parameters and reproduce the detections exactly.

- **CRS, Units, and Sensor Info:** In QA maps and outputs, clearly label coordinate reference system and units. For instance, a map should note "Projection: EPSG:xxxx (WGS84 UTM 20S)" or similar. Heights should be labeled (e.g., "HAG in meters"). Including the sensor ID or profile used for that site in the metadata is useful (since different sensors might have different point density or precision, which could be relevant if questions arise). If the pipeline uses a different ground algorithm per sensor, note that too. Basically, any assumption or profile that was applied should be recorded. This metadata might also travel with the detections into the fusion stage so that stage knows the context (e.g., which sensor's thermal data to pair, etc.).

- **Runtime and Quality Metrics:** It can be helpful to output some basic quality metrics: e.g., point count processed, ground point count, non-ground count, number of blobs found, number of final detections. These can be logged per tile and aggregated. They serve as sanity checks – e.g., if one tile had zero detections but neighboring ones had many, that stands out (maybe an input or processing issue). Or if a tile had an abnormally high raw point count or very few ground points, that could indicate a sensor data anomaly (e.g., water or reflecting surfaces). Logging these ensures any oddities in the LiDAR data or processing are caught early.

In summary, produce **visual maps (HAG, detections)** for intuitive review, **statistical summaries** (counts, histograms) for aggregate monitoring, and embed **detailed metadata** so every output is traceable and reproducible. This combination will make field QA much easier and provide evidence to defend your methodology.

7. Geospatial Output Conventions and CRS Handling

Getting coordinate systems right is vital – misplacing data by meters or degrees can lead to "wrong planet" scenarios that are surprisingly common in GIS pipelines. Follow these conventions:

- **Preserve and Declare CRS:** LiDAR tiles likely come in a projected CRS (e.g., UTM or local grid). Ensure you read that CRS from the LAS header (if available) or otherwise know it. All outputs should either stay in that CRS with proper metadata, or be transformed to a known CRS, **but never left ambiguous**. A common standard is to use a projected CRS (like UTM) for all intermediate processing (for metric units), and then transform final outputs to WGS84 geographic if needed for certain deliverables. GeoJSON, for instance, by IETF standard *must be in WGS84 (EPSG:4326) lat/long* [9]. If

you output GeoJSON, convert the coordinates from the projected CRS to lat/long degrees to comply (and note that precision may suffer if you have high-precision local coordinates). Alternatively, use formats that support projections (GeoPackage, Shapefile, etc.) so you can include the original CRS. Given the need for accuracy and avoiding any datum issues for subsequent fusion, it might be safer to keep everything in the projected CRS of the LiDAR and use GeoPackage or Shapefile (with a .prj) for outputs. Only produce WGS84 GeoJSON if there's a specific need (like web mapping). If you do, double-check a few points in GIS to ensure they land exactly where expected.

- **CRS Consistency:** Different sensors might output data in different coordinate systems (maybe one in WGS84 UTM, another in a local datum). Normalize this early in the pipeline. Decide on a project CRS (UTM zone that covers the sites, or one per site if needed). Transform input point clouds if necessary, or at least handle each tile's CRS explicitly. This prevents mixing coordinates. Mixed CRS issues can cause the dreaded "data appears in wrong location" problem. A guardrail test: compute the approximate latitude/longitude of a known point (say, the colony center) in each dataset and verify it matches expectation (e.g., known site coordinates). If a tile comes with no CRS info (occasionally LAS files might not have it embedded), treat that as an error – don't assume, but rather verify externally and then assign the correct CRS before processing.

- **ASPRS LAS Standards:** If you output back to LAS/LAZ for any reason (say, classify penguin points), use proper LAS 1.4 classification codes. While there is no standard code for "animal," you could use a user-defined class if needed. More relevant, ensure the CRS is stored in the LAS header's VLR as WKT (LAS 1.4 prefers WKT for CRS definition) [10]. Many tools (laspy, PDAL) do this automatically, but double-check. This avoids any consumer of the LAS misinterpreting coordinates. Given your pipeline outputs are centroids (likely as CSV/GeoJSON/GeoPackage), the LAS spec mainly applies upstream. However, referencing the **ASPRS LAS 1.4 specification** is useful for defensibility: e.g., state that you adhere to it for coordinate systems and that intensity values are left as-is (0–255 or 16-bit) unless otherwise normalized per standards.

