TED UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Final Report

CMPE 492 – Senior Design Project II

by

Berk Kaya

İlhan Ün

İrem Ayça Uçankale

Alperen Aktaş

Onur Turan

1. Introduction

The Figion project addresses the need for food safety in the dried fig export industry. The primary goal of this project is to automate the quality control process for dried figs, specifically finding aflatoxin contamination. Our project aims to create a rapid, low-cost, and automatic system using image analysis under UV light. The system is built for quality assurance managers and production line operators to make sure food safety standards are met. Overall, the system aims to make the control processes faster, more reliable, and more traceable.

2. Problem Definition and Motivation

Aflatoxins are very toxic and carcinogenic compounds created by mold, and they are a big risk for food safety. Dried figs are a very important export product for Türkiye. However, if aflatoxin is found at customs, the fig batches are rejected, which causes big economic losses and damages the country's international reputation.

The motivation for this project comes from the problems of current inspection methods. Right now, the quality control process is completely manual. Workers look at figs under UV light in dark rooms to find fluorescent spots. This manual method is very slow, subjective, and has a high error rate because workers get eye strain and fatigue. On the other hand, the definitive chemical analysis method (HPLC) is accurate, but it destroys the fig and cannot be used for the whole production line. These problems motivated us to develop a fast, real-time, and non-destructive screening solution from scratch.

3. Proposed System

To solve the problems of manual inspection, the Figion project proposes a fully automated, AI-driven computer vision solution. The proposed system integrates both hardware and software to work in an industrial environment.

The hardware part consists of a dark cabinet isolated from external light, a 365nm UV LED array to make the aflatoxin visible, and a high-resolution USB camera to capture images. The software is a desktop application developed with Python and the PyQt6 framework. For the artificial intelligence part, the system uses the YOLOv11n (Nano) object detection model. We

optimized this model in ONNX format to run very fast on a standard laptop CPU. This allows the system to process each fig in less than 1 second to match the manual production speed of 175 kg/hour.

When a fig is scanned, the operator sees instant results on the dashboard. The AI model analyzes the image and draws a green box for healthy figs or a red box for figs with aflatoxin. Finally, for traceability, the system archives the images and writes the inspection results to a local SQLite database, allowing users to export CSV reports based on date and batch.

4. Final Architecture and Design

4.1 Overall Architectural Approach

The final architecture of the Figion system was designed as a modular desktop application prototype for real-time aflatoxin risk detection in dried figs. The main purpose of the architecture is to transform the traditional manual UV inspection process into a faster, more objective, traceable, and AI-supported quality control workflow. For this reason, the system combines computer vision, artificial intelligence inference, local data storage, image archiving, and a desktop-based user interface within a structured software design.

The overall software design follows a layered architecture. This architectural decision was made to separate the responsibilities of the system and to make the project easier to develop, test, and maintain. The main layers are the Presentation Layer, Application Logic Layer, and Data and Infrastructure Layer. The Presentation Layer is responsible for user interaction, live video display, detection visualization, system status messages, and Start/Stop commands. The Application Logic Layer coordinates the image-processing pipeline, AI inference process, session control, and decision-making logic. The Data and Infrastructure Layer manages persistent storage, image archiving, SQLite database operations, CSV report generation, configuration management, logging, and file path creation.

In addition to the layered structure, the final software design follows a pipeline-based workflow for image processing. Each captured frame passes through a clear sequence: camera capture, AI inference, result visualization, image archiving, database recording, and optional CSV export. This pipeline structure is important because every inspection result should be processed consistently and saved in a traceable format.

At the same time, the system uses an event-driven structure for user actions and interface updates. For example, when the operator clicks the “Start Scan” button, the scanning process begins. When the operator clicks the “Stop Scan” button, the scanning process is paused or stopped. When a detection result is produced, the interface is updated with visual feedback.

Therefore, the architecture combines a pipeline-based image-processing flow with event-driven user interaction.

4.2 Software Subsystems

The final design is divided into several main software modules: vision, data, control, ui, utils, and models. This modular structure follows the subsystem-based design approach of the project and makes the application more readable, maintainable, and extendable.

The vision module is responsible for camera access and AI inference. It includes camera_manager.py, which works as an OpenCV-based camera wrapper, and inference_engine.py, which manages the YOLOv11n ONNX inference process. This module receives frames from the USB camera and processes them through the AI model. The output of this module is the detection result, which indicates whether the fig is healthy or has possible aflatoxin contamination.

The data module is responsible for database operations, session handling, inspection records, and image archiving. It includes database_handler.py, session_dao.py, inspection_repository.py, image_archiver.py, and session_manager.py. These components support the traceability goal of the system. Inspection results, session information, image paths, and related metadata can be saved and later exported as CSV reports.

