TED UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

High Level Design Report

CMPE 491 – Senior Design Project I

by

Berk Kaya

İlhan Ün

İrem Ayça Uçankale

Alperen Aktaş

Onur Turan

1. Introduction

1.1 Purpose of the system

The primary purpose of the Figion project is to automate and modernize the quality control process for dried figs, specifically targeting the detection of aflatoxin contamination. Currently, this process relies on manual inspection under UV light in dark rooms, which is subjective, prone to human error due to fatigue, and inefficient for high-volume production lines.

Figion aims to replace this manual workflow with an AI-driven, computer vision solution that operates in real-time on the production line. By utilizing deep learning algorithms to analyze fluorescence in UV-illuminated images, the system provides an objective, fast, and non-destructive screening method. The system is designed to serve quality assurance managers and production line operators by ensuring food safety standards are met while creating a verifiable digital audit trail of the inspection process.

1.2 Design goals

The design of the Figion system is driven by strict industrial and safety requirements. The key design goals are defined as follows:

Real-Time Performance (Latency): The system must match the throughput of manual processing pipelines. The total processing time—including image capture, inference, and result display—must not exceed 1 second per fig, allowing for a processing speed comparable to approximately 175 kg/hour.

High Reliability (Recall Priority): Given the public health implications of aflatoxin, the system prioritizes minimizing false negatives. The design goal is to achieve a Recall rate of at least 95% for the "Aflatoxin" class, ensuring contaminated products do not pass through the screening.

Portability and Edge Computing: The system is designed to operate independently of cloud resources to ensure stability and speed. It must run effectively on standard laptop CPUs or integrated GPUs, utilizing edge computing principles to avoid the need for expensive, specialized hardware.

Data Integrity and Traceability: Unlike the current transient manual inspection, the system must act as a traceability hub. It is designed to automatically archive the UV image of every scanned product alongside its classification decision in a structured format (e.g., CSV), enabling retrospective analysis and model retraining.

Usability: The user interface must be intuitive enough for operators to learn the scanning process within a short training period (under 30 minutes).

1.3 Definitions, acronyms, and abbreviations

Aflatoxin: A family of toxic and carcinogenic compounds produced by certain fungi found on agricultural crops such as dried figs. In the context of this project, it refers to the target contamination that exhibits fluorescence under UV light, which the system aims to detect without chemical analysis.

UV (Ultraviolet): Electromagnetic radiation with a wavelength shorter than that of visible light. The system utilizes high-intensity UV illumination in a dark environment to expose fluorescent contamination on the surface of dried figs, serving as the primary method for image-based detection.

HPLC (High-Performance Liquid Chromatography): An advanced, definitive laboratory analysis method used to separate, identify, and quantify components in a mixture. It serves as the "ground truth" for confirming aflatoxin presence but is a destructive and slow process applied only to random samples, unlike the project's non-destructive total scanning approach.

Recall and Precision: The two critical performance metrics for the AI model:

- **Recall (Sensitivity):** Measures the percentage of actual aflatoxin-contaminated figs correctly identified by the system. A minimum of 95% is required to meet public health obligations.
- **Precision:** Measures the percentage of figs labeled as "contaminated" that are actually contaminated. A minimum of 85% is targeted to prevent the financial loss associated with discarding healthy products (false positives).

Session and Batch:

- **Session:** A specific period of scanning activity initiated and ended by the operator. Starting a new session resets counters and creates a new directory for archiving images and logs.
- **Batch (Batch ID):** A unique identifier assigned to a specific group or lot of dried figs being processed. This ID is used to tag exported CSV reports and ensure traceability of the product line.

Inference: The process where the pre-trained Deep Learning model analyzes live image data to make a classification decision ("Healthy" or "Contaminated"). In this project, inference is performed in real-time on standard edge hardware (Laptop CPU/GPU) without relying on cloud services.

Latency: The time delay between capturing an image of a fig and displaying the classification result on the interface. To match the manual processing speed of approximately 175 kg/hour, the system requires a latency of less than 1 second per fig.

1.4 Overview

The Figion system is a solution designed to operate in an industrial environment, integrating hardware and software. The system hardware consists of a cabinet isolated from external light (dark room), a conveyor belt that transports products, a 365nm UV LED array that illuminates the figs, and a high-resolution USB camera that captures images. The software component is a desktop application that manages this hardware, processes the images, and provides user interaction.