- **Projected vs. Geographic Outputs:** Use **projected coordinates for local analysis and metrics**, use **geographic for final dissemination or multi-layer integration**. For instance, the fusion with thermal might be easier if both LiDAR and thermal are in the same projection (likely UTM or a local planar system) so that distance units match (meters). Output your detection centroids in that projection for the fusion stage. But if you need to provide results to ecologists or in a report, they might prefer lat/long. In that case, provide a separate file (or a GeoPackage layer) in WGS84 or include a column for lat/long in a CSV. Always label them clearly. And remember that mixing up lat/long as x/y in GIS can shift data dramatically. To avoid the "wrong planet" or wrong location: perform a quick QC by loading the outputs on a basemap. This manual step (or automated if you have a test script) will catch a mistaken coordinate system. For example, if you output something as if it were EPSG:4326 but actually in UTM, you might see points all bunched or offset by kilometers on a world map – a clear red flag.

- **GeoJSON vs GeoPackage vs CSV: GeoJSON** is convenient for web and lightweight use, but as noted it mandates WGS84 and doesn't preserve metadata like CRS or even units beyond coordinates. **GeoPackage** (an SQLite-based GIS format) is a great choice for an "engineering-friendly" output: it can store points in a table with their attributes, include the CRS, and is a single file easy to share. It also avoids problems of text formatting and precision that GeoJSON/CSV might have. If your team is comfortable, use GeoPackage for final delivery to GIS/QC (it can be directly opened in QGIS, ArcGIS,

etc., with no ambiguity). **CSV** (with maybe X, Y in projection or lat/long) is okay for quick lists, but it lacks CRS info – only use if accompanied by documentation of the coordinate system. We recommend providing at least one self-describing geospatial format (GeoJSON with WGS84 clearly indicated, or a shapefile/GeoPackage with projection). This prevents downstream users from guessing the projection incorrectly (the classic "wrong planet" error happens when, say, a CSV of UTM coordinates is assumed to be lat/long by someone).

- **Preventing CRS Mistakes:** Implement a safety check in code: if you know the expected numeric range of coordinates (e.g., for UTM in Argentina, X ~ 300k to 400k, Y ~ 4 million), you can assert that output coordinates fall in that range. If you accidentally output in degrees when you think it's meters, the values would be ~ -65, -42, etc., which is clearly out of expected range for UTM. Catch that and warn or correct. Also, if outputting lat/long, consider including altitude or even a height attribute (e.g., average HAG or elevation) to maintain some 3D info; but if not needed, at least document that heights are not in those outputs to avoid confusion.

In summary, choose output coordinate systems deliberately and document them. When in doubt, **include the EPSG code** in file names or metadata. For example, an output could be `PenguinDetections_SanLorenzo_2025_UTM20S.geojson` – this hints it's in UTM zone 20S. Clarity here will save headaches and ensure your results overlay correctly with thermal imagery and maps.

8. Validation Strategy (Precision/Recall & Performance Tracking)

A rigorous validation plan will quantify how well the LiDAR pipeline is counting penguins and guide improvements:

- **Sampling for Manual Verification:** It's often infeasible to ground-truth every detection, especially across large colonies. Instead, use a stratified sampling approach. For each site or survey, randomly select a subset of detection locations to verify with independent data. This could be done via high-resolution RGB photos, on-site counts, or the upcoming thermal imagery. Ensure the sample covers various scenarios: dense colony sections, sparse edge areas, areas with rocks, etc. By stratifying (e.g., 10 samples in high-density, 10 in low-density), you can estimate performance in each regime. For each sampled location, determine the "truth" (was there actually a penguin there? Did we miss any nearby?). This will allow estimation of **precision** (positive predictive value) – what fraction of detections are real penguins – and **recall** – what fraction of real penguins were detected.

- **Handling Imperfect Ground Truth:** Field counts are often aggregate (e.g., "we counted 500 penguins in the colony") and not pinpointed. You can still use them for coarse validation: if your pipeline consistently finds, say, 550 in that colony, that's a potential overcount (maybe false positives or double counts). If it finds 400, an undercount (misses). However, aggregate counts lack spatial info. To supplement, use any partial ground truth you have: e.g., if field teams took GPS points or marked locations on images for some penguins or groups, use those as validation points. One practical approach: overlay your detections on any available high-res orthophotos or ground video if available and have an expert visually check a few tiles thoroughly, counting penguins manually and comparing to detections. This can give an approximate recall measure. It's okay if ground truth is imperfect – you can treat it as *baseline* rather than absolute truth. For example, if field count was 500 and you detect 480, you might be at ~96% of field count (assuming field count itself might be off by ±5%). Use that as an indicator.