The control module contains hardware_monitor.py and state_manager.py. In the current prototype, this module mainly handles hardware health checking and application state management. The state_manager.py file represents the finite-state-machine logic of the system, such as Ready, Scanning, and Error states. This is important because the application must behave predictably during normal operation and possible error cases. In the current README-based implementation, the UV light is not controlled directly by the software; it is manually turned on by the user before scanning. Therefore, the control layer should be understood as a state and hardware-monitoring layer in the current prototype, while automatic UV relay control can be added in future versions.

The ui module contains the PyQt6-based desktop user interface. The main window of the application is implemented in main_window.py. The file video_processor_worker.py is used as a QThread-based worker for camera and AI-related operations. This is an important design decision because camera reading and inference operations should not freeze the graphical user interface. The widgets.py file contains reusable interface components, and styles.py contains the Qt style sheet for the dark theme.

The utils module contains shared helper classes and functions. It includes config_manager.py for reading config.ini, logger.py for file and console logging, path_builder.py for creating file

paths, and dto.py for data transfer objects. These utility files reduce code repetition and keep the main modules cleaner.

The models folder is reserved for the trained ONNX model file. According to the README structure, the YOLOv11n ONNX model should be placed as models/figion_yolo.onnx. This makes the model location clear and separates the trained AI model from the source code.

4.3 Hardware and Software Integration

The current Figion prototype is designed to work with a USB camera or webcam and a manually operated UV light source. The user manually turns on the UV light before starting the scan. Then, the software captures images from the camera and analyzes them using the YOLOv11n ONNX model. This design keeps the prototype simple and runnable on standard computers without requiring full hardware automation.

The README also mentions that dried figs are placed on the conveyor belt or inspection area before scanning. However, in the current software structure, the conveyor belt is not directly controlled by the application. Therefore, the software focuses on camera-based image capture, AI-based detection, interface visualization, and data recording. Conveyor movement and UV light activation are treated as external or manual parts of the current prototype workflow.

The hardware-software workflow can be summarized as follows: first, the user turns on the UV light manually and positions the dried figs under the camera. Then, the user starts the application and clicks the scan button. The USB camera captures frames, and these frames are sent to the inference engine. The AI model analyzes the image and produces a result. The result is displayed on the interface with a visual indicator, and the related data is saved for traceability.

This level of integration is suitable for the current prototype stage because it allows the team to test the most important part of the project: UV image-based aflatoxin detection using AI. In future work, the same architecture can be extended with USB relay support for automatic UV light control and more advanced conveyor synchronization.

4.4 Data Management and Traceability Design

Traceability is one of the most important design goals of Figion. In the traditional manual inspection process, inspection decisions are often temporary and not digitally recorded. Figion improves this process by saving inspection results, image records, timestamps, and session-related information.

The current project structure includes a data module for database and session logic. This module contains the main components responsible for SQLite database connection, session creation, inspection record storage, image archiving, and session coordination. In addition to this code-level data module, the application also creates runtime data files such as the SQLite database, archived images, logs, and exported CSV reports.

The SQLite database is used to store structured inspection metadata. This may include information such as session ID, Fig ID, decision result, confidence score, timestamp, and image path. Image files are saved separately instead of being stored directly inside the database. This design is important because saving large image files directly in the database could slow down the system and make the database harder to manage.

The application also supports CSV export. After scanning is completed, the user can export the inspection results as a CSV report. This report can be used for quality-control documentation, later review, and project demonstration. CSV export also supports the audit-trail goal of the system because inspection results become easier to share and examine.

The config.ini file stores important runtime settings. According to the README, these settings include the camera index, model confidence threshold, IoU threshold, and SQLite database path. For example, the default camera index is 0, the confidence threshold is 0.50, and the IoU threshold is 0.45. This design makes the system easier to configure without changing the source code directly.

4.5 Concurrency and Real-Time Processing Design

Real-time performance is an important requirement of the Figion system. During scanning, the application must capture frames, run AI inference, update the interface, and save records without making the user interface unresponsive. For this reason, the current design separates interface operations from camera and AI processing.

The video_processor_worker.py file in the ui module is responsible for running camera and AI-related processing in a separate QThread. This design helps prevent the PyQt6 interface from freezing while the system is reading frames from the camera or running inference through the YOLOv11n ONNX model.

This concurrency approach improves both performance and usability. While the AI model analyzes camera frames, the main interface can continue displaying the video feed, detection results, buttons, counters, and status information. This is important because the operator must be able to monitor the system in real time and stop the scan when necessary.