The software architecture is designed as a hybrid application of Pipeline and Event-Driven architectural patterns. The image processing workflow (Capture -> Pre-processing -> Inference -> Post-processing -> Recording) operates with a pipeline logic that follows a strict sequence; meanwhile, the user interface, hardware status monitoring, and reporting tasks are handled in an event-driven manner.

The technology stack selection was determined by the project constraints (standard laptop hardware, high speed requirement). For object detection, the YOLOv11n (Nano) model was chosen, which offers higher efficiency and speed on the CPU compared to previous generations. For the user interface, a native desktop application framework was selected because Python-based web frameworks (such as Streamlit) cause latency and redraw issues in video streaming, whereas this approach enables the creation of hardware-accelerated, high-performance desktop interfaces. For the data storage layer, the SQLite database will be used, which does not require server setup, is file-based, but offers high reliability (ACID). Concurrency management is designed using the Producer-Consumer pattern to maintain the fluidity of the interface and prevent data loss, with camera and artificial intelligence operations running on separate threads.

2. Proposed software architecture

2.1 Overview

The software architecture of the Figion system is based on the Layered Architecture principle to manage complexity, ensure independent development of components, and offer ease of maintenance. The system is divided into three main logical layers according to their functional responsibilities:

- **Presentation Layer (View):** The point where the user interacts with the system. Displaying live video feed, visualizing detection results (class labels, confidence scores), reporting system status (ready, scanning, error), and receiving operator commands (Start/Stop) are the responsibilities of this layer.
- **Application Logic Layer (Controller/Service):** Functions as the brain of the system. Management of the image processing pipeline, execution of the AI model, making algorithmic decisions (threshold control, etc.), and coordination of hardware components (camera, relay) take place here. This layer acts as a bridge between the Presentation Layer and the Data Layer.
- **Data and Infrastructure Layer (Model):** Manages the system's persistent data storage operations and low-level hardware communication. Writing image files to disk, saving metadata to the SQLite database, creating CSV reports, and communication with cameras/relays at the driver level are provided in this layer.

Communication between these layers, especially between Presentation and Application layers, is designed in an asynchronous structure using Qt's Signal & Slot mechanism. In this way, while AI operations requiring intensive processing power are carried out in the background, the user interface can continue to work without freezing (non-blocking).