- **Precision/Recall Estimation:** If you manage to get some labeled data (even post-hoc from thermal or photos), calculate precision and recall formally. Precision = TP/(TP+FP) – fraction of detections that were correct. Recall = TP/(TP+FN) – fraction of actual penguins that were detected. In wildlife surveys, missing individuals (FN) might be more concerning than a few false alarms, but both matter. Aim for high recall (so you don't undercount), and manage precision by filtering obvious false positives. If precision is low (lots of FPs like rocks), you'll spend more time in manual QC anyway. One could tolerate a slightly lower precision if it means near 100% recall (counting every penguin but also some extra which can be dropped by manual review). However, since a human QC will review, you want to minimize false positives to not overwhelm them. Striking a balance is key and can be informed by these metrics. Maintain a small confusion matrix if possible from verified samples and track it.

- **Cross-Site Calibration:** Different sites or sensors may require slight parameter tweaks – but we want to minimize these. Use validation results to inform if a site-specific profile is needed. For example, if at San Lorenzo the average penguin height came out ~0.55 m and at Caleta ~0.45 m (perhaps juveniles present), you might adjust thresholds slightly. Document any such adjustments and ideally derive them from data (e.g., "Caleta site: observed HAG distribution peaks at 0.45 m, so lower threshold by 0.1 m"). Ensure that these calibrations are decided *using a validation subset* of data, and then locked in for the rest of that site's processing (to avoid biasing results by "peeking" at all data). In other words, treat a small portion of labeled/known data for calibration, then apply blindly to the rest and evaluate.

- **Performance Tracking Over Time:** As you process 2025 campaign data, maintain a log of counts and any validation metrics for each site and sensor. This could be a simple table: for each site-date, field count vs LiDAR count vs difference, plus any sample-based precision/recall. Over multiple sites, you can see if the pipeline tends to over- or under-count systematically. For instance, if LiDAR counts are consistently ~10% higher than field counts, that might indicate minor false positives (perhaps counting some rocks). If consistently lower, perhaps missing some obscured penguins (maybe ones in burrows or heavy shadows?). Tracking this allows you to adjust the pipeline (or at least flag the outputs) in a scientifically defensible way, e.g., "Our LiDAR detection tends to overshoot by ~5%, so final estimates are adjusted accordingly" – but that adjustment should be last resort; better is to improve the algorithm to remove bias. **Trend analysis:** also watch the effect of point density – if one sensor yields lower recall, it might be due to lower point density, so you could incorporate that knowledge (maybe more aggressive thresholds or making up for holes with multisensor fusion).

- **Imperfect Truth Strategies:** In absence of full detection-by-detection ground truth, you might employ **double sampling**: have two independent methods and compare. For example, use the thermal imagery (once available) to produce an independent count and compare to LiDAR count. Or use a subset of data where two humans counted penguins in ortho images and use that as pseudo-truth. Another strategy is *mark-recapture* in imagery: identify individuals in two modalities or times and reconcile counts. These are complex, but since you plan a later fusion, you can for now validate LiDAR by itself as much as possible, knowing that fusion with thermal will likely increase confidence (thermal should catch live bodies even if LiDAR missed them due to being hidden behind another penguin).

- **Continuous QA:** As the campaign progresses, periodically take a sample of new data and validate again. Environments can change (different substrate, different penguin behaviors). Maintaining a validation set for each major condition (rocky vs flat colony, dense vs sparse, etc.) is useful. You don't

need exhaustive ground truth for all, just enough to be confident the pipeline holds. Any major pipeline change (even minor parameter tweaks) should be tested on previous validation data to ensure it didn't degrade performance.

Lastly, involve field experts in the validation loop. Show them maps of detections vs known colony extents; they might spot if the algorithm is consistently missing birds in a certain terrain (e.g., ones in shadow of a cliff, or chicks huddled low). Their feedback can guide targeted improvements (like "increase sensitivity in area X").

