



**TUS**

**Technological University of the Shannon:  
Midlands Midwest**

Ollscoil Teicneolaíochta na Sionainne:  
Lár Tíre Iarthar Láir

---

# FINAL YEAR PROJECT

---

## Machine Learning Part Classification System

Industrial Automation & Robotics Systems

Cormac Farrelly – K00259724

SEPTEMBER 2, 2024

TUS

## 1 Declaration

I, Cormac Farrelly, declare this thesis as my own. All the research, design, development and documentation in this document was done by me alone. In this thesis, references and citations of any other materials sourced from someone else were done.

This work has not been submitted anywhere else for any academic award or a professional qualification.

Cormac Farrelly

## 2 Dedication

The student wishes to thank the Technological University of the Shannon in particular for their assistance, knowledge, with this final year project.

The student also would like to thank Jeffrey Ryan for supervising the student throughout this project and all his help and support. The student wishes to thank Thomas Burns for his help in this project for providing all appropriate appliances and his recommendations.

### 3 Contents

1	Declaration .....	1
2	Dedication.....	2
4	Table of Figures .....	7
5	Abstract .....	8
6	Introduction .....	10
7	Background.....	13
8	Algorithm Development.....	14
8.1	Data Preparation .....	14
8.2	Data Augmentation.....	15
8.3	Image Loading and Preprocessing.....	16
8.4	Model Architecture.....	17
8.5	Model Compilation and Training .....	19
9	Establishing Communication Between Raspberry Pi and Siemens PLC.....	23
9.1	Learning and Implementing Python Snap7.....	23
9.2	Networking Challenges and Subnet Configuration .....	24
9.3	Troubleshooting Connection Refusals and CPU Configuration .....	24
9.4	Refining the Communication Process.....	27
10	Wiring and Configuring the System .....	28

10.1	Initial Setup and Powering the PLC .....	28
10.2	Wiring the Conveyor Belt and Addressing Issues.....	30
10.3	Finalizing Initial Wiring and Testing .....	31
11	Integrating and Configuring the Cognex INSIGHT 7801 Camera.....	32
11.1	Initial Setup and Wiring of the Camera .....	32
11.2	Connecting the Camera to the Laptop .....	34
11.3	Camera Configuration and Mounting.....	34
11.4	Camera Triggering and Image Capture .....	35
12	Integrating Camera Image Transfer with FTP and Training the Algorithm .....	37
12.1	Setting Up FTP Image Transfer .....	37
12.2	Handling Multiple Images and FTP Queue Management .....	38
12.3	Training the Machine Learning Algorithm.....	38
1.	Image Collection: .....	39
2.	Image Preparation:.....	39
3.	Algorithm Training:.....	39
4.	Testing and Validation:.....	39
12.4	Summary .....	40
13	Implementation of Safety Features and System Integration .....	41
13.1	Implementing Safety Features.....	41
13.1.1	Emergency Stop (E-Stop) Relay:.....	41

13.1.2	JSBR4 E-Stop Relay into the Automated System .....	42
13.1.3	Incorporation of Miniature Circuit Breakers (MCBs): .....	45
13.1.4	Panel and Wiring Organization: .....	46
14	System Integration and Final Testing .....	48
14.1.1	System Workflow: .....	48
15	Detailed Breakdown of the Python Code for Image Processing and PLC Communication ..	50
15.1.1	Overview .....	50
15.1.2	Importing Libraries .....	50
15.1.3	Model Loading.....	51
15.1.4	Directory Setup.....	51
15.1.5	PLC Communication Setup.....	52
15.1.6	File Readiness Check .....	53
15.1.7	Image Processing .....	53
15.1.8	Sending Results to PLC.....	55
15.1.9	Main Loop for Monitoring and Processing Images .....	57
15.1.10	Summary .....	59
16	Detailed Breakdown of the PLC Program.....	60
16.1	PLC Program Overview.....	60
16.1.1	1. Initial Setup and Memory Bit Management .....	60
16.1.2	2. Sensor Integration and Conveyor Control .....	62