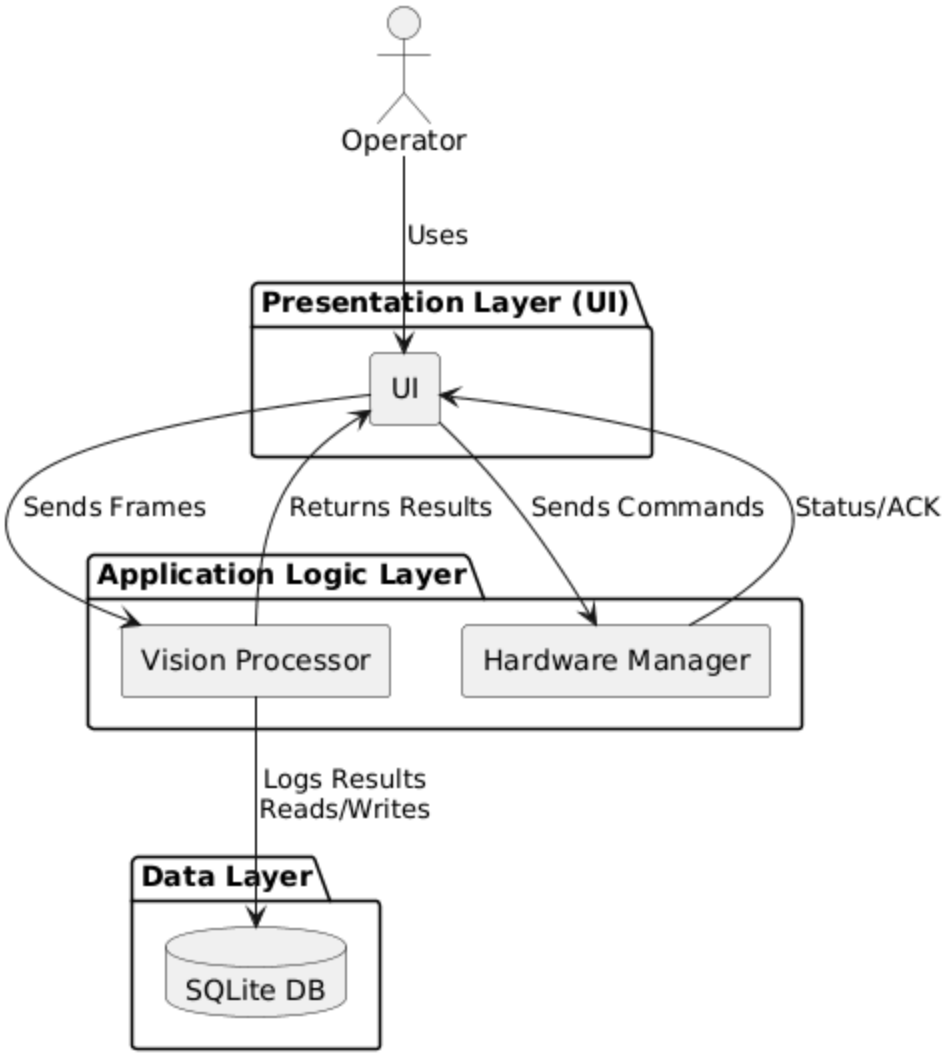

Figure 1: High-Level System Architecture Diagram illustrating the interaction between the User, UI, Logic, and Data layers.

2.2 Subsystem decomposition

The system is divided into four main subsystems to ensure functional integrity and isolate complexity: Vision Subsystem, Control Subsystem, Data Management Subsystem, and User Interface Subsystem.

2.2.1 Vision Subsystem

This subsystem, the most critical component of Figion, is responsible for collecting and analyzing visual data.

Image Acquisition Module: Connection with the USB camera is established using the cv2.VideoCapture class of the OpenCV library. This module pulls raw frames from the camera. It allows programmatic adjustment of camera parameters such as exposure, gain, and white balance so that fluorescent glow can be clearly seen in the dark room environment. Raw frames are forwarded to a Queue structure for processing.

Preprocessing Module: Converts the captured image into the format expected by the AI model. These operations include:

- **Resizing:** Reducing the image to 640x640 pixels (or according to the selected model size), which is the model's input size.
- **Color Space Conversion:** Conversion from OpenCV's default BGR format to the RGB format the model was trained on.
- **Normalization:** Scaling pixel intensity values from the [0, 255] range to [0, 1].

Inference Engine: This module hosts the trained YOLOv11n model. The model converted from .pt (PyTorch) format to ONNX (Open Neural Network Exchange) format, which runs faster in production environments, is used. Optimized inference on CPU is performed using the ONNX Runtime library. Research shows that YOLOv11 runs 20-30% faster on CPU compared to YOLOv8 and offers similar or higher accuracy (mAP) with fewer parameters. This is a critical advantage for laptop hardware constraints.

Post-processing Module: Converts raw tensor outputs returning from the model into meaningful data.

- **Non-Maximum Suppression (NMS):** Eliminates multiple overlapping boxes generated for the same object.
- **Thresholding:** Filters predictions below the determined confidence score threshold to meet the 95% Recall target specified in the Specifications Report.

2.2.2 Control Subsystem

Responsible for the management of physical hardware and the general status of the system.

Hardware Controller: Provides communication with the relay card controlling UV LED light sources. Generally, relay cards emulating Serial Port (COM) over USB (e.g., with CH340 chip) will be used. Using Python pyserial or pyhid-usb-relay libraries, signals are sent to turn on lights when scanning starts (ON command) and turn them off when finished (OFF command).

This module also checks whether hardware (Camera, Relay) is connected at system startup (Health Check).

State Manager: A Finite State Machine (FSM) structure managing the current state of the system (e.g., Initializing, Ready, Scanning, Paused, Error). It controls state transitions based on triggers from the user interface (Button press) and prevents invalid operations (e.g., starting scanning when there is no camera).

2.2.3 Data Management Subsystem

Manages the permanent storage of analysis results and images.

Session Manager: Creates a unique "Batch ID" for each new scanning operation (e.g., BATCH_20231215_001). Manages and resets session-based counters (Total, Defective, Clean).

Image Archiver: Ensures images are saved to disk. Since I/O operations are slow, this module runs in a separate thread (Worker Thread). Images are saved according to the determined folder structure (./data/images/YYYY-MM-DD/Batch_ID/) and naming convention (Fig_ID_Result.jpg).

