ANSLPR — ALPRChecker & Plate Format Developer Guide

Module: **ANSLPR** · Base class: **ANSCENTER::ANSALPR** · Affects every engine subclass (**ANSALPR_OD**, **ANSALPR_OCR**, **ANSALPR_CPU**)

This guide documents two independent post-processing features in the ANSLPR pipeline: the **ALPRChecker** (per-track majority voting + lock + skip-when-locked) and the **plate format validator** (`SetPlateFormat` / `SetPlateFormats`). They share neither state nor toggle; the format validator runs as a filter *before* the ALPRChecker voter sees any text.

Part 1 — ALPRChecker (tracker-based voting)

1.1 Purpose

Convert noisy per-frame OCR strings for the same physical plate ("50H 304.07", "50H304O7", "50H304.07"…) into a single stable *locked* text. The mechanism is layered:

track identity → **rolling history per track** → **recency-weighted majority vote** → **stable lock with edit-distance noise rejection** → **periodic re-verify** so wrong locks can self-correct.

1.2 Default state

Flag	Default	Source
`_enableALPRChecker`	**false** (opt-in)	Member initializer in ANSLPR.h (`std::atomic<bool> _enableALPRChecker{false}`)
`Full-frame auto-detect`	engages after a few pixel-identical frames (only relevant once enabled)	shouldUseALPRChecker / ImageSizeTracker
`Skip-when-locked re-verify period`	**30 frames**	tryReuseLockedText default `reverifyEvery=30`

> **Out of the box:** ALPRChecker is **OFF**. The pipeline emits raw recognizer text per frame — no voting, no lock, no skip-when-locked, no spatial dedup. Callers must opt in with `SetALPRCheckerEnabled(true)` (or `ANSALPR_SetALPRCheckerEnabled(h, 1)`) to enable cross-frame stabilization. Once enabled, full-frame auto-detect still gates it on per camera, so pre-cropped pipeline calls remain stateless even when the flag is on.

1.3 Public API

C++ class API (`ANSCENTER::ANSALPR`):

Method	Effect
`void SetALPRCheckerEnabled(bool enable)`	Atomic flip of `_enableALPRChecker`. Logs `[ALPR_Checker]` with prev/new.
`bool IsALPRCheckerEnabled() const`	Atomic read. Logs the current value.

C exports (LabVIEW / .NET / other clients):

Function	Args	Return
`ANSALPR_SetALPRCheckerEnabled`	`Handle, int enable (0|1)`	1 OK / -1 invalid handle / 0 exception

Function	Args	Return
ANSALPR_GetALPRCheckerEnabled	Handle	1 enabled / 0 disabled / -1 invalid handle

1.4 Internal state (per ANSALPR instance)

```
class ALPRChecker {
    int   maxFrames           = 60;     // rolling history depth
    int   minVotesToStabilize = 3;      // identical reads to lock
    float iouMatchThreshold   = 0.30f;  // spatial match (legacy API)
    std::recursive_mutex _mutex;        // protects all maps

    // trackId-based primary path
    unordered_map<string, unordered_map<int, TrackedPlateById>>
        trackedPlatesById;

    // spatial-IoU legacy path
    unordered_map<string, vector<TrackedPlate>> trackedPlates;
};

struct TrackedPlateById {
    int               trackId;
    deque<string>     textHistory;          // RAW reads, capped
    string            lockedText;            // "" until locked
    int               lockCount;
    int               framesSinceLastSeen;   // for prune
    int               framesSinceVerify;     // for skip-when-locked
};
```

1.5 Core algorithm — checkPlateByTrackId

Primary entry point. Hybrid identity strategy: trackId hash lookup (O(1) common case), Levenshtein-1 fallback for lost tracks.

```
LOCK _mutex
if detectedPlate empty: return ""

plates = trackedPlatesById[cameraId]
for (id, p) in plates: p.framesSinceLastSeen++           // age all

// Periodic prune: every 30 calls, drop entries unseen > 180 frames
_pruneCounterById++
if _pruneCounterById >= 30 and plates.size() > 20:
    drop entries where framesSinceLastSeen > maxFrames * 3

// PRIMARY: O(1) trackId hit
if trackId in plates:
    tp = plates[trackId]
    tp.framesSinceLastSeen = 0
    tp.textHistory.push_back(detectedPlate)              // RAW
    if size > maxFrames: pop_front()

    voted = majorityVote(tp.textHistory)

    if tp.lockedText empty:
        if count(history == voted) >= minVotesToStabilize:  // 3
            tp.lockedText = voted        // LOCK
        return voted
    else:
        if Levenshtein(detectedPlate, lockedText) == 0:
            tp.lockCount++; return lockedText
        elif Levenshtein(voted, lockedText) > 1:
            // Re-lock if 3 of last 6 votes agree on new text
```

```
            if count(recent6 == voted) >= 3:
                tp.lockedText = voted    // RE-LOCK
                return lockedText
        return lockedText                // emit lock, ignore noise

// FALLBACK: trackId changed (occlusion, ID switch). Migrate.
for (id, p) in plates:
    if Levenshtein(detectedPlate, p.lockedText) <= 1:
        migrate p to plates[trackId]; erase old id
        return p.lockedText
    elif p.lockedText empty and any of last 3 history within edit-1:
        migrate; return majorityVote(migrated.history)

// No match anywhere: fresh entry
plates[trackId] = { trackId, history=[detectedPlate], locked="" }
return detectedPlate
```

