Design Document

Distribute Image System (underwater-ipc)

1. **Introduction**

   This document describes the overall architecture and design of the *Distributed Image System*. The system is built using modern C++ and is designed to simulate real-world image streaming, processing and storage in a distributed, fault-tolerant manner.

   The system consists of three independent applications that communicate using Inter-Process Communication (IPC). Each application runs as a separate process and can start, stop or restart without affecting the other.

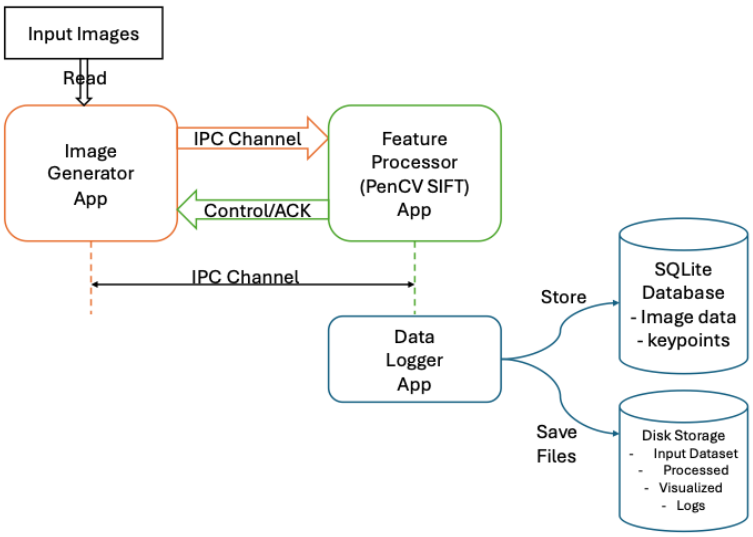
2. **High-Level System Overview**

   The system follows a pipeline-based architecture.

   Each stage performs a single responsibility.

   1. **Generator** – Produces image data
   2. **Processor** – extracts feature from images
   3. **Logger** – stores the result for future use
   4. **Storage**:
      - SQLite: Image data + keypoints data
      - Disk Storage: Processed Images, Logs
   5. **Additional**: Visualization (Explainability)
      - Visualization: Keypoints visualization.
      - Visualized Images stored on Disk.

   System Architecture Diagram:



This design allows:

- Easy debugging
- Independent scaling
- High modularity
- Fault isolation

3. Core Architectural Principles

The system was designed using the following principles:

- Loose Coupling: Each application is independent
- Single Responsibility: Each application does only one job.
- Fault Tolerance: If one app crashes, others continue running.
- Configurability: Runtime behaviour is controlled via a JSON config file.
- Portability: Designed to run on Linux and macOS.

4. Components Description

1) Image Generator

Purpose:

Simulates a real camera or sensors but continuously reads images from disk and publishes to the system.

Main Responsibilities:

- Reads images from:

  underwater_images/

- Converts images into binary format.
- Publish image data via IPC
- Loops continuously through the dataset.

Key Characteristic:

- Handles small and large images.
- Runs continuously until manually stopped.
- Does not depend on the Processor or Logger to be running.

2) **Feature Processor**

Purpose:

Receives image data and extracts important visual features using OpenCV SIFT.

Main Responsibilities:

- Subscribes to image data from Generator.
- Uses SIFT (Scale-Invariant Feature Transform) to extract:
  - Keypoints
  - Descriptors.
- Publishes
  - Original Image
  - Extracted keypoints.

Key Characteristics:

- Pure processing unit (no storage)
- Works independently
- It can be extended with new algorithms in the future.

3) **Data Logger**

Purpose:

Stores processed results for future analysis.

Main Responsibilities:

- Subscribes to processed data from Processor.
- Stores metadata into a SQLite database

- Saves:
    - Raw processed images
    - Visualized keypoint images

Storage Location:

data/database.db

processed_images/processed/

processed_images/visualized/

logs/logger.log

Key Characteristics:

- Persistent Storage.
- Supports future analytics and reporting.
- Does not perform any image processing.

5. **Inter-Process Communication (IPC)**

The system uses IPC messaging to send data between applications.