Database Handler: The DAO (Data Access Object) layer managing interaction with the SQLite database. It writes data generated for each fig (Time, ID, Result, Confidence Score, File Path) to the database as an atomic transaction. It also hosts functions that convert records in the database to CSV format upon request (Export button).

2.2.4 UI Subsystem

The graphical interface that allows the operator to monitor and control the system.

Video Viewer: Displays images received from the camera with model predictions (bounding box, label) drawn on them in real-time. High-performance rendering is provided using QLabel or QGraphicsView widgets.

Control Panel: Contains Start/Stop button, settings menu, and status indicators.

Statistics Panel: Shows instantly updated counters (Total, Aflatoxin-infected, Clean, Ratio).

Session Log: Shows the list of recently scanned products in a sliding window using QTableWidget.

2.3 Hardware/software mapping

The Figion system is designed to run on standard computer hardware without requiring expensive servers or cloud connections. The system creates a bridge between physical devices (like the camera and lights) and the software components running on the user's laptop. This mapping ensures that the hardware resources are used efficiently to meet the speed requirements.

As shown in Figure 2, the system consists of three main hardware elements connected to the central computer:

Laptop / PC: This is the heart of the system. It runs the Figion Application, processes the AI model using the CPU, and displays the user interface.

USB Camera: It captures high-resolution images of the figs under UV light and sends them to the software via a USB connection.

UV Lighting & Relay: The software sends simple "On/Off" commands to a USB Relay, which controls the electrical power of the UV lights in the dark box.

To ensure the application runs smoothly and does not freeze, we map different software tasks to different computing resources. The User Interface (UI) runs on the main processor thread to stay responsive to clicks. The AI Analysis (YOLO model) runs on a separate background thread (worker) so it does not stop the video flow. Finally, saving images to the disk is handled separately to prevent delays during saving.

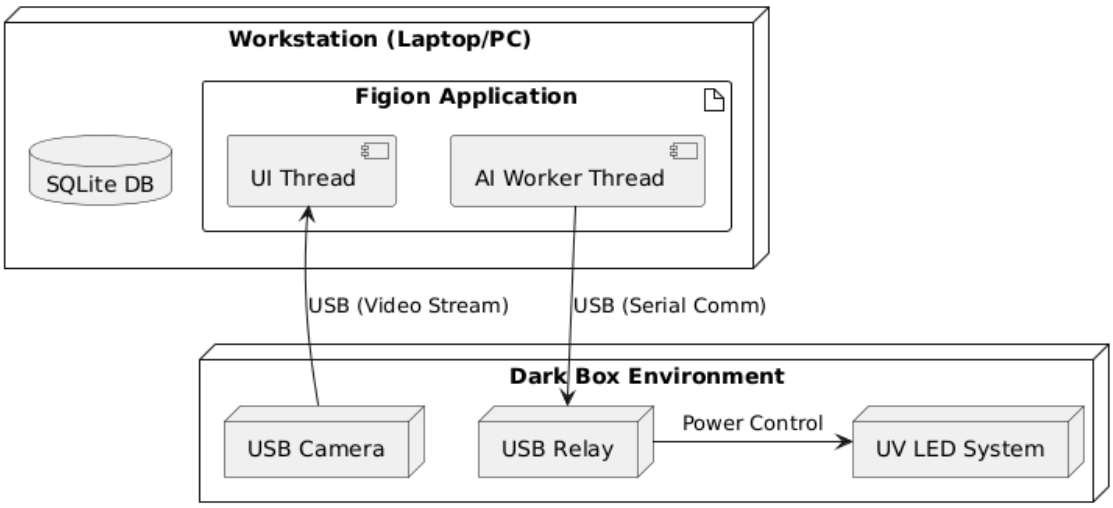

Figure 2: Deployment Diagram showing the mapping of software components to hardware nodes.

2.4 Persistent data management

The system needs to store two types of data permanently: the inspection results (text data) and the actual pictures of the figs (image data). This ensures that factory managers can review past production batches and generate quality reports.

2.4.1 Database System

We use SQLite as our database solution. It is a simple, file-based database that does not require installing a separate server application. It is very reliable and protects data even if the power goes out suddenly.

As illustrated in Figure 3, the database contains two main tables:

- **Sessions Table:** Stores general information about a work batch, such as when it started, when it ended, and the total number of figs processed.
- **Inspections Table:** Stores the detailed result for every single fig. This includes the timestamp, the decision (Healthy vs. Aflatoxin), the confidence score of the AI, and the link to the saved image file.