16.1.3	3. Communication with Raspberry Pi and Result Handling .....	63
17	Cognex In-Sight Software Configuration and Troubleshooting.....	65
17.1	Configuration of the Patterns Max Tool .....	65
17.2	FTP Server Setup and Troubleshooting.....	65
17.2.1	Final Implementation .....	66
18	Risk Assessment and Health & Safety Considerations.....	68
18.1	Risk Assessment .....	68
19	Health & Safety Considerations .....	70
19.1	Emergency Stop (E-Stop) Integration: .....	70
20	Sustainability Considerations .....	71
21	Conclusions and Further Developments .....	72
21.1	Conclusions .....	72
21.2	Further Developments.....	72
22	References .....	73
23	Appendices .....	75

## 4 Table of Figures

Figure 1 FYP Rig .....	12
Figure 2 Initial Concept Design.....	12
Figure 3 Neural Network.....	14
Figure 4 Activation Functions .....	17
Figure 5 Model.....	22
Figure 6 Raspberry Pi Comms.....	26
Figure 7 Initial Wiring of PLC .....	29
Figure 8 Implementation of VSD .....	31
Figure 9 Cognex 7801 Power .....	33
Figure 10 Cognex camera set up and wired in .....	36
Figure 11 Training terminal demonstration.....	40
Figure 12 Estop Relay .....	44
Figure 13 Final Panel Fully Wired.....	47
Figure 14 Network Switch for System.....	49
Figure 15 PLC Network 1 .....	61
Figure 16 PLC Network 2 .....	62
Figure 17 PLC Network 3 .....	63
Figure 18 PLC Network 4 .....	64
Figure 19 PLC Network 5 .....	64
Figure 20 FTP job .....	67



## 5 Abstract

This thesis is of Cormac Farrelly's project for the Technological University of the Shannon (TUS) Industrial Automation & Robotics Engineering course. The project's goal is to build a machine learning system that integrates into automation to classify parts using TensorFlow and Keras.

The thesis opens with an overview of the project's goals, explaining the reason for the system's creation and its use to industrial automation. The project's technological components are the Cognex Insight 7801 camera, Siemens S7-1200 PLC, and Raspberry Pi 4, Servo Motor and Sensor which together made up the system.

The machine learning algorithm, which separates parts into pass and fail categories, was the priority. A detailed overview of the algorithm's use with the PLC & Camera is shown, with an emphasis on how the live image acquisition, sorting, and classification works together.

The thesis gives a large amount of space to talk about the difficulties that came about during the project, such as creating the algorithm, setting up the hardware, and establishing connection between the Raspberry Pi and the PLC and also establishing a connection between the camera through an FTP server to the Pi. It also outlines the actions done to get over these obstacles.

The report's conclusion shows a reflection on the project's learning outcomes.

This thesis also includes references referenced with the Harvard referencing style and is in in a formal tone for academic submission.

## 6 Introduction

To classify parts, this thesis details the design, development, and use of an automation system that combines machine learning with Python - a higher-level programming language, specifically object-oriented programming. In order to train an algorithm using real world pics for classifying parts as they go along a conveyor belt, the project uses of TensorFlow and Keras.

The panel incorporates such electrical components as a speed dial, emergency stop relay, a variable speed drive (VSD) Altivar 11, Siemens PLC S7-1200 (type 1215 DC/DC/DC), single phase ac servo motor, and a 24V PSU among wiring the system.

The project initiates when the system is turned on, causing the conveyor belt to start moving. After a certain part has been placed on the conveyor belt, it continues to move until the sensor has been activated. At this point, there is a Cognex In-Sight 7801 camera that is configured to capture images in intervals of 300 milliseconds, and the camera takes a picture of the part. The camera utilizes the PatternMax function of In-sight software to train it with the scope and details of the part.