9. Practical Recommendations for Penguin Colonies

Penguin colonies introduce some unique challenges – here's how to handle them:

- **Dense Group Handling:** In peak breeding season, penguins can form very dense groups. Expect many detections in close proximity. We've covered watershed splitting to address touching individuals. In addition, consider the **post-processing step of declustering**: if two centroids end up extremely close (e.g., <0.1 m), it might be an indication of an over-split (or possibly two penguins literally shoulder-to-shoulder). If your algorithm splits one bird into two, likely those centroids will be unrealistically close. You could merge or flag such pairs. A distance threshold can be based on penguin body diameter (~0.3 m) – two separate live penguins won't usually have centroids closer than ~0.3 m (they can press together, but their centers can't coincide). So if you get two within, say, 0.15 m, consider merging them as one detection (or mark as uncertain). This can catch some over-segmentation artifacts. Conversely, **under-splitting** (two penguins as one detection) can be partially addressed by looking at blob size and shape as noted; large weird-shaped blobs often mean multiple birds. In dense colonies, be prepared to accept a small error and possibly correct via known density estimates or thermal fusion. It's also valuable to output something like "colony cluster ID" if you detect a big group – that could help an analyst quickly see "this group of 50 detections might actually be 55 penguins – check manually."

- **Terrain and Slopes:** Colonies on slopes (e.g., beach dunes or rocky slopes) require careful ground modeling (as discussed) and also awareness that penguins on the slope might present a tilted posture to the LiDAR. Ensure your height measurement is truly vertical (standard HAG is vertical difference, which is correct). If using a grid DEM, that's usually fine (vertical datum). One potential issue is that a penguin on a steep slope might be partially obscured by the slope or appear with a skewed footprint. However, since the drone LiDAR is overhead, this is minor compared to terrestrial LiDAR. You might find that on slopes, penguin returns concentrate on the upslope side (shadowing on downslope). This could make an individual look more elongated in plan view. Be cautious in shape-based filtering for such cases – an elongated blob might still be one penguin if on a slope. Vegetation on slopes (shrubs, tussocks) could also appear penguin-height. If certain colonies have such vegetation, you might need site-specific rules (e.g., if intensity or return count difference can separate vegetation). At minimum, note those areas for manual check.

- **Rocks and Guano Patches:** Rocks of penguin size are the classic false positive. If large boulders exist, they'll be in the DSM and could appear as "penguin-height" if the ground model sinks between rocks. Mitigations: you can make a **static mask** of known large boulders if you have a pre-survey map (perhaps from photogrammetry or a manual mapping). Lacking that, use LiDAR intensity or multi-return characteristics: rocks often give strong returns and perhaps multiple returns if complex

shape; penguins are biological and might have different reflectance (though with DJI Lidar, intensity is uncalibrated and might not reliably distinguish black feathers vs dark rock). If thermal is integrated later, that will easily distinguish warm penguin vs cold rock – but for LiDAR-alone stage, perhaps be slightly conservative (it's better to false-detect a rock and later have thermal say "no heat = discard" than to miss a penguin). Guano patches (flat areas covered in guano) are highly reflective but usually flat – they shouldn't trigger height thresholds. However, guano on a rock can make a rock look larger. Just be aware in QC of environments that could confuse the sensor. If there are man-made objects (old huts, stakes) in the area, map those out or set the HAG upper threshold to exclude them (e.g., a 1.5 m stake will be out of range).

- **Flight Altitude and Point Density:** Variation in flight altitude (and LiDAR sensor characteristics) means some tiles might have 200 pts/m^2 while others maybe 50 pts/m^2. Lower point density can cause incomplete detection (only a few points hit a penguin). In those cases, the CHM might have holes: e.g., only one or two pixels of a penguin get filled with height, making it a very small blob. To mitigate this, consider using a *slightly larger cell size* for lower-density data (so each penguin still ideally hits multiple cells). If you cannot change cell per tile easily, you could implement a density-based morphological dilation: basically, if a detection consists of just 1–2 pixels but isolated and of the right height, treat it as a valid detection (maybe dilate it to make a slightly bigger blob to compute centroid). Also, for lower densities, rely more on the point cluster method: you could cluster points above ground directly (like the cattle example) to catch cases where raster failed. The pipeline could choose method based on a threshold: e.g., if < X points/m^2, skip raster and do 3D clustering. But having two different approaches might reduce consistency, so test carefully. Ideally, choose a single approach that works for the lowest density you anticipate (maybe TrueView 515 can be flown higher but still gets decent density). If some flights are extremely sparse, highlight those outputs with a warning that "density was low; counts might be underestimates here."