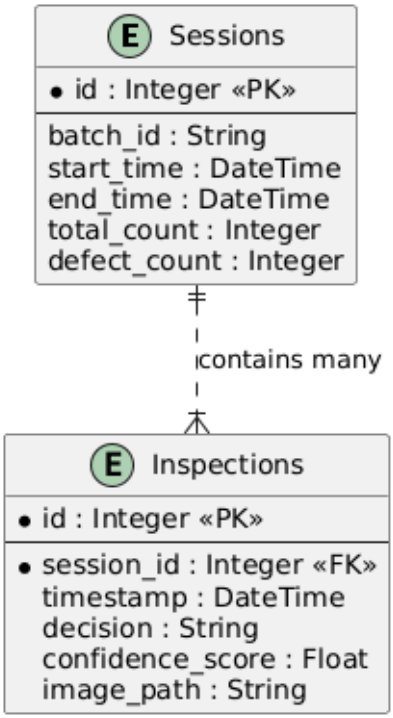

Figure 3: Entity-Relationship Diagram (ERD) representing the database schema for Sessions and Inspections.

2.4.2 Image Storage Since storing thousands of images directly inside a database can slow down the system, we save the actual image files directly to the computer's hard drive (File System). The database only stores the "address" (file path) of these images.

To keep files organized and easy to find manually, the system automatically creates folders based on the date and the batch ID. A typical folder structure looks like this:

- `Data/`
    - `Images/`
        - `2024-05-20/` (Date)
            - `Batch_001/` (Session ID)
                - `Fig_101_Healthy.jpg`
                - `Fig_102_Aflatoxin.jpg`

This structure allows users to easily copy or back up specific days or batches without needing special software.

2.5 Access control and security

Although the system will work in a physically secure area (inside the factory), software security measures will be taken to prevent data integrity and misuse.

Authentication: A simple login screen will greet at application startup. Users (Operator, Supervisor) will log in with a PIN or password defined for them.

Role-Based Access Control (RBAC):

- **Operator:** Can only perform "Start/Stop Scanning", "Reset", and "Get Report" operations. Cannot change settings or delete past records.
- **Supervisor:** Has all authorities. Can change camera settings, model confidence thresholds, delete old data.

Data Security: Passwords will never be stored as plain-text in the database; they will be kept hashed with secure algorithms like PBKDF2 or SHA-256. CSV reports and database files will be presented in "read-only" mode by the application or protected with file system permissions to prevent accidental modification or deletion by the operator.

