Truck Inspection PoC — Step-by-step Guide

==

Author: Truck PoC Bot

Date: 2025-12-31

Overview

This document describes the step-by-step Proof-of-Concept (PoC) to build a truck interior inspection system using pattern matching and object detection (YOLOv8 + OpenCV). The repo contains detection code, a FastAPI REST API, a static HTML demo, a Streamlit UI, sample generation script and CI/E2E tests. Use this guide to reproduce the PoC locally or in CI.

Prerequisites

- Python 3.10+ (3.11 recommended)
- Git
- Docker (optional, for container builds)
- A virtual environment (venv) is recommended

Key components

- src/: implementation (api, detection, training helpers, demo server)
- demo/: static JS demo to call the API
- scripts/generate_samples.py: generate synthetic samples for tests/demo
- tests/: pytest unit tests and Playwright E2E test
- .github/workflows/: CI, E2E and release automation

Step-by-step PoC instructions

----

1) Clone the repo

- git clone https://github.com/ramachennupati/PatternDetect_POC.git
- cd PatternDetect_POC

2) Create & activate a virtualenv

- python -m venv .venv
- Windows: .\.venv\Scripts\Activate.ps1 (PowerShell) or .\.venv\Scripts\activate
- Mac/Linux: source .venv/bin/activate

3) Install dependencies

- python -m pip install --upgrade pip
- pip install -r requirements.txt

4) Generate or prepare sample images

- The repo contains `data/samples/` with example images and `data/template.png`.

- If missing, run: python scripts/generate_samples.py --out data/samples --count 3

5) Run the API locally

- uvicorn src.api:app --host 0.0.0.0 --port 8000

- Endpoints:

- GET /health → health check

- POST /detect (multipart/form-data) → JSON with detections + base64 annotated image

- POST /detect/image → returns raw annotated JPEG

6) Run the static demo

- Serve the demo folder from the repo root: python -m http.server 8004 --directory demo

- Open http://127.0.0.1:8004 and set API Base URL to http://127.0.0.1:8000

- Choose `/detect` and upload a sample image; click Run → annotated image / JSON appears

7) Run Streamlit UI (optional)

- python -m streamlit run src.api_interface.py

- Open the URL printed by Streamlit in the terminal and interact with the interface

8) Run unit tests and E2E tests

- pytest -q # runs unit tests

- For Playwright E2E test (headless): pytest tests/test_demo_e2e.py -q

- On CI we use GitHub Actions to run tests automatically on push/PR

9) Build Docker image (optional)

- docker build -t yourrepo/patterndetect:latest .

- To publish via CI, add Docker Hub secrets to repository: DOCKERHUB_USERNAME, DOCKERHUB_TOKEN

10) Create a Release (CI)

- Create a git tag (semantic; e.g., v0.1.3) and push it: git tag v0.1.3; git push origin v0.1.3

- The `.github/workflows/release-and-docker.yml` workflow will create a GitHub Release and build/push Docker image (if secrets are present)

Implementation notes & tips

- Detection: the PoC uses a pre-trained YOLOv8 (ultralytics) model; `yolov8n.pt` is included as a small model for demos.

- Pattern matching: a template-based approach using OpenCV is implemented (works well for rigid templates; consider feature-based matching (ORB/SIFT) for robustness).

- CORS: API enables CORS so the static demo can call it from the browser.

- Automated sample generation: scripts/generate_samples.py lets CI create deterministic test images for E2E tests.

Troubleshooting

- CI failures due to missing `pytest` or `playwright` → ensure `requirements.txt` includes them (we keep them in the repo).

- Playwright timeouts → increase selector timeouts or debug by running Playwright locally with headful browser.

- Docker publish fails in Actions → add `DOCKERHUB_USERNAME` and `DOCKERHUB_TOKEN` secrets in repo settings.

Next steps & enhancements

- Improve detection accuracy by training on more labeled images (use `train.py`).

- Add more robust pattern matching (keypoint descriptors like ORB/SIFT or modern feature matchers).

- Create an integration test that validates detection accuracy metrics over a curated dataset.

- Add deployment recipes (k8s manifest, GitHub Pages hosting for demo, or Docker Compose stacks).

Appendix: useful commands

- Run API: uvicorn src.api:app --host 0.0.0.0 --port 8000

- Run demo: python -m http.server 8004 --directory demo

- Generate samples: python scripts/generate_samples.py --out data/samples --count 3

- Run unit tests: pytest -q

- Run E2E test: pytest tests/test_demo_e2e.py -q

License & attribution

Licensed under MIT (see `LICENSE` in repo) — adapt and reuse freely.

Contact

If you want additional formatting, a shorter executive summary, or a slide deck derived from this guide, tell me which format you prefer and I’ll prepare it.