- **Boundary and Tile Overlap Deduplication:** It's common that a penguin near a tile edge might get detected in both adjacent tiles (especially if you include overlap in processing). To avoid double counting, implement a deduplication step. For example, if two detections from different tiles are within, say, one cell size of each other (and roughly same elevation), consider them the same penguin and merge or drop one. Using a small buffer in processing then discarding overlapping zone detections in one tile (as mentioned in determinism section) is one way. The **USGS Lidar spec** explicitly calls for tiled deliveries with no overlap [11] – meaning each point belongs to one tile in final products. You should mimic that: ensure each detection is uniquely assigned to one tile's output. A simple rule: if working with buffered tiles, give each detection an (x,y) and if it lies in the buffer area (e.g., > some threshold from tile center), drop it in favor of the adjacent tile's detection. Another approach: combine all tile outputs and perform a clustering on the centroids themselves to eliminate duplicates. We prefer avoiding duplicates in the first place via buffered processing and careful assignment. **Edge artifacts:** also ensure your HAG thresholding or morphology doesn't behave oddly at edges. With a buffer, you handle it, but double-check that no partial penguin (cut by tile boundary) is lost. Using overlap ensures a penguin at edge is fully seen in one of the tile processes.

- **Site-Specific Overrides:** While the goal is a generalizable pipeline, allow configuration for known site differences. For instance, if one site has smaller penguin species or chicks present, the height range might differ; or if one site is on concrete (no vegetation, easy ground) vs another on brush. Set up a simple config per site (could be a JSON with parameters) so that you can tweak thresholds, etc., in a controlled way. This is more efficient and less error-prone than scattering if/else in code. But keep

these differences minimal. Ideally, 90% of parameters are consistent globally, and you only adjust those absolutely needed for site conditions. This ensures **defensibility** – you can say "we used the same algorithm everywhere, only changing X and Y for Site B due to known terrain differences." That's scientifically acceptable as long as rationale is documented.

- **Upcoming Sensor Fusion:** Knowing that thermal imagery will be used later, design your output to facilitate that. For example, output the time stamp or flight ID associated with each detection (if available), so that fusion can match thermal frames to LiDAR detections by time or position. If thermal is georeferenced, then spatial join is possible, but sometimes temporal sync is used. If each detection has a unique ID and maybe even stores the number of LiDAR points or intensity mean, those could be features the fusion algorithm uses (maybe to help decide if a detection is strong or weak). Essentially, think ahead: provide as much info about each detection as might be useful for a later model to ingest. That includes possibly the **confidence or score** – you could assign a simple confidence (e.g., based on HAG height or blob shape) to each detection, which a fusion algorithm or human QA can use. For example, "score 0.9" might be a textbook penguin-shaped detection, whereas "0.5" might be something slightly off (maybe low height or weird shape). Even if these scores are heuristic, recording them is useful.

By implementing these penguin-specific considerations, you reduce the chance of unpleasant surprises during field deployment (like an entire rocky area being counted as 100 penguins, or dense colony counts being way off). In practice, a combination of robust default parameters and a few targeted adjustments will handle most scenarios.

Prioritized Recommendations (High to Low Impact)

High Impact:
- **Robust Ground Filtering:** Integrate a more robust ground DTM algorithm (e.g., CSF or improved morphological filter) to prevent penguin-sized objects from biasing ground height [5] [6]. This will significantly improve detection accuracy (ensures true HAG values).
- **Instance Separation via Watershed:** Implement marker-controlled watershed for dense groups [2]. This addresses under-counting in colonies – a major accuracy win. Tune it with local maxima so it's reliable and documentable.
- **Reproducibility Safeguards:** Enforce deterministic processing (fixed sorting, identical results across runs) and add unit tests/validation hashes. This is mission-critical for trust in results and later fusion.
- **CRS Handling and Output Validation:** Standardize CRS usage and output format (recommend GeoPackage in native projection, plus optional WGS84 GeoJSON) [9]. Include CRS metadata everywhere to eliminate any geolocation errors. High-impact because a CRS mistake can invalidate entire datasets.