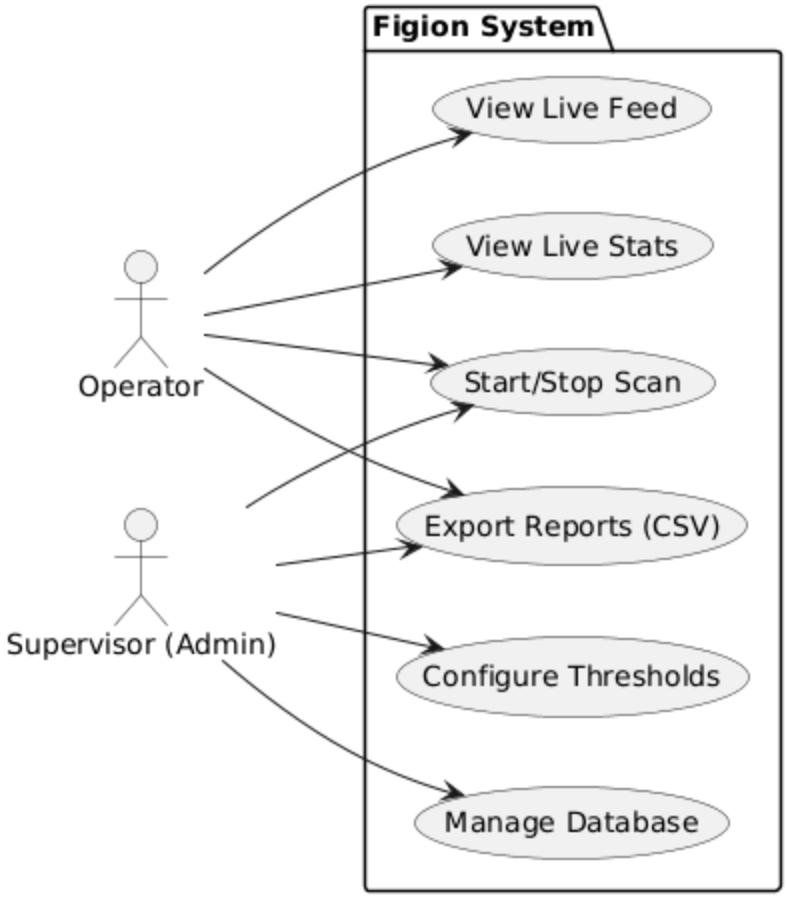

Figure 4: Use Case Diagram defining user roles (Operator vs. Supervisor) and their access privileges.

2.6 Global software control

The application's lifecycle and workflow are controlled by an event-driven state machine.

States and Transitions:

Boot: Application opens. Configuration files are read. Database connection is checked.

HW Check: Pings are sent to Camera and Relay card.

- **Success:** Transitions to **Idle** state.
- **Failure:** Transitions to **Error** state and "Camera Not Found" warning is shown to user.

Idle: System waits for command from user. Camera preview is active but recording/inference is not performed. UV lights are off (or in standby mode).

Scanning: Transitioned when "Start" button is pressed.

- **Action:** UV Lights are turned ON. Inference Thread is activated. Images start being recorded.
- **Paused:** Transitioned when "Pause" button is pressed.
- **Action:** Inference is stopped. UV lights are (optionally) turned off.
- **Shutdown:** Triggered when application is closed.
- **Action:** All hardware connections are released (Camera Release), UV lights are forcibly turned OFF, database connection is securely closed.

Exception Handling: To prevent the system from crashing in unexpected situations (e.g., camera cable being pulled during operation), a global try-catch (try-except in Python) block will wrap the main loop. In case of an error, the system will transition to a safe state (UV lights off), write the error to an error.log file, and show an understandable message to the user.

