Senegalese License Plate Detection System

Manual Parsing and Regex Approaches

Group 11: A. Irigenera, M. Y. Mohamed, O. T. Ogundepo, J. B. Habineza

AIMS Rwanda

October 8, 2025

Agenda

Motivation

- Extract structured information (license plates) from plain text.
- Useful for logs, reports, monitoring, and preprocessing.
- Focus: Senegalese vehicle plate formats only.
- Two complementary methods: manual parser and regex pattern.

Plate Formats

Accepted canonical patterns (letters A–Z, digits 0–9):

- `XY-1234-T`
- `XY-1234-ZT`

Separators may be hyphen or single space in input: `XY 1234 T`.

Rules Recap

- 2 starting letters
- Separator: – or space
- 4 digits
- Separator: – or space
- 1 or 2 trailing letters
- Not embedded inside longer alphanumeric strings
- Case-insensitive input; uppercase normalized output

Core Objectives

- Detect all valid plates in arbitrary text.
- Print normalized unique results in order.
- Report when none are found.
- Provide simple, readable logic.

Extended Objectives

- Allow punctuation around plates.
- Interactive menu (manual testing).
- Provide alternative regex method.
- Keep code dependency-free (core version).

Why Manual Parsing?

- Educational transparency.
- Fine control over each rule.
- Easy to extend for new patterns.
- Avoids reliance on regex engine.

Pseudocode (High-Level)

```
UPPER = text uppercased
plates = empty list
i = 0
while i < length - minimal_length:
  if next 2 chars not letters: i++ ; continue
  if next sep not '-' or ' ': i++ ; continue
  if next 4 chars not digits: i++ ; continue
  if next sep not '-' or ' ': i++ ; continue
  read 1 or 2 letters as end_part
  if none: i++ ; continue
  check boundary before and after
  if ok:
     plate = canonical form with hyphens
     store if new
     i = end of match
```

Boundary Handling

- Ensure no letter/digit directly touches start or end of detected pattern.
- Prevents false matches inside longer tokens (e.g. `AAXY-1234-T`).
- Accepts punctuation (comma, period) near plates.

Regex Pattern

Core pattern (case-insensitive):

Pattern

```
(?i)(?<![A-Z0-9])([A-Z]{2})[- ](\d{4})[- ]([A-Z]{1,2})(?![A-Z0-9])
```

Pattern Components

- `(?i)` Case-insensitive matching
- `(?<![A-Z0-9])` Left boundary (no letter/digit before)
- `([A-Z]2)` Two letters
- `[- ]` Separator (hyphen or space)
- `(4)` Four digits
- `([A-Z]1,2)` One or two letters
- `(?![A-Z0-9])` Right boundary (no letter/digit after)

- Lookbehind / lookahead enforce clean boundaries.
- Captures letter block, digits, ending letters.
- Short, expressive, fast.

Manual vs Regex

Manual	**Regex**
Stepwise logic	Compact expression
Easy to tweak mid-steps	Faster to write
Verbose	Dense syntax
Didactic	Concise

Test Design

Mixed sample paragraph included:

- Valid plates with hyphens and spaces.
- Single-letter and two-letter endings.
- Mixed casing.
- Near-miss invalid patterns.

Observed Outcomes

- All expected valid plates detected.
- No false positives in sample.
- Output normalized consistently.
- Order preserved; duplicates removed.

Current Limitations

- No fuzzy/typo tolerance.
- Only Senegal formats covered.
- Pattern validity only (not real issuance).

Potential Enhancements

- Add fuzzy (edit-distance) matching.
- Multi-country format registry.
- Batch file / dataset scanning.
- JSON / CSV export.
- Performance benchmarking.

Conclusion

- Two clear methods implemented: manual parser and regex.
- Both meet detection, normalization, and uniqueness goals.
- Manual path aids learning; regex path aids brevity.
- Solid base for future extensions (fuzzy logic, more formats).

Key Takeaways

- Keep patterns explicit when teaching.
- Normalize early for consistency.
- Enforce boundaries to avoid false positives.
- Provide alternative implementations for flexibility.

Questions

Thank You!

Questions / Feedback?