[

FUNCTIONAL DESIGN SPECIFICATION for K00259724 FYP

TUS

Technological University of the Shannon:
Midlands Midwest

Ollscoil Teicneolaíochta na Sionainne:
Lár Tíre Iarthar Láir

Contents

REVISION HISTORY

Rev.	Date	Approval	REVISION SUMMARY
A0	05/11/23	_____	Initial Draft – Cormac Farrelly

Document Description:	AI Powered Industrial Automated Part Sorting System.
Status:	For Integration.
Client:	TUS.
Doc Name/Nr:	FDS_K00259724_CF

1.0 INTRODUCTION

This document outlines the functional requirements for the custom machine learning algorithm industrial part sorting system, which aims to classify industrial parts into three categories: damaged, undamaged, and unrecognized. The system incorporates a conveyor belt, servo motor, Siemens S7 1200 PLC, supporting HMI, and a Raspberry Pi 4 for algorithm storage. The Cognex camera captures images of the moving parts for classification. This FDS provides an overview of the system's key functions.

The creator of this document is the FYP leader working on the project. Any updates to this document will be the responsibility of the leader.

The document originator is accountable for the creation and development of the FS. They will indicate their name and date on the revision history in the ‘Revision Summary’ section and maintain it.

2.0 GLOSSARY

Table 1.0 Glossary	
Acronym	**Definition**
cGMP	common Good Manufacturing Practices
GAMP	Good Automation Manufacturing Practices
HMI	Human Machine Interface
PLC	Programmable Logic Controller
FDS	Functional Design Specification
URS	User Requirement Specification
AI	Artificial Intelligence

3.0 SYSTEM OVERVIEW

In this final year project, a system for automating the classification and sorting of industrial parts has been developed. The architecture of this system has several components working together. It starts with a conveyor belt powered by a servo motor, responsible for moving the industrial parts for inspection. An encoder has been used to gauge the distance each part traverses. As the parts traverse along the conveyor, they pass beneath a camera, a Cognex camera or a compatible alternative. The camera plays a role in capturing images of the passing parts. These images are transmitted to a Raspberry Pi 4, serving as the storage for a machine learning algorithm crafted using TensorFlow and Keras. The primary objective of this algorithm is to analyze the images and make determinations regarding the condition of the parts.

This algorithm uses a sophisticated classification process, categorizing each part into one of three distinct classes: damaged, undamaged, or unrecognized. They are classified by the details and features extracted from the images. The outcomes of this classification, expressed in binary form, are then dispatched to a Siemens S7 1200 PLC, which will move the conveyor and collection chamber.

As the process unfolds, it does so continuously until all the parts on the conveyor have been inspected and sorted. The comprehensive process is supervised by sensors at both the part collection and release points.

3.1 Control System

The control system is the cornerstone of our project, ensuring seamless and efficient operation, and it prominently features the Siemens S7-1200 PLC at its core. This PLC is located within the electrical panel and serves as the central hub for overseeing the entire project workflow.

3.1.1 Siemens S7-1200 PLC:

At the heart of the control system, the Siemens S7-1200 PLC is a powerful and versatile industrial automation controller. It has been programmed to enable control over our project's operations. This PLC offers a wide range of capabilities, from real-time data processing to integration with external devices, making it a reliable and efficient choice.

3.1.2 Digital Input and Output Modules:

The Siemens S7-1200 PLC has digital input and output modules, enabling communication with various sensors and actuators. These modules facilitate bi-directional communication, allowing the PLC to receive data from the sensors and transmit commands to the project's components. This bidirectional data exchange guarantees that our system operates with precision.

3.1.3 Human Machine Interface (HMI):

Serving as the link between our project and human operators is the advanced HMI, which allows for user-friendly control. The HMI allows operators to initiate, oversee, and manage the system's operations. With a high-resolution display and responsive touch controls, the HMI ensures that operators can interact with the project.

3.1.4 Ethernet Communication:

To facilitate real-time data exchange and synchronization, the Siemens S7-1200 PLC and Raspberry Pi are interconnected through an Ethernet network. This network transmits data, commands, and results, ensuring that our project operates seamlessly.

3.2 Vision System

Within our system, a component is the vision inspection process. It relies on a camera to capture images of industrial parts as they traverse the conveyor belt. These captured images are subjected to analysis and classification via a machine learning algorithm, all executed on the Raspberry Pi. This intelligent algorithm allows the system to categorize the parts into three distinct classes: damaged, undamaged, or unrecognized.