Medium Impact:
- **Parameter Tuning Framework:** Establish a clear, data-driven method to set key parameters (cell size, HAG thresholds, min blob size) using site profiles. This will improve consistency across sites and sensors. Not having to hand-tune each flight will save time and reduce errors.
- **False Positive Reduction:** Add simple shape/size filters (e.g., remove obvious non-penguin blobs by area or height) and consider using LiDAR intensity or multi-echo info to filter non-organics. This will boost precision (fewer spurious detections to review).
- **QA Visualization Tools:** Develop the automated QA plots (HAG maps with overlays, density maps, etc.). While not affecting the algorithm's output, these are medium priority for ensuring quality and defensibility

– they will likely catch issues early and facilitate corrections.
- **Unified Output Schema:** Design a clear output schema (with fields for ID, location, HAG, blob area, etc., plus global metadata). A consistent schema makes it easier to integrate with fusion and for reviewers to understand. Medium impact as it doesn't change detection counts but enhances clarity and downstream use.

Low Impact:
- **Micro-optimizations in Code:** e.g., using numba or vectorization to speed up – only pursue if runtime is an issue. For now, clarity and correctness trump speed, given dataset is manageable.
- **Advanced ML Segmentation (Offline):** Experiment with point-cloud deep learning (e.g., PointNet) as a research exercise, but it's not needed for the field pipeline in 2025. A classical approach as described is sufficient and more transparent.
- **Additional Attributes:** Embedding more attributes (like per-detection intensity variance, or adding a 3D convex hull volume) are nice-to-have for research, but not immediately necessary. Focus on core outputs first, then these can be added if useful for fusion or analysis.
- **Backup Formats:** Providing multiple output formats (shapefile, KML) for stakeholders is low impact on the detection quality, and can be done as needed. GeoPackage/GeoJSON covers most needs; others are convenience.

These recommendations are prioritized to first ensure **accuracy and reliability** of the counts, then **repeatability and clarity**, and finally nice-to-have enhancements.

Implementation Checklist for Code and Tests

- [ ] **Ground Model Generation:** Implement new DTM filtering (or tune existing) – verify no penguin-sized artifacts remain in ground. Test: a synthetic case with a 0.5 m high dummy should result in ground staying low (HAG ~0.5 m).
- [ ] **HAG Computation Validation:** After ground subtraction, confirm HAG values make sense (no negative values beyond tolerance, no systematic offsets). Include a test with known flat ground.
- [ ] **Threshold and Segmentation:** Parameterize height thresholds and ensure they can be configured per site/sensor. Test with edge cases (e.g., a 0.1 m object and a 0.7 m object) to see that one is excluded and the other included.
- [ ] **Connected Components Labeling:** Use a stable algorithm (scikit-image or custom) and verify it yields consistent labels regardless of processing order. Test on a small binary image with known expected regions.
- [ ] **Local Maxima Detection:** Implement a peak-finding on HAG. Write a unit test on a fabricated CHM (e.g., two Gaussian bumps) to ensure it finds the correct number of peaks and none in flat areas.
- [ ] **Watershed Splitting:** Integrate marker-controlled watershed. Test on a known scenario: e.g., two blobs attached by a 0.3 m saddle – ensure they split. Also test on a single smooth blob that it does not over-split.
- [ ] **Detection Merging/Deduping:** If two detections are extremely close (within threshold), have logic to merge or flag. Simulate two centroids within 0.1 m and ensure code merges them.
- [ ] **Tile Overlap Handling:** Implement processing buffers and post-process removal of duplicates on edges. Create a test by splitting a sample dataset into tiles and run both tile-wise and whole – confirm outputs match and no double counts on boundaries.

- [ ] **Deterministic Output:** Ensure sorting of detections (e.g., by x then y). Write a test that shuffles input points or processes tiles in reverse order and confirm the final detections are identical.
- [ ] **Floating Precision Test:** Run the pipeline twice in different float precisions (if possible) or on different machines to ensure stable results to within a tiny epsilon. If differences occur, adjust summation order or increase precision.
- [ ] **Version Lock/Logging:** Make the code print or record library versions used. Test that this metadata is captured.
- [ ] **Output Schema Test:** Validate that the output files contain expected fields (e.g., GeoJSON has `properties` with needed info, GeoPackage table has correct columns). Use a small test output to verify CRS is correctly embedded (open in GIS programmatic check).
- [ ] **Metadata Consistency:** Cross-check that CRS and parameter metadata in outputs match the actual used values (no copy-paste errors). For example, if threshold was 0.2–0.6, ensure those appear in metadata. Automate this insertion to avoid human error.
- [ ] **Reference Frame Check:** For a sample detection, verify coordinates by manually locating that penguin in the point cloud or a map. This ensures no misprojection.
- [ ] **Precision/Recall Sample Test:** If any ground-truth data available, run a small validation function to compute detection metrics. Even if using a mock truth (e.g., manually labeled subset), include this as a regression test (especially after any algorithm change).
- [ ] **Performance Monitoring:** Not critical for correctness, but keep an eye on runtime per tile. Ensure no memory leaks or huge slowdowns – process a test tile multiple times in a loop and monitor memory usage (to ensure e.g. laspy file handles aren't leaking).
- [ ] **Guardrail Hash:** Implement the output hashing of detection coordinates. After any code changes, compare the hash on a standard test tile to expected hash. Include tolerance or systematic update if algorithm intentionally changed.