The image archiving process is also separated from the main user interaction logic. The image_archiver.py component is responsible for saving captured images, which supports the

traceability goal of the project. By organizing image saving as a separate responsibility, the system becomes easier to maintain and can be improved later for more advanced background processing.

4.6 Design Rationale

The final design choices were made according to the project's prototype goals, real-time detection needs, portability, and maintainability.

Python was selected as the main programming language because it has strong support for computer vision, AI model inference, GUI development, and database operations. It also allows faster development and easier integration with libraries such as OpenCV, ONNX Runtime, PyQt6, and SQLite.

YOLOv11n was selected as the object detection model because it is lightweight and suitable for real-time inference. Since the system is expected to work on standard computers, a smaller and faster model is more practical than a larger model that may require stronger hardware. The model is used in ONNX format, and the expected file location is models/figion_yolo.onnx. This makes deployment easier and keeps the model separate from the application code.

OpenCV is used for camera access and image processing. This is suitable because the system needs to read frames from a USB camera or webcam. PyQt6/Qt is used for the desktop interface because the application requires live video display, buttons, status indicators, and real-time feedback. A desktop interface is also more suitable than a web interface for this prototype because it can interact more directly with local camera resources.

SQLite is used for local data storage because it is lightweight and does not require a separate database server. This fits the edge-computing logic of the project. The application can run locally on a standard computer and save inspection data without depending on an internet connection or external cloud service.

The modular folder structure also supports maintainability. For example, the camera code is separated from the inference engine, the database code is separated from the UI, and shared utilities are placed in a separate utils folder. This makes it easier to debug, test, and improve the project in the future.

Overall, the final architecture and design of Figion provide a practical and runnable desktop application prototype. The system supports UV image-based inspection, YOLOv11n ONNX inference, OpenCV camera input, PyQt6 user interaction, SQLite-based metadata storage, image archiving, logging, configuration management, and CSV export. Although UV light control and conveyor control are manual or external in the current version, the architecture is

modular enough to support future hardware automation. This makes the project suitable as a strong prototype foundation for an AI-based aflatoxin pre-screening system.

5. Final Status of the Project

5.1 Current Implementation Status

At the final stage of the project, Figion has been transformed from a conceptual design into a structured desktop application prototype. The current version of the project is organized as a Python-based application that analyzes UV camera images of dried figs by using a YOLOv11n model in ONNX format. The system is designed to support real-time aflatoxin risk detection through a modular software structure, local data storage, image archiving, and CSV reporting.

The current implementation focuses on the main operational workflow of the system. A user can run the application, activate the UV light source manually, place dried figs under the camera or on the conveyor area, start the scanning process from the interface, observe the detection results visually, stop the scan, and export the inspection results as a CSV report. In this prototype version, the UV light source is not controlled directly by the software; instead, it is manually turned on by the user before scanning. However, the software structure still includes a control layer for hardware monitoring and system state management, which allows future relay-based UV control to be integrated more easily.

The project is currently arranged around a clear folder structure. The main entry point of the application is main.py, while the general configuration values are stored in config.ini. Dependencies are listed in requirements.txt, and setup/run scripts are provided for Windows, macOS, and Linux environments. This makes the project easier to install and run on different operating systems.

5.2 Implemented Software Modules

The current project structure follows the subsystem design that was planned in the earlier design documents. The application is divided into several main modules: vision, data, control, ui, utils, and models. This modular structure improves readability, maintainability, and future extensibility.

The vision module is responsible for camera access and AI inference. It includes camera_manager.py, which works as an OpenCV-based camera wrapper, and inference_engine.py, which manages the YOLOv11n ONNX model. This part of the system receives frames from the USB camera and processes them through the AI model to detect whether a fig is healthy or potentially contaminated with aflatoxin.

The data module manages database operations, session handling, inspection records, and image archiving. It includes components such as database_handler.py, session_dao.py, inspection_repository.py, image_archiver.py, and session_manager.py. These files support the traceability goal of the system by saving inspection information, creating session records, and archiving captured images.

The control module contains hardware_monitor.py and state_manager.py. In the current prototype, this module is mainly used for hardware health checks and application state control. The state_manager.py file represents the finite-state-machine logic of the system, such as Ready, Scanning, and Error states. This is important because the system must behave predictably during normal scanning and error situations.

The ui module contains the PyQt6-based user interface. The main interface is handled by main_window.py, while video_processor_worker.py runs camera and AI-related processing in a separate QThread. This design prevents the user interface from freezing during scanning. The module also includes reusable widgets and style definitions for the dark-themed Qt interface.

The utils module includes shared helper components such as configuration management, logging, path creation, and data transfer objects. These utilities make the codebase cleaner because common functions are separated from the main business logic.