1.6 majorityVote

Recency-weighted vote with Levenshtein-1 group boosting. Newest entries get weight 2.0, oldest 1.0. Variants within edit distance 1 donate 0.5× their score to each candidate, so 1-character OCR drift fuses instead of competing.

```
weightedFreq = {}
for i in 0..n-1:
    weight = 1.0 + i / max(1, n-1)        // newest=2.0, oldest=1.0
    weightedFreq[history[i]] += weight

for kv in weightedFreq:
    total = kv.score
    for other in weightedFreq:
        if abs(len(kv) - len(other)) <= 1
           and Levenshtein(kv, other) == 1:
            total += other.score * 0.5
    track best
return best
```

1.7 tryReuseLockedText — skip-when-locked fast path

Saves ~50–95 ms per locked plate per frame by skipping the OCR call entirely when a stable lock exists. Periodic re-verify (every 30 frames by default) lets one OCR run through so a wrong lock can self-correct via the normal voting path.

```
LOCK _mutex
tp = trackedPlatesById[cameraId][trackId]    // miss -> return ""
if tp.lockedText empty: return ""             // not locked yet

if reverifyEvery > 0 and tp.framesSinceVerify >= reverifyEvery:
    tp.framesSinceVerify = 0
    return ""        // signal: caller runs OCR this frame

tp.framesSinceVerify++
tp.framesSinceLastSeen = 0     // critical: keep entry alive across
                               //  long skip-when-locked windows
return tp.lockedText            // skip OCR; emit lock directly
```

1.8 Pipeline integration

Pipeline mode always bypasses the ALPRChecker, regardless of how `_enableALPRChecker` is set. Two independent mechanisms enforce this:

Mechanism 1 — dedicated batch entry point (hard bypass). `ANSALPR_RunInferencesBatch_LV` / `_LV_V2` (`dllmain.cpp:836, 1405`) routes through `engine->RunInferencesBatch(...)`. The implementations at `ANSALPR_OD::RunInferencesBatch` (line 2637) and `ANSALPR_OCR::RunInferencesBatch` (line 1863) contain **zero references** to the checker — no `shouldUseALPRChecker`, no `checkPlateByTrackId`, no `tryReuseLockedText`, no `ensureUniquePlateText`. The flag is never read on this path. Header contract at `ANSLPR.h:163-178`: "Tracker, voting, spatial dedup, and per-camera accumulating state are all bypassed."

Mechanism 2 — auto-detect floors on the regular `RunInference` path. `shouldUseALPRChecker` applies a chain of gates; the last two reject pipeline crops even when the flag is on:

```
if !_enableTrackerVoting:                  return false  // OD master (1)
if !_enableALPRChecker.load(relaxed):      return false  // user flag (2)
if imageSize.width < MIN_FULLFRAME_WIDTH:  return false  // HARD FLOOR (3)
                                                          //   = 1000 px
tracker = _imageSizeTrackers[cameraId]
if imageSize == tracker.lastSize:                          // CONSISTENCY (4)
    if ++tracker.consistentCount >= CONFIRM_THRESHOLD:     //   = 5 frames
        tracker.detectedFullFrame = true
else:
    tracker.lastSize = imageSize; consistentCount = 1     // reset on any
    tracker.detectedFullFrame = false                      //  mismatch
return tracker.detectedFullFrame
```

Gates (3) and (4) make the bypass automatic for cropped inputs:

Pipeline indicator	Effect	Independent of flag?
`width < 1000 px` (`MIN_FULLFRAME_WIDTH`)	Returns `false` immediately, before the auto-detect state machine runs.	**Yes** — rejects all small crops unconditionally after the flag check passes.
Frame size varies between calls	`consistentCount` resets to 1 on every mismatch; `detectedFullFrame` stays `false`. Pre-cropped vehicle ROIs (e.g. 496×453, 497×455, …) never accumulate consistency.	**Yes** — the consistency requirement is independent of the user flag.