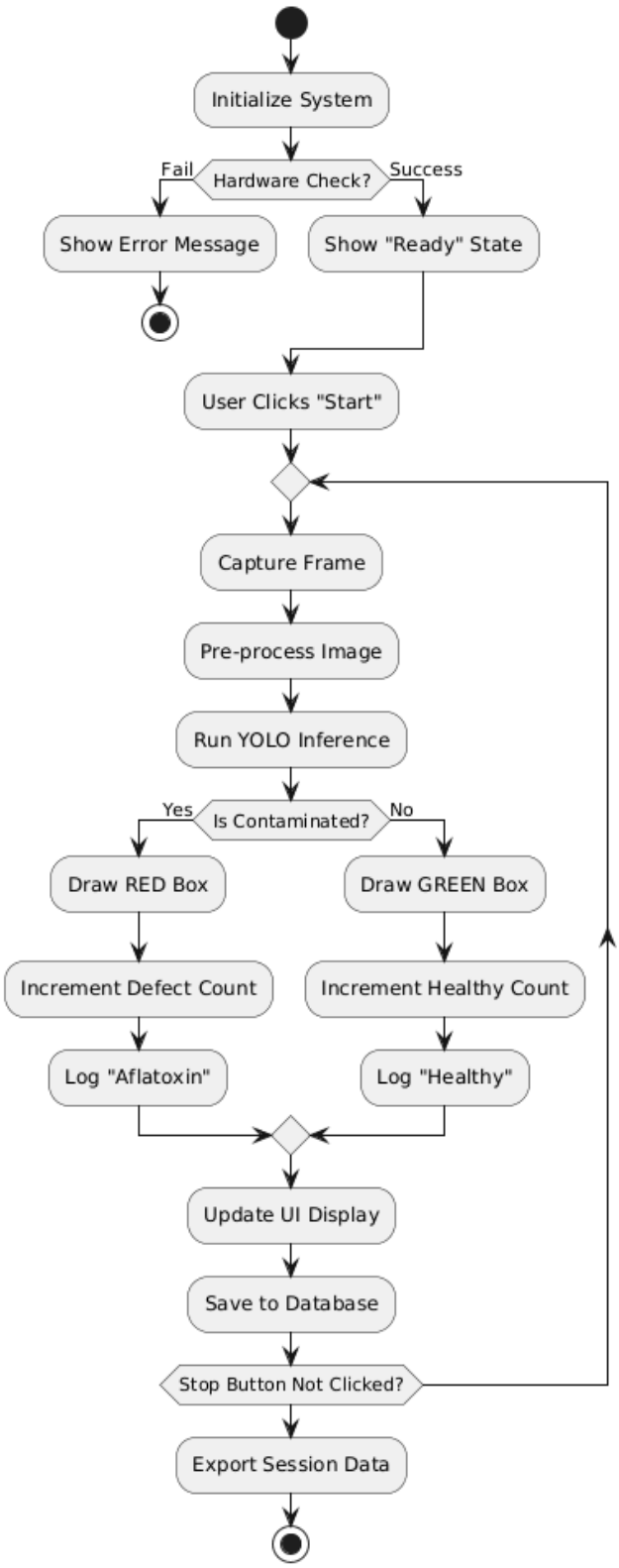

Figure 5: Activity Diagram illustrating the global control flow and decision logic of the inspection process.

2.7 Boundary conditions

To ensure system stability, edge cases will be managed as follows:

Cameraless Startup: If the system cannot find the camera at startup, it will continue to start but will warn the user by disabling the "Start" button.

Disk Full: If the disk space falls below a critical level during image recording, the system will stop image recording (only keeping a text log) and give the operator an audible/visual warning.

Excessive Conveyor Speed: If the conveyor belt flows faster than the model's processing speed (queue filling up), the system will prioritize processing the current frame by dropping the oldest frame (Frame Dropping). This prevents latency from accumulating and causing the live image to lag behind.

No Detection: Since there will be no fluorescent light in "clean" figs, "no object detection" is an expected situation. If there is no fluorescence in the image despite the presence of figs, the system should record this as "Healthy" and not as "Error". For this, the model may need to recognize not only the "Aflatoxin" stain but also the "Fig" itself, or the presence of the fig may need to be confirmed through image processing (blob detection).

3. Subsystem services

3.1 Vision Services

initialize_camera(camera_id): Loads the camera driver and sets the parameters (exposure, size).
capture_frame(): Returns a snapshot from the camera.
run_inference(frame): Retrieves an image frame, passes it through the YOLO model, and returns a list of detected objects ([{class: 'Aflatoxin', confidence: 0.98, bbox: [...]}]).
release_resources(): Safely disconnects the camera connection.

3.2 Control Services

connect_relay(port): Opens a serial connection to the relay board.
toggle_uv_light(state: bool): Turns the lights on with the True parameter and off with False.
get_hardware_status(): Queryes the health status of the hardware (Connected/Disconnected).

3.3 Data Services

start_new_session(user_id): Creates a new session record and returns session_id.
save_inspection_record(data_object): Writes a single analysis result to the database.
queue_image_save(image, metadata): Sends the image to the background thread to write it to disk.
export_data(session_id, format='csv'): Exports the data of the specified session as a CSV file.
get_live_statistics(): Queries current statistics (Total, Errors, %) for the counters in the interface.

3.4 Interface Services (UI Service)

update_video_feed(qimage): Draws the processed image to the video component on the screen.
refresh_counters(stats_dict): Updates the statistics widgets.
display_error(message): Opens an error message window for the user.
on_start_stop_clicked(): Captures the user's start/stop action and passes it to the Control Service.