Finally, the models folder is reserved for the ONNX model file, figion_yolo.onnx. This separation makes it clear where the trained detection model should be placed before running the application.

5.3 Current User Workflow

The current version of Figion supports a simple and understandable user workflow. First, the user installs the required dependencies by running the setup script or by manually creating a Python virtual environment and installing the packages from requirements.txt. Then, the user starts the application by running the appropriate script or executing python main.py.

After the application starts, the user manually turns on the UV light source and positions the dried figs under the camera. The scanning process is started by clicking the “Start Scan” button on the interface. During scanning, the system captures frames from the camera, sends them to the YOLOv11n ONNX inference engine, and displays the result visually on the interface. A red bounding box indicates that aflatoxin risk has been detected, while a green bounding box indicates a healthy fig.

When the scanning process is completed, the user can stop the scan and export the inspection results as a CSV file. This exported report can be used for documentation, later review, and

quality-control records. Therefore, even in the prototype version, the system supports the basic traceability requirement of the project.

5.4 Configuration and Runtime Data

The current application uses a config.ini file to manage important system settings. This allows the user or developer to change key parameters without editing the source code directly. For example, the camera index can be changed if the computer has more than one camera connected. The confidence threshold and IoU threshold of the model can also be adjusted according to detection needs.

The default camera index is set to 0, which usually represents the built-in camera or the first connected USB camera. If another camera is used, this value can be changed to 1, 2, or another available index. The model confidence threshold is set to 0.50, meaning that detections below this confidence level are ignored. The IoU threshold is set to 0.45, which is used during non-maximum suppression to remove overlapping detections.

The SQLite database path is also defined in the configuration file. The database stores inspection-related metadata, while images, logs, and exported reports are saved in the automatically generated runtime data folders. This structure supports the project's traceability goal by keeping images, database records, logs, and exported reports organized.

5.5 Installation and Deployment Status

The project is prepared to be installed and run on Windows, macOS, and Linux. For Windows users, setup.bat is provided for first-time setup, and run.bat is provided for launching the application later. For macOS and Linux users, setup.sh and run.sh scripts are provided for the same purpose.

Manual installation is also possible. The user can create a Python virtual environment, activate it, install the required dependencies from requirements.txt, and run the program with python main.py. This gives flexibility to both technical and non-technical users.

This installation structure shows that the project has reached a runnable prototype stage. It is not only described in reports but also organized as a working software project with a clear entry point, dependency file, configuration file, model folder, and runtime data structure.

5.6 Remaining Limitations

Although the current version provides a strong prototype foundation, some limitations still remain. The most important limitation is that the UV light is manually controlled in the current

version. This means that the user must turn on the UV light before starting the scan. In a more advanced version, the UV light can be controlled automatically through a USB relay, as described in the design documents.

Another limitation is related to the model file. The models folder expects a figion_yolo.onnx file to be placed manually. Therefore, the application depends on the availability of a trained and exported YOLOv11n ONNX model. If the model file is missing or not trained sufficiently, the detection system cannot provide reliable results.

The system also still requires more extensive testing with real dried fig samples. Camera quality, UV lighting strength, external light leakage, fig surface differences, and conveyor movement may affect detection results. Therefore, the prototype must be validated under more realistic production conditions.

Dataset limitations are also still important. A larger and more diverse dataset is needed to improve model reliability. Laboratory-confirmed labels, such as HPLC-based ground truth data, would also improve the scientific validity of the model evaluation. Without enough validated data, the system should be treated as a pre-screening and decision-support tool rather than a final replacement for laboratory analysis.

5.7 Overall Final Status

Overall, the final status of the Figion project is successful at the prototype and software-architecture level. The project has a clear Python-based desktop application structure, a defined installation process, a modular folder organization, a YOLOv11n ONNX inference pipeline, OpenCV camera support, PyQt6 user interface components, SQLite-based data storage, image archiving, logging, and CSV export functionality.

The current system demonstrates the main idea of the project: using UV camera images and AI-based inference to support aflatoxin pre-screening in dried figs. It also supports important engineering goals such as modularity, traceability, local processing, and user-friendly operation.

However, the system still needs further improvements before full industrial deployment. These improvements include automatic UV relay control, larger dataset collection, stronger model validation, real-environment testing, hardware durability improvements, and safer UV operation procedures. With these improvements, Figion can move from a working prototype toward a more complete and industrially applicable aflatoxin detection system.