Each of these checklist items can translate to a code review point or automated test. By ticking them off, you'll ensure the pipeline is robust, reliable, and ready for field deployment at scale.

References and Canonical Sources

- **ASPRS LAS Specification 1.4 (2011)** – Defines LiDAR data formats, classification codes, and CRS storage (with WKT) [10] . Ensures our pipeline adheres to data format standards for CRS and classification.
- **USGS Lidar Base Specification (2020)** – Guidelines for LiDAR data handling and deliverables (e.g., no tile overlaps, intensity normalization) [11] . Provides best practices we incorporated for tiling and data management.
- **Zhang et al. (2016), *Cloth Simulation Filter*** – Presents the CSF algorithm for ground extraction [5] , achieving high accuracy with minimal parameters [6] . Supports our ground model recommendations.
- **Chen et al. (2006), *Marker-Controlled Watershed for Trees*** – Demonstrates using local maxima and watershed to segment individual trees from LiDAR [2] [8] . We analogized this approach to penguin separation.
- **Wang et al. (2024), *UAV LiDAR Cattle Segmentation*** – Conference paper showing height threshold + DBSCAN to detect cattle in 3D [4] . Informs our discussion on point-cloud clustering and its results.
- **Pitkänen et al. (2004)** – Early work on adaptive window local maxima for tree detection. Referenced by MathWorks example, it underpins our variable-window peak finding approach [12] .

- **GeoJSON RFC 7946 (2016)** – Specifies that GeoJSON coordinates are WGS84 in decimal degrees [9]. We abide by this for any GeoJSON outputs to avoid misprojection.
- **Internal Documentation & Team Convention** – (Implied references to your team's standards). E.g., sensor specs for DJI L2 and TrueView 515, and any prior campaign results from 2024 that shaped threshold choices. These are not published but are part of our defensibility (we base decisions on known sensor performance).

Each of these sources either guided the method or serves as a standard to justify our pipeline choices. By grounding our approach in published algorithms and standards, we ensure scientific defensibility and alignment with industry best practices.

---

[1] [3] (PDF) Raster Vs. Point Cloud LiDAR Data Classification
https://www.researchgate.net/publication/286897772_Raster_Vs_Point_Cloud_LiDAR_Data_Classification

[2] [8] Isolating Individual Trees in a Savanna Woodland Using Small Footprint Lidar Data
https://pdfs.semanticscholar.org/9106/7a9e7f60b43cf40292d47aea790549b2c7b0.pdf

[4] Automated cattle segmentation from UAV-based LiDAR point clouds - Wageningen University & Research
https://research.wur.nl/en/publications/automated-cattle-segmentation-from-uav-based-lidar-point-clouds/

[5] [6] An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation
https://www.mdpi.com/2072-4292/8/6/501

[7] GFM_Wen Xiao_25400.pdf
http://essay.utwente.nl/93606/1/Wen%20Xiao_25400.pdf

[9] Everything you need to know now about RFC 7946 GeoJSON - macwright.com
https://macwright.com/2016/11/07/the-geojson-ietf-standard

[10] Georeferencing LAS files with LAS 1.4 — liblas.org
https://liblas.org/development/wkt.html

[11] USGS NGP Base Lidar Specification
https://transportation.ky.gov/Highway-Design/Documents/KY%20DGI%20LiDAR%20SPECS.pdf

[12] Extract Individual Tree Attributes and Forest Metrics from Aerial Lidar Data - MATLAB & Simulink
https://www.mathworks.com/help/lidar/ug/extraction-of-forest-metrics-and-individual-tree-attributes.html