3.2.1 Camera Image Capture:

At the heart of our vision inspection process lies a high-quality camera, positioned to capture images of the industrial parts as they progress along the conveyor belt. The camera's lens is calibrated to ensure precise and high-resolution image acquisition, making it possible to view each part with exceptional detail.

3.2.2 Custom Machine Learning Algorithm:

The custom machine learning algorithm, made in Python using TensorFlow and Keras, is the brains of the operation. It is hosted on the Raspberry Pi, ready to execute complex image processing tasks. This algorithm has been finely tuned to identify the subtlest aspects in each part's condition, allowing it to make accurate classifications. It can differentiate between damaged, undamaged, and unrecognized parts.

3.2.3 Classification of Parts:

Upon processing each captured image, the machine learning algorithm classifies the industrial parts into one of three categories: damaged, undamaged, or unrecognized. Its decision-making process is based on a careful analysis of the part's features and condition, ensuring that the categorization is accurate and reliable. This classification step is an important part of our project, as it directly impacts the actions and decisions taken by the system.

3.2.4 Communication with the PLC:

The machine learning algorithm's classification results are relayed to the PLC in a binary format. The PLC uses this information to direct the parts to their respective collection chambers.

3.3 Key Objectives

3.3.1 Develop a Machine Learning Algorithm:

The objective of our FYP is to create a machine learning algorithm with the capability to accurately classify industrial parts into one of three distinct categories: damaged, undamaged, or unrecognized. This algorithm, coded in Python and employing TensorFlow and Keras, serves as the brain of our system. Its development is the essence of our project.

3.3.2 Integrate an Operator HMI Interface:

Our FYP incorporates an operator Human Machine Interface (HMI) interface to provide an intuitive and user-friendly platform for controlling and monitoring the system's operations. This interface ensures that operators, whether for educational or commercial purposes, can interact with the system effortlessly, initiating, overseeing, and managing its functionalities with ease.

3.3.3 Implement a Robust PLC Control System:

The central nervous system of our project is a Programmable Logic Controller (PLC). This component has been carefully selected and configured to control and coordinate the diverse components of our system. It acts as the leader, managing the communication between different elements and making real-time decisions. The PLC ensures the system's operation and data flow.

3.3.4 Deploy a Vision Inspection System:

Incorporating a vision inspection system, our FYP aims to capture images of industrial parts in motion. This system is designed to be highly responsive and precise, enabling the collection of detailed visual data. These images are subjected to in-depth analysis and classification, a crucial step in our project.

3.4 Data Handling Requirements

In our system, effective data handling is important to ensure smooth operation, coordination, and communication between various components. The heart of this data management lies in the Siemens S7-1200 PLC.

3.4.1 Centralized Data Management:

The Siemens S7-1200 PLC serves as the central hub for data management. It acts as the center, collecting & processing data from all components within the system. This centralized approach streamlines the data flow, ensuring that information is efficiently shared and utilized.

3.4.2 Communication Backbone:

A fundamental role of the PLC is to establish and maintain communication links between components of the system. It communicates VIA TCP/IP Ethernet to the RaspberryPi 4 as well as other components like the encoders etc.

3.4.3 Real-time Data Flow:

One of the features of the PLC is its ability to manage data in real-time. This ensures that the information exchanged among system components is up-to-date and accurate.

3.4.4 Data Security:

While managing data flow, the PLC also prioritizes data security. Measures are in place to protect sensitive information and ensure that only authorized users can access and modify critical data.

3.4.5 Feedback Loop:

The PLC establishes a feedback loop within the system, providing essential data to operators and other system components. This loop allows for monitoring, control, and quick response to changing conditions, ultimately enhancing the efficiency and safety of the system.

3.5 Security Definitions

In our system, security is a fundamental aspect, ensuring that the operations and data remain protected and accessible only to authorized individuals. Let's explore the security definitions and how we safeguard the system:

3.5.1 User Access Levels

The system employs a user access hierarchy to control and manage permissions effectively. This hierarchical approach provides different levels of access based on user roles. These levels include:

3.5.2 Default Access:

Default users are granted access to fundamental system controls. This minimal access level allows for basic system interaction and

monitoring.

3.5.3 Engineer and Operator Access:

Engineers and operators who possess the requisite training and expertise, are granted elevated access privileges. They can access advanced functionalities, including control over individual system components. This access level empowers them to manage the system's operations efficiently.