Communication Flow:

- Generator → Processor
- Processor →Logger

Data Transmitted:

- Binary image data
- Image metadata
- SIFT keypoints
- Processing timestamps

Why IPC was chosen (ZeroMQ)

- High performance
- Low Latency
- Decoupled execution
- Real-time data flow

- Works between independent processes

6. **Configuration Design**

   All runtime behaviour is controlled using a central file:

   config/default_config.json

   This file manages:

   - Communication ports.
   - Input/output paths
   - Database location
   - Logging configuration

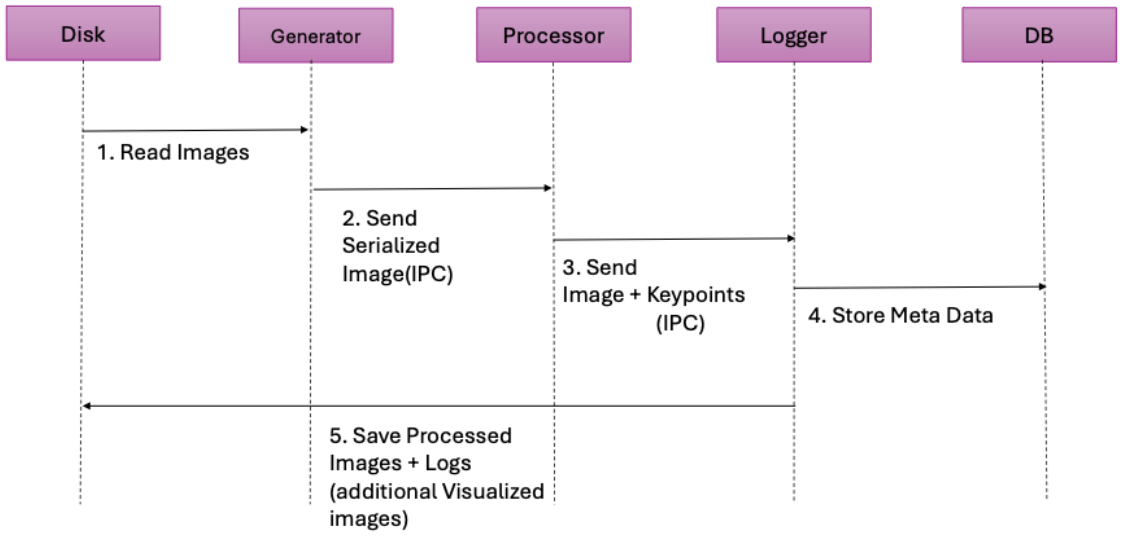
   This allows:

   - Running the same binary files in different environments
   - Easy changes without recompiling code.

7. **Data Flow Process**

   1) The generator reads an image from disk
   2) Image is serialized and sent to the Processor.
   3) Processor:
      a. Deserialize image
      b. Runs SIFT feature extraction
      c. Packages image + keypoints
   4) Data is sent to the Logger.
   5) Logger:
      a. Saves data to SQLite
      b. Writes processed images to disk
      c. Records logs

   Sequence Diagram:



Sequence Diagram: Distributed Image System (Underwater -IPC)

8. **Logging and Monitoring**

   Each application maintains its own log file for debugging and traceability:

   logs/generator.log
   logs/processor.log
   logs/logger.log

   Logs Contain:

   - Startup events
   - Processing status
   - Errors and warnings
   - Data transfer conformations

9. **Testing Strategy**

   The project includes:
   Unit Tests:

   - Test low-level utilities such as:
     - Image serialization
     - IPC message handling

End-to-End tests

- Validates the full pipeline:
  Generator → Processor → Logger

This ensures:

- Data integrity
- IPC correctness
- System reliability

10. Reliability & Fault Tolerance

- All applications can:
  - Start in any order.
  - Reconnect automatically
  - Continue running independently
- If one app crashes:
  - The others remain active
  - No System-wide failure occurs

11. Scalability & Future Enhancements

This architecture easily supports:

- Multiple processors running in parallel.
- New feature extraction algorithms
- Cloud-based deployment
- Web-based monitoring dashboards
- Distributed database storage
- Real hardware camera integration

12. Summary

The Distributed Image System is a robust, modular, and production-style distributed application built using modern software engineering practices.
It demonstrates:

- Distributed systems design
- IPC-based communication
- Image processing with OpenCV and SIFT
- Persistent data storage with SQLite

- Logging, testing and automation
- Configuration-driven execution