6. Engineering Impact

The Figion project is not only a software and hardware design project; it is also an engineering solution that can create impact in food safety, agricultural exports, industrial quality control, sustainability, and public health. Since aflatoxin contamination is a serious issue for dried figs, especially in export processes, the proposed system has potential effects at global, economic, environmental, and societal levels.

6.1 Global Impact

Aflatoxin contamination is a global food safety concern because aflatoxins are toxic and carcinogenic compounds that may appear in agricultural products such as dried figs. In international food trade, especially for exported food products, safety standards are strict. If contaminated products are detected at customs or during laboratory inspections, the entire batch may be rejected. Therefore, the problem is not limited to a single producer or company; it directly affects international trade, consumer trust, and the reputation of exporting countries.

The Figion system contributes to this global issue by offering a faster and more accessible pre-screening method for dried figs. Traditional laboratory methods such as HPLC provide definitive analysis, but they are destructive, time-consuming, and usually applied only to samples rather than every product. Figion, on the other hand, is designed as a non-destructive, image-based screening system that analyzes figs under UV light before they move further in the production or export process. This makes it possible to detect risky products earlier and reduce the chance of contaminated figs reaching international markets.

From a global engineering perspective, Figion supports the trend of using artificial intelligence and edge computing in agriculture and food safety. Instead of depending on expensive cloud infrastructure or centralized laboratories for every inspection step, the system performs inference locally on standard computer hardware. This makes the solution more portable and easier to adapt to different production environments. The project's edge-computing approach is also consistent with the system goal of operating independently from cloud services while maintaining real-time performance.

In addition, Figion can improve the international competitiveness of dried fig exporters. Türkiye is an important dried fig producer and exporter, and the project specifically addresses the problem of export rejection caused by aflatoxin detection. The Project Specifications Report states that fig batches may be rejected at customs due to aflatoxin, causing economic and international reputation losses. By reducing this risk, the system can help producers meet global food safety expectations more consistently.

Overall, the global impact of Figion is its contribution to safer food supply chains, more reliable agricultural exports, and the adoption of AI-based quality control systems in food production.

6.2 Economic Impact

The economic impact of Figion is one of the most important aspects of the project because aflatoxin contamination directly affects producers, exporters, and the national economy. In the current workflow, dried figs are inspected manually under UV light, and definitive chemical analysis is applied only to samples. If contamination is discovered late in the process, companies may face rejected shipments, product loss, additional laboratory costs, delays, and damage to their export reputation.

Figion can reduce these losses by detecting potentially contaminated figs earlier in the quality-control chain. Early detection allows companies to separate risky figs before they are packaged, shipped, or submitted to customs control. This can decrease the probability of batch rejection and reduce the financial damage caused by late-stage contamination discovery.

The system also has potential to reduce labor-related costs. Manual inspection under UV light requires trained workers to visually examine figs in dark rooms. This process is repetitive, tiring, and subject to human error caused by fatigue. The Analysis Report explains that the current system relies completely on human inspection under UV light and is subjective, slow, and prone to errors due to eye strain and fatigue. By automating this task, Figion can help companies improve inspection consistency while allowing workers to focus on supervision, verification, and process management instead of continuous manual scanning.

Another economic contribution is increased production efficiency. The system is designed to meet the manual production-line speed of approximately 175 kg/hour, with a latency requirement of less than one second per fig. This is important because an engineering solution that improves accuracy but slows down production would not be practical in an industrial setting. Figion aims to provide both speed and reliability, which makes it more suitable for real production environments.

The project can also reduce the dependency on repeated laboratory testing. HPLC and similar chemical analyses are still necessary for definitive confirmation, but Figion can act as a pre-screening tool. This means that laboratory resources can be used more efficiently, focusing on suspicious or sample-confirmation cases instead of being the only reliable control point.

Furthermore, Figion creates economic value through traceability. The system stores inspection results, image records, timestamps, decisions, and batch-related information. This creates a digital audit trail that quality managers can use for reporting, customer assurance, and future

process improvement. The proposed system is described as a traceability hub that archives UV images and classification data for later analysis and model retraining. This can improve accountability and help companies demonstrate quality-control practices to buyers, auditors, and regulatory authorities.

6.3 Environmental Impact

The environmental impact of Figion is mainly connected to waste reduction, resource efficiency, and the energy footprint of the system.

Aflatoxin detection is traditionally confirmed by laboratory analysis, which is destructive and usually performed on selected samples. In contrast, Figion is designed as a non-destructive pre-screening system. It analyzes surface fluorescence under UV light and classifies figs without physically damaging them. This is environmentally beneficial because it can help reduce unnecessary product destruction during the early inspection phase.