3.5.4 Password Protection:

To ensure that only authorized personnel can access the system, robust password protection is implemented. Users are required to input their credentials, including usernames and passwords, during login. This process verifies their identity and grants access based on their assigned access level.

3.6 Conformance with Non-Functional Requirements

Our system adheres to a set of non-functional requirements, which are essential for its performance, reliability, and maintainability:

3.6.1 Continuous Operation:

The system is designed for continuous operation, ensuring that it can function seamlessly without significant interruptions. This uninterrupted operation is vital for applications where reliability and uptime are critical.

3.6.2 Error Recovery:

The system incorporates robust error recovery mechanisms. In the event of errors or faults, the system can recover. Immediate corrective actions are taken to address the issue and restore normal operation, minimizing downtime and disruptions.

3.6.3 Maintainability:

Our system is engineered to be highly maintainable. Components used in the system are carefully selected to minimize wear and tear. This design approach ensures that maintenance can be carried out swiftly, reducing downtime and maximizing system availability.

3.6.4 Data Recovery in the Event of a Failure:

To guarantee the integrity and functionality of the system, all programs developed for the FYP are diligently backed up on storage devices. These backups serve as a safeguard against data loss or corruption.

3.7 Machine Learning Algorithm

Current Model in development trained on x-ray images.

```
import matplotlib.pyplot as plt
import pandas as pd
from keras import layers, models, optimizers, callbacks
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import ResNet50V2

# Prepare the Dataset
train_dir = r'C:\Users\cormac.farrelly\OneDrive - Regeneron Pharmaceuticals,
Inc\Desktop\Dissertation\MedicalML\xray-dataset\train'
validation_dir = r'C:\Users\cormac.farrelly\OneDrive - Regeneron Pharmaceuticals,
Inc\Desktop\Dissertation\MedicalML\xray-dataset\val'

# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

validation_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 16

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# Load the ResNet-101 model without the top layers
base_model = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224,
3))

# Freeze the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Add new layers on top of the pre-trained model
```

```
model = models.Sequential()
model.add(base_model)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dropout(0.5))  # Add dropout for regularization
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.3))  # Add dropout for regularization
model.add(layers.Dense(1, activation='sigmoid'))

# Use an adaptive learning rate optimizer like Adam
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(learning_rate=1e-3),  # Adjust learning rate
              metrics=['accuracy'])

# Add early stopping
early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

history = model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=20,  # Increase epochs for more training
        validation_data=validation_generator,
        validation_steps=len(validation_generator),
        callbacks=[early_stopping],
        verbose=1
    )

# Save training history to a CSV file
history_df = pd.DataFrame(history.history)
history_df.to_csv('training_history_resnet.csv', index=False)

# Save the model
model.save('MedicalML_ResNet.h5')
```

3.7.1 Explanation

1. Importing Libraries:
The code begins by importing necessary Python libraries, including:
   - `matplotlib.pyplot`: Used for plotting and visualizing data.
   - `pandas`: Used for data manipulation and storage.
   - Various modules and functions from the Keras library, which is a high-level neural networks API.
   - `ResNet50V2` from the Keras applications, which is a pre-trained deep learning model.

2. Preparing the Dataset: This section defines the directories where the training and validation datasets are stored. The dataset is likely organized into subdirectories representing different classes (e.g., 'normal' and 'abnormal').

3. Data Augmentation: Data augmentation is a technique to artificially increase the diversity of the training dataset by applying random transformations to the images. In this script,

`ImageDataGenerator` is used to apply various transformations like rotation, shift, shear, zoom, and horizontal flip to the images. Data augmentation helps the model generalize better.

4. Batch Size: The `batch_size` variable is set to 16, which determines the number of images processed in each training iteration.

5. Data Generators: `train_generator` and `validation_generator` are created using the data augmentation settings and the paths to the training and validation datasets. These generators yield batches of augmented images during training.

6. Loading Pre-trained Model: The ResNet-50V2 pre-trained model is loaded from Keras with the 'imagenet' weights. The 'include_top' parameter is set to `False`, indicating that the top (classification) layers of the model are not included. This is because you are going to add your own classification layers on top of the pre-trained model.

7. Freezing Pre-trained Layers: The script iterates over the layers of the pre-trained model and sets them as non-trainable (frozen). This is a common practice when you want to use pre-trained features and fine-tune the model for a specific task to prevent overfitting.