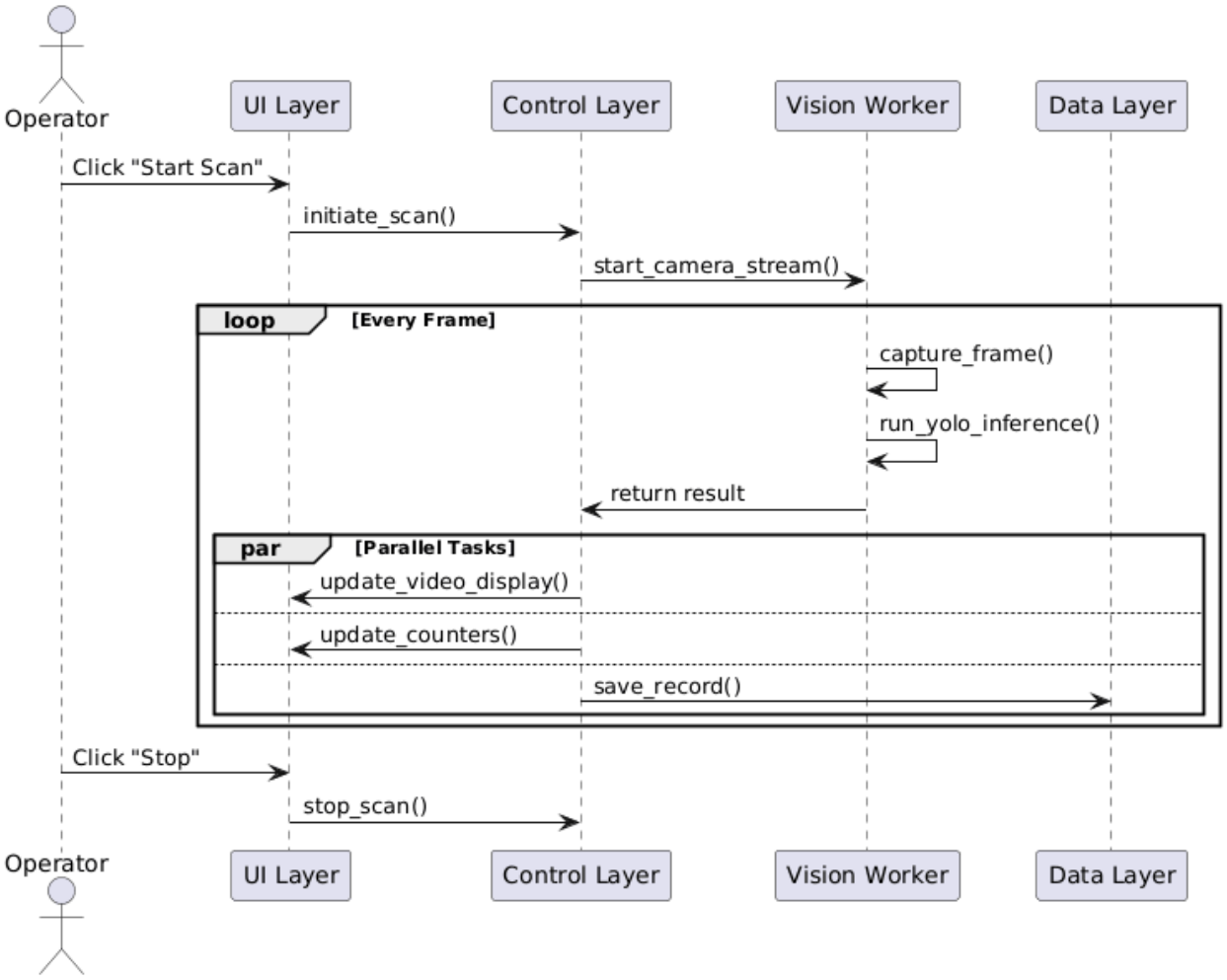

Figure 6: Sequence Diagram detailing the interactions between subsystems during a scanning session.

4. Glossary

Aflatoxin: A family of toxins produced by certain fungi that are found on agricultural crops such as maize (corn), peanuts, cottonseed, and tree nuts.

HPLC : An advanced laboratory analysis method used to separate, identify, and quantify the components in a mixture. A liquid sample is forced through a column filled with very fine particles at high pressure. Each substance passes through the column at different speeds, allowing for separation.

5. References

1. Kuru incirlerde AFLATOKSİN ve okratoksin a bulaşisinin önlenmesi. (n.d.). https://www.tarimorman.gov.tr/GKGM/Belgeler/Uretici_Bilgi_Kosesi/Egitim/Hijyen_Kilavuz/kuru_incir_aflatoksin_okratoksin_a_onleme_azaltma.pdf

2. Ömer Barış Özlüoymak. (2014). Development of an UV-Based Imaging System for Real-Time Aflatoxin Contaminated Dried Fig Detection and Separation. Tarım Bilimleri Dergisi/Ankara Üniversitesi Ziraat Fakültesi Tarım Bilimleri Dergisi, 20(3), 302–302. https://doi.org/10.15832/tbd.87873

3. Kılıç, C., Özer, H., & İnner, B. (2024). Real-time detection of aflatoxin-contaminated dried figs using lights of different wavelengths by feature extraction with deep learning. Food Control, 156, 110150. https://doi.org/10.1016/j.foodcont.2023.11015

4. Ultralytics. (n.d.). YOLO11 vs. YOLOv8: Performance Comparison and Benchmarks. Ultralytics Documentation. https://docs.ultralytics.com/compare/yolo11-vs-yolov8/