Another environmental benefit is waste reduction. If contaminated figs are detected earlier, producers can separate risky products before they are mixed with larger batches. This can prevent the rejection or disposal of larger quantities of food. In food production, late-stage rejection often creates more waste because packaging, transportation, storage, and labor have already been used. By identifying risks earlier, Figion can reduce wasted materials and energy across the supply chain.

The system may also help reduce transportation-related environmental costs. If contaminated batches are detected only after export or customs inspection, products may need to be returned, destroyed, or replaced. These actions increase transportation emissions and resource consumption. A reliable local pre-screening system can help reduce such unnecessary logistics by improving quality control before shipment.

Figion also uses edge computing, meaning that the analysis is performed locally instead of continuously sending data to cloud servers. This can reduce dependency on network infrastructure and remote computation. While the system still consumes electricity through the camera, UV LEDs, computer, and possibly conveyor integration, local processing may be more practical and energy-efficient for a production-line setting than cloud-dependent alternatives.

However, the environmental impact is not entirely positive unless the system is designed and operated responsibly. UV LED arrays, cameras, computers, and electronic control components consume energy and eventually contribute to electronic waste. Therefore, the system should be designed with energy-efficient components, durable hardware, safe UV operation, and repairable/replaceable parts. The use of standard laptop CPUs or integrated GPUs instead of

expensive specialized hardware supports portability and may reduce the need for heavy industrial computing infrastructure.

6.4 Societal Impact

The societal impact of Figion is strongly related to public health, worker conditions, consumer trust, and the digital transformation of the food industry.

The most important societal contribution is improved food safety. Aflatoxins are dangerous compounds, and contaminated food products can create serious health risks. Because of this, the system prioritizes recall, meaning that it aims to minimize false negatives: contaminated figs should not be incorrectly classified as healthy. The design goal is to achieve at least 95% recall for the aflatoxin class. This priority shows that the system is designed with public health responsibility in mind, not only technical performance.

Figion can also increase consumer confidence. When food companies use traceable and data-driven inspection systems, they can provide stronger evidence that their products were checked systematically. This is especially important in export markets where buyers and consumers expect consistent quality and safety. A system that records images, decisions, timestamps, and batch information can support transparency in the food supply chain.

Another societal impact is the improvement of working conditions. Manual UV inspection requires workers to spend long periods in dark-room environments while visually checking figs for fluorescence. This can cause fatigue, inconsistency, and eye strain. Figion does not necessarily remove the need for human operators, but it changes their role from continuous manual inspection to system supervision, decision verification, and quality management. This can make the work less repetitive and more data-oriented.

The project also contributes to the digital transformation of agriculture and food production. By combining AI, computer vision, UV imaging, and local data management, Figion demonstrates how traditional agricultural quality-control processes can be modernized. This can encourage producers and students to adopt engineering-based solutions for real agricultural and industrial problems.

However, the societal impact also includes ethical responsibilities. Since the system makes decisions related to food safety, it must be tested carefully and transparently. A weak model, biased dataset, or poorly controlled lighting environment could create incorrect results. False negatives may harm public health, while false positives may cause unnecessary economic loss for producers. Therefore, the system should not be presented as a complete replacement for laboratory analysis. It should be described as a pre-screening and decision-support tool. Final confirmation may still require chemical analysis, especially in critical cases.

The project also has an educational and national-development impact. The Analysis Report states that Figion aligns with food security, agricultural technologies, and artificial intelligence application goals. This means the project contributes not only to a single company's workflow but also to broader goals such as improving agricultural technology, food safety, and AI-based industrial innovation.

7. Contemporary Issues Related to the Project

The Figion project is related to several contemporary issues in food safety, artificial intelligence, agriculture, and industrial automation. Since the project focuses on detecting aflatoxin in dried figs using UV imaging and AI, it connects directly with current technological and societal needs.

7.1 Food Safety and Public Health

Food safety is one of the most important contemporary issues connected to this project. Aflatoxins are toxic and carcinogenic compounds that can appear in agricultural products such as dried figs. If contaminated products are not detected before reaching consumers, they may create serious health risks.

Figion addresses this issue by providing an early pre-screening system. The system is not intended to fully replace laboratory analysis, but it can help detect risky figs before they continue through the production or export process. This supports safer food supply chains and reduces the possibility of contaminated products reaching the market.

7.2 Artificial Intelligence in Agriculture

Artificial intelligence is increasingly used in agriculture and food production for tasks such as classification, disease detection, quality control, and automation. Figion follows this trend by applying computer vision and deep learning to dried fig inspection.

The project shows how AI can improve traditional inspection methods. Instead of relying only on human observation under UV light, Figion uses image-based analysis to make the inspection process faster, more consistent, and more traceable.