> **Practical guarantee.** Even if a caller sets `SetALPRCheckerEnabled(true)` and feeds pipeline crops, voting stays off per camera because the auto-detect never flips `detectedFullFrame` to true. Mixing full-frame and pipeline calls on different cameras works cleanly: the per-camera `_imageSizeTrackers` entry tracks each camera's mode independently.

Caller path × flag matrix

Caller path	Input characteristic	Flag = OFF	Flag = ON
`RunInferencesBatch` (pipeline batch, LV/V2)	any size, any consistency	**`bypassed (hard)`**	**`bypassed (hard, flag never read)`**
`RunInference`	full video frames (≥ 1000 px, consistent size)	`bypassed`	`active after 5 consistent frames`
`RunInference`	cropped vehicle ROI (< 1000 px)	`bypassed`	**`bypassed by MIN_FULLFRAME_WIDTH floor`**

Caller path	Input characteristic	Flag = OFF	Flag = ON
`RunInference`	varying-size crops (≥ 1000 px but inconsistent)	`bypassed`	**`bypassed by consistency reset`**

In effect, the flag answers a single question: *"if the caller is sending real video frames from this camera, should I stabilize the output?"* It has no effect on pipeline / cropped inputs — those are always raw OCR.

1.9 OCR engine wiring (per frame)

```
useChecker = shouldUseALPRChecker(frameSize, cameraId)  // once

for each detected plate box:
    if useChecker:
        locked = tryReuseLockedText(cam, trackId)
        if locked not empty:
            emit { className=locked, +color }; SKIP CROP
            continue
    build crop(s) (padded, optionally rectified, top/bottom split)

ONE batched recognizer call across all crops

for each plate (non-skipped):
    text = StripPlateArtifacts(top + " " + bottom)
    if Country == JAPAN and IsJapaneseIncomplete(text):
        text = recover kana from bottom-half tile
    text = StripPlateSeparators(text)
    text = AnalyseLicensePlateText(text)        // <-- format filter
    if text empty: continue                      //     (drops invalid)

    if useChecker:
        className = checkPlateByTrackId(cam, text, trackId)
    else:
        className = text                          // raw
    emit { className, +color }

if useChecker:
    ensureUniquePlateText(output, cam)            // spatial dedup
```

1.10 OD engine wiring

Same gate at every `alprChecker.checkPlateByTrackId` call. The OD path doesn't use skip-when-locked because it already runs character classification per detection — nothing to skip. Note OD also has a separate master `_enableTrackerVoting` flag checked *before* `_enableALPRChecker`; if that one is off, the disabled-path applies regardless.

1.11 Disabled mode (default) vs enabled — what runs

Stage	Enabled (opt-in)	Disabled (default)
LP detection	Runs	Runs
ROI cropping + 5% padding	Runs	Runs
Japan rectification + kana recovery	Runs (Japan only)	Runs (Japan only)
Batched recognizer call	Runs	Runs
StripPlateArtifacts / StripPlateSeparators	Runs	Runs
AnalyseLicensePlateText (format filter)	Runs (independent toggle)	Runs (independent toggle)

Stage	Enabled (opt-in)	Disabled (default)
Skip-when-locked (tryReuseLockedText)	Runs	**Bypassed**
Per-track voting (checkPlateByTrackId)	Runs	**Bypassed** — raw recognizer text
Spatial dedup (ensureUniquePlateText)	Runs	**Bypassed**
trackedPlatesById state	Populated and pruned	Untouched (existing entries frozen)

Cost. Disabling skip-when-locked adds ~50–95 ms per locked plate per frame because every plate now goes through OCR every frame.

1.12 Thread safety

Concern	Resolution
Toggle race between config thread and inference thread	`_enableALPRChecker` is `std::atomic<bool>`; store/load use `memory_order_relaxed`. Tear-free, no fence cost.
Concurrent track map mutation	`ALPRChecker::_mutex` (recursive) wraps every read & write of `trackedPlatesById` / `trackedPlates`.
Concurrent spatial dedup state	`_plateIdentitiesMutex` in `ANSALPR_OCR` protects `_plateIdentities`.
Toggle while OCR is in flight	Safe. The in-flight frame uses whichever value it observed; the next frame picks up the new value.