8. Adding New Layers: A new model is created using Keras' Sequential API. Layers are added to this model, starting with the pre-trained ResNet-50V2 base, followed by a Global Average Pooling layer, dropout layers for regularization, and finally, a dense layer with one neuron and a sigmoid activation function, which is common for binary classification tasks.

9. Compiling the Model: The model is compiled with the loss function (binary cross-entropy), optimizer (Adam), and metrics (accuracy). An adaptive learning rate optimizer like Adam is commonly used in deep learning.

10. Early Stopping: An early stopping callback is defined, which monitors the validation loss and stops training if it doesn't improve for a certain number of epochs (patience). This helps prevent overfitting.

11. Training the Model: The model is trained using the `fit` method, specifying the training and validation generators, the number of steps per epoch, the number of epochs, and the early stopping callback. Training history is recorded during this process.

12. Saving Training History: The training history, including metrics like loss and accuracy, is saved to a CSV file named 'training_history_resnet.csv.'

13. Saving the Model: Finally, the trained model is saved to a file named 'MedicalML_ResNet.h5.'

5.0 COMPATIBILITY AND UTILITIES

5.1 System Compatibility

5.1.1 Software Compatibility

5.1.1.1 The system shall be meticulously crafted to work with commonly used software tools in the development of machine learning algorithms. This includes Python, TensorFlow, and Keras, ensuring that these essential components interact flawlessly.

5.1.1.2 The embedded software shall adhere to industry best practices in software development. It will be designed to ensure compatibility with standard libraries and guidelines for machine learning in Python.

5.1.2 Preferred Manufacturers

5.1.2.1 The system's design will prioritize compatibility including cameras and sensors suitable for industrial image capture and processing.

5.2 Utility Requirements

5.2.1 Power Supply

5.2.1.1 The system's energy needs shall be met through the utilization of

standard electrical services. This includes a 230-volt, 1-phase, 60 Hz power supply with proper grounding, ensuring a stable and reliable power source.

5.3 Power Failure and Recovery

5.3.1 System Recovery on Power Failure:

5.3.1.1 Safe State on Power Failure: In the event of a power failure, the system will transition into a safe state.

5.3.1.2 Stop All Motion: All system motion, related to the conveyor and image capture, will cease immediately.

5.3.1.3 Reset Required: To restart the system after power failure, a manual reset operation is necessary.

5.3.1.4 Prevent Damage: The system is designed to ensure that no damage occurs due to the transition into a safe state.

5.3.2 Power Restoration:

5.3.2.1 Operator Input Required: The system will not automatically restart upon power restoration; operator input is essential.

5.3.3 Data and Program Retention:

5.3.3.1 Program Retention: All equipment shall be designed to retain the machine learning algorithm and PLC program in case of power loss.

5.3.3.2 Minimal Operator Actions: Recovering from power loss shall require minimal operator actions.

5.4 Emergency Stop

5.4.1 Emergency Stop Buttons

5.4.1.1 Emergency Stop Buttons: Emergency stop buttons shall be provided within reach for the operator.

5.4.1.2 Activation Consequences: When activated, the emergency stop shall immediately halt all system motion.

5.4.1.3 No Damage: Activation of the emergency stop shall not result in any damage to the system.

5.4.1.4 Operator Intervention: The system cannot restart without operator intervention and a reset operation.

5.4.1.5 Device Power Disconnect: All emergency stops shall be hardwired to disconnect device power.

5.4.1.6 HMI Alarm: An alarm will be displayed on the HMI to inform the operator of the emergency stop activation.

5.5 Alarms and Warnings

5.5.1 System Alarms

5.5.1.1 Image Capture Error: An alarm will be triggered if there's an issue with image capture.

5.5.1.2 Algorithm Fault: An alarm will be activated if there's a fault in the machine learning algorithm.

5.5.1.3 Conveyor Belt Error: An alarm for conveyor belt issues.

5.5.1.4 PLC Communication Error: Alarm for communication faults between the Raspberry Pi and PLC.

5.5.2 Operator Messages

5.5.2.1 Operator Guidance: Display messages on the HMI to inform the operator.

5.5.2.2 about alarms and provide instructions for resolution.

6.0 PROCEDURAL CONSTRAINTS

6.1 Regulatory Compliance

7.1.1 Safety and Regulatory Standards

6.1.1.1 Adherence to safety and quality standards will be part of the system's design and operation, ensuring it complies with established regulatory norms like GAMP, cGMP, GMP.