7.3 Automation and Workforce Transformation

Industrial automation is changing the role of workers in many sectors. In the current dried fig inspection process, workers manually examine products under UV light, which can be repetitive and tiring. Figion automates part of this process.

This does not mean that human workers become unnecessary. Instead, their role can shift from continuous manual inspection to system monitoring, result verification, and quality control management. Therefore, the project supports a more technology-assisted working environment.

7.4 Data Reliability and AI Bias

A major contemporary issue in AI-based systems is data reliability. The performance of an AI model depends strongly on the quality, size, and diversity of the dataset used for training and testing. If the dataset is limited or not representative of real production conditions, the model may perform well in tests but fail in industrial use.

For Figion, this is especially important because lighting conditions, fig surface differences, camera quality, and contamination patterns can affect the results. Therefore, the system must be tested with diverse samples and, when possible, laboratory-confirmed ground truth data.

7.5 Traceability in Food Supply Chains

Modern food production increasingly requires traceability. Companies need to record how products are inspected, when they are inspected, and what decisions are made. This is important for audits, customer trust, and regulatory compliance.

Figion supports this issue by saving inspection results, images, timestamps, and batch information. These records can help quality managers review previous inspections, generate reports, and improve future production decisions.

7.6 Edge Computing and Local Processing

Another contemporary issue is the use of edge computing instead of cloud-based systems. Many industrial environments require fast response times, offline operation, and data privacy. Cloud-based systems may introduce latency, internet dependency, and additional cost.

Figion uses local processing on standard computer hardware. This makes the system more suitable for factory environments because it can work without continuous internet access and can provide real-time results.

7.7 Ethical Responsibility in AI-Based Food Inspection

Since Figion is related to food safety, ethical responsibility is very important. Incorrect classification can have serious consequences. A false negative may allow contaminated figs to pass inspection, while a false positive may cause healthy products to be rejected unnecessarily.

For this reason, the system should be used as a decision-support and pre-screening tool, not as the only final authority. Laboratory testing should still be used when definitive confirmation is needed. The project must also report its limitations clearly and avoid presenting the AI model as perfect.

8. New Tools and Technologies Used

During the Figion project, several software and hardware tools were used to develop a real-time, AI-based aflatoxin detection system. These technologies were selected to support image processing, deep learning inference, desktop user interaction, local data storage, hardware control, reporting, and project collaboration. The main tools and technologies used in the project are listed below:

1. Python
2. YOLOv11n
3. ONNX
4. OpenCV
5. PyQt6 / Qt
6. SQLite
7. CSV Export
8. USB Camera
9. 365 nm UV LED Array
10. UML
11. GitHub

9. Use of Library and Internet Resources

During the Figion project, library and Internet resources were used to understand the technical background of aflatoxin detection, UV-based inspection, computer vision, artificial intelligence models, and software/hardware integration. These resources helped the team compare similar systems, select suitable technologies, understand engineering principles, and support design decisions throughout the project.

The main library and Internet resources used in the project are listed below:

1. Academic papers and articles about aflatoxin detection
2. Food safety and dried fig export regulations
3. UV fluorescence-based inspection resources
4. Computer vision and image processing documentation
5. OpenCV documentation
6. YOLO / Ultralytics documentation

7. ONNX documentation
8. PyQt6 / Qt documentation
9. SQLite documentation
10. Python official documentation
11. UML and software design resources
12. GitHub repositories and technical examples

10. Test Results

The verification of the **Figion** system was conducted using the integrated prototype hardware and the custom-built dataset consisting of 6525 unique images (3457 healthy, 3068 contaminated). The results confirm that the system meets industrial requirements for aflatoxin pre-screening while maintaining high classification accuracy.

Test ID	Test Description	Expected Behavior	Actual Result	Status
TC-AI-001	Max Recall (Sensitivity)	Catch at least 95% of contaminated figs.	98.6% Recall	PASSED
TC-AI-002	Precision	Maintain a minimum of 85% Precision.	97.9% Precision	PASSED
TC-PERF-001	One-Second Rule	End-to-end latency below 1000ms.	Mostly < 850ms	PASSED
TC-UI-001	Live Video Rendering	Smooth rendering of processed frames.	Stable 30 FPS	PASSED
TC-EXC-001	Hardware Disconnect	Graceful notification during disconnect.	Visual indicator displayed	PASSED
TC-DAT-002	Data Integrity	ACID-compliant database operations.	Initial verification successful	PASSED