1.13 Diagnostic logging (DebugView)

Tag	Emitted by
`[ALPR_Checker]`	Set/Get accessors and C-export wrappers — every flag transition and every read.
`[ALPR_TrackId]`	checkPlateByTrackId — LOCKED, RE-LOCK, MIGRATED, noise, NEW; tryReuseLockedText — SKIP-LOCKED, REVERIFY.
`[ALPR_OCR_Checker]`	shouldUseALPRChecker — full-frame vs pipeline mode auto-detect transitions per camera.
`[ALPR_Format]`	AnalyseLicensePlateText — REJECT lines when a plate doesn't match any configured format (see Part 2).
`[OCR_Leak]`	Once-per-60s heartbeat with sizes of each map — useful for spotting unbounded growth in production.

Tags are gated behind `ANSCORE_DEBUGVIEW` and `ANSCENTER_IsDebugViewEnabled()`. Filter *DebugView* on the bracketed tag to isolate any layer.

Part 2 — Plate format validation

2.1 Purpose

Reject OCR output that doesn't match any configured plate pattern. Acts as a hard filter **upstream** of the ALPRChecker voter, so malformed reads never enter `textHistory` and a track never locks on garbage.

2.2 Pattern syntax

Patterns are simple character templates. Each character in the pattern matches one character in the candidate plate; lengths must match exactly.

Token	Matches	Notes
`d`	Any digit (`0–9`)	`std::isdigit`
`l`	Any letter (`A–Z, a–z`)	`std::isalpha`. Plate text is uppercased before matching.
`any other char`	Literal — must match exactly	Use uppercase literals (e.g. `M`, `D`, `NG`) since the plate is uppercased first.

> **Empty format list** = no validation; all plates accepted (`MatchesPlateFormat` returns true). Calling `SetPlateFormats(emptyList)` disables filtering.

2.3 Examples

Pattern	Matches	Doesn't match
`ddlddddd`	`30A12345 | 51F67890`	`30A1234 (too short) | 30AA12345 (2nd letter where digit expected)`
`lldddd`	`AB1234 | QC9876`	`A12345 (1st char must be letter) | AB12345 (too long)`
`dddddNGdd`	`12345NG67`	`12345NN67 (literal mismatch)`
`ddMDdddddd`	`12MD345678`	`12mD345678 (case mismatch on literal — plate is uppercased, so 'D' must be 'D')`

2.4 Default formats per country

`ApplyCountryDefaultPlateFormats()` populates the format list automatically based on `_country`. Called at the end of `Initialize()` (after country.txt is read) and from `SetCountry()`, so a runtime country switch picks up the right format set on the very next frame.

Country	Formats installed
VIETNAM	ddlddddd, ddldddd, ddlddddd, ddllddddd, ddMDdddddd, dddddNGdd, dddddQTdd, dddddCVdd, dddddNNdd, lldddd (11 patterns covering car / motorbike / military / diplomatic / police)
JAPAN, CHINA, AUSTRALIA, USA, INDONESIA, …	*Empty* — no validation. `AnalyseLicensePlateText` is also a near-no-op for non-Vietnam countries (preserves kana / kanji / region-specific punctuation verbatim).

2.5 Public API

C++ class API (`ANSCENTER::ANSALPR`):

Method	Effect
`void SetPlateFormat(const string& format)`	Replace the list with a single pattern.
`void SetPlateFormats(const vector<string>& formats)`	Replace the entire list. Pass empty vector to disable validation.
`vector<string> GetPlateFormats() const`	Returns the current list.
`virtual void SetCountry(Country c)`	Calls ApplyCountryDefaultPlateFormats() so the list resets to the country's defaults.

C exports:

Function	Args	Return
`ANSALPR_SetFormat`	`Handle, const char* format`	1 OK / -1 invalid handle / 0 exception
`ANSALPR_SetFormats`	`Handle, const char* formats (*semicolon-separated*, e.g. "ddlddddd;ddlddddd;llddddd")`	1 OK / -1 invalid handle / 0 exception
`ANSALPR_GetFormats`	`Handle, LStrHandle outFormats`	Count written, semicolon-separated
`ANSALPR_SetCountry`	`Handle, int country (0=VN, 1=CN, 2=AU, 3=US, 4=ID, 5=JP)`	1 OK / -1 invalid handle / 0 exception