- **Performance Metrics:** The AI model, optimized in ONNX format, achieved a Recall of 98.6% and Precision of 97.9% by epoch 50, exceeding the safety thresholds required for public health obligations. The total processing latency remains consistently below 850ms, allowing the system to keep up with the manual production line speed of 175~kg/hour.
- **Exception Handling:** In the event of a camera or USB relay disconnection, the system does not crash or throw an unhandled error code. Instead, a specific visual indicator appears on the dashboard stating that the USB input is missing, allowing the operator to address the hardware issue immediately.
- **User Interface:** The live video feed is rendered at a stable 30 FPS, providing sufficient fluidity for real-time monitoring of the bounding boxes (Red for Aflatoxin, Green for Healthy) without taxing the CPU.
- **Data Reliability**: While preliminary tests indicate that ACID properties (Atomicity, Consistency, Isolation, Durability) are maintained during SQLite transactions, it is noted that further extensive testing with the final physical setup is required to verify full integrity during power interruptions.

11. GitHub and Project Web Page URLs

The source code, trained models, and documentation for the **Figion** project are publicly available at the following links:

- **Project Official Website:** https://figion.tech/
- **Main Software Repository (GitHub):** https://github.com/aycariko/Image-Processing-Based-Aflatoxin-Detection-System-for-Figs
- **Model Training & Evaluation Scripts:** https://github.com/aycariko/Aflatoxin_Model_Training/

12. User Manual and Installation Instructions

The **Figion** application is designed to run on standard workstation hardware using **edge computing** principles.

12.1 Installation Steps

1. **Environment Setup:** Create a Python 3.10+ virtual environment.
2. **Dependency Installation:** Install the required packages via the provided requirements.txt:
   pip install -r requirements.txt
3. **Model Deployment:** Ensure the trained figion_yolo.onnx file is placed in the models/ directory.

4. **Execution:** Launch the application using the command python main.py or the provided run.bat (Windows) / run.sh (Linux) scripts.

12.2 Operation Instructions

- **Startup:** At application launch, the system performs an automatic **Health Check**. If the camera is not detected, ensure the USB connection is secure and check the on-screen status indicator.
- **Scanning:** Place the dried figs under the 365nm UV LED illumination and click the **"Start Scan"** button to initiate the session.
- **Monitoring:** View real-time results on the dashboard. Use the **Live Counters** to track the total count and contamination ratio.
- **Session Termination:** Click **"Stop Scan"** to turn off UV lights and finalize the session. Click **"Export CSV"** to generate the audit trail report.

13. Limitations and Future Work

While the **Figion** prototype successfully demonstrates the feasibility of automated aflatoxin detection, several areas remain for further refinement:

Limitations:

- **Manual Hardware Control:** In the current version, the UV light source and conveyor belt are managed manually rather than through automated software triggers.
- **FPS Stabilization:** The video feed is currently capped at **30 FPS** to ensure CPU stability on standard laptop hardware; industrial-grade GPUs could push this to 60+ FPS.
- **Verification:** While database operations are technically **ACID-compliant**, full validation against sudden industrial power failures requires a specialized testing rig.

Future Work:

- **Full Automation:** Integrating the **USB Relay** controller directly into the scanning state machine to automate UV lighting and conveyor movement.
- **Advanced Data Validation:** Conducting extensive stress tests with the physical setup to guarantee 100% data integrity under power-loss scenarios.
- **Dual-Camera Scanning:** Adding a second camera to scan both the top and bottom surfaces of the figs simultaneously for improved detection coverage.
- **Cloud Reporting:** Developing an optional web-based management portal for factory supervisors to track multiple production lines remotely.

14. Conclusion

In conclusion, the Figion project presents a complete engineering solution for the automatic detection of aflatoxin risk in dried figs using UV imaging, artificial intelligence, and edge computing. Throughout the project, the problem definition, system requirements, final architecture, software and hardware design, implementation status, engineering impacts, contemporary issues, tools and technologies, resource usage, and test results were examined in detail.

The system was designed to improve the traditional manual inspection process by providing a faster, more consistent, and traceable pre-screening method. By using computer vision and AI-based inference, Figion aims to support food safety, reduce human error, improve production efficiency, and help prevent economic losses caused by contaminated product batches.

The project also shows the importance of engineering solutions in a wider context. Its impact is not limited to technical performance; it also contributes to global food safety, economic sustainability, environmental waste reduction, and public health. In addition, the project is related to contemporary issues such as AI in agriculture, automation, traceability, edge computing, and ethical responsibility in food inspection systems.

Overall, Figion demonstrates that modern software, hardware, and AI technologies can be combined to address a real industrial and societal problem. Although further improvements such as larger datasets, laboratory-confirmed validation, and industrial-scale testing are needed, the project provides a strong foundation for a practical, reliable, and scalable aflatoxin pre-screening system.

15. 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.

4. 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

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