2.6 Where it runs in the pipeline

Sequential post-processing inside the per-plate loop — **before** the ALPRChecker voter:

```
topText, bottomText  = OCR results per row
combinedText = StripPlateArtifacts(top + " " + bottom)   // °, ■, screws
[Japan] kana recovery if hiragana missing
combinedText = StripPlateSeparators(combinedText)         // -, comma, ws
                              v
combinedText = AnalyseLicensePlateText(combinedText)      // <-- PART 2
    1. keep only alnum chars
    2. uppercase
    3. if _plateFormats not empty:
         if no pattern matches: log REJECT, return ""
    return cleaned plate
                              v
if combinedText empty: continue            // dropped — voter never sees it
                              v
if useChecker:                               // <-- PART 1
    className = checkPlateByTrackId(cam, combinedText, trackId)
else:
    className = combinedText
```

Consequence. When a format list is configured, the voter's `textHistory` contains only format-valid strings. `lockedText` is therefore always format-valid by construction. A track that consistently produces format-invalid OCR will never lock — it just never reaches the voter, and no output is emitted for that plate.

2.7 Country interaction

`AnalyseLicensePlateText` behaves differently per country (see `ANSALPR_CPU.cpp`, `ANSALPR_OCR.cpp`, `ANSALPR_OD.cpp`):

Country	AnalyseLicensePlateText behavior
VIETNAM	Strips non-alnum, uppercases, applies suburb-code remapping (13→73, 42→62, 44→64, …), then validates against `_plateFormats`. Mismatch → ""
Other countries	Returns input unchanged so kana / kanji / region punctuation flow through verbatim. `_plateFormats` is empty by default for these countries; even if you populate it manually, the format check still runs in OCR (same code path) but is currently a no-op for non-Vietnam in OD.

2.8 ALPRChecker × format combined behavior

Format set	ALPRChecker	Behavior
`empty`	`OFF`	Raw OCR every frame. No filter, no voting. Highest noise, highest recall.
`empty`	`ON`	Voted/locked text. Voter learns from raw recognizer output — can lock on bad reads if they're consistent.
`configured`	`OFF`	**Default for Vietnam.** Format-filtered raw OCR per frame. Invalid reads dropped, valid ones emitted as-is (no cross-frame memory).
`configured`	`ON`	**Recommended for stable cameras.** Format filters bad reads out, voter locks the surviving canonical text. Best stability, lowest false-positive rate. Opt in with `SetALPRCheckerEnabled(true)`.

Toggling at runtime. Both flags are independent and thread-safe. Existing locks survive a format-set change — they're not retroactively re-validated. To force a clean slate, destroy and recreate the ANSALPR handle (or wait for the prune pass to expire stale tracks).

2.9 Caller recipes

Goal	C export sequence
Default pipeline (raw OCR + format filter, no voting)	`Nothing — defaults are fine.`
Enable tracker voting + skip-when-locked	`ANSALPR_SetALPRCheckerEnabled(h, 1)`
Switch country at runtime	`ANSALPR_SetCountry(h, 5)  // Japan; resets formats`
Vietnam-only police plates	`ANSALPR_SetFormats(h, "dddddCVdd")`
Vietnam plus a custom long-form pattern	`ANSALPR_SetFormats(h, "ddlddddd;ddlddd;dddddCVdd;ddlddddddddd")`
Disable format validation entirely	`ANSALPR_SetFormats(h, "")  // empty -> accept all`
Disable tracker voting (revert to raw OCR every frame)	`ANSALPR_SetALPRCheckerEnabled(h, 0)`
Read current state	`ANSALPR_GetALPRCheckerEnabled(h);` `ANSALPR_GetFormats(h, &buf)`

2.10 Diagnostic logging (DebugView)

```
[ANSLPR]        ANSALPR_SetFormat: HandlePtr=0x... format=ddlddddd
[ANSLPR]        ANSALPR_SetFormats: HandlePtr=0x... formats=ddlddddd;lddddd
[ANSLPR]        ANSALPR_SetCountry: HandlePtr=0x... country=0
[ALPR_Format]   REJECT plate='30AA1234' (no _plateFormats match)
```

Filter on `[ALPR_Format]` to count how many plates the format filter is rejecting per camera per minute. A high reject rate with a low ALPR result count means the format list is too tight for what the recognizer is producing.

This guide reflects the ANSLPR API as of the ALPRChecker atomic + getter export change. File references point to modules/ANSLPR/ANSLPR.{h,cpp}, modules/ANSLPR/ANSLPR_OCR.{h,cpp}, modules/ANSLPR/ANSLPR_OD.{h,cpp}, modules/ANSLPR/ANSLPR_CPU.cpp, and modules/ANSLPR/dllmain.cpp.