When the image is taken by the camera, the image is uploaded to an FTP source whose IP address is 192.168.0.100 that is a Raspberry pi. A Python program installed in Raspberry pi checks the FTP folder for new images all the time. If a new image is found, the program moves it to the 'images' folder into which images are stored for further processing. The next step is putting the image through a trained machine learning algorithm developed by the student, which classifies the part as either pass or fail. To stop redundancy, the image is then moved to a 'scanned' folder after processing. The result is sent back to the Siemens PLC using the Snap7 Python function, which allows communication between Python and the PLC via the IP address

192.168.0.1. Depending on the classification, the PLC updates a data block, changing a specific bit to show whether the part has passed or failed.

Whether the part fails or passes the test, there is a light in the control screen for pass and fail which lights for only two seconds to indicate that the part has been worked on and takes it into the chamber for collection. The conveyor belt is still running, and the next part is ready for processing.

For protection mechanisms in the system, the system is used with the E-stop button in place.

Under such conditions, once the E-stop button is pressed, the E-stop relay immediately disables the supply of power to KM1 contactor, causing an instant shut down of the VSD and the conveyor belt. This thesis will go through each part and phase of the project in detail, showing the use of machine learning with automation and the challenges overcome during development. The project aims to show how modern technologies such as Python and machine learning can be used in real world environments.



Figure 1 FYP Rig

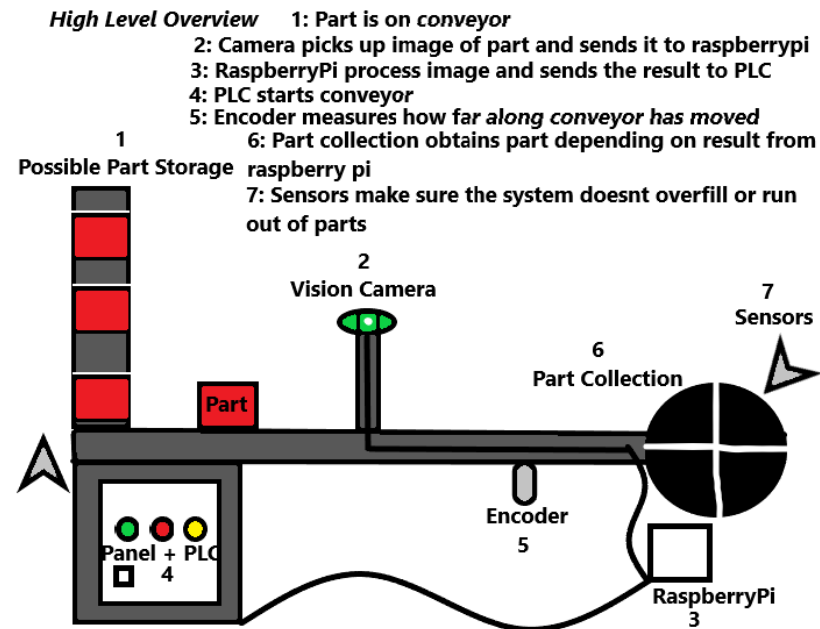


Figure 2 Initial Concept Design

## 7 Background

The student who finished this project is a fourth-year student of the Industrial Automation & Robotics Engineering course at the Technological University of the Shannon (TUS). Over their academic careers, they have developed a strong foundation in programming, robotics, and automation, which has been essential to the success of this project.

During their time at TUS, the student has learned about a variety of subjects, including control systems, wiring and advanced programming techniques. Thanks to their academic experiences, they now possess the knowledge and practical skills necessary to take on difficult engineering issues, such as the ones our final year project brings.

In addition to the academic work, the student has also acquired practical knowledge through internships and placements. They completed a work placement at Mergon in the second year, where they learned about some practical aspects of industrial systems and procedures. During the course, the student also undertook an internship at Regeneron Pharmaceuticals. They were trained on cutting edge automation system during their internship and were assigned tasks in projects.

The education acquired by the student at TUS, complemented with practical work experience at Mergon and Regeneron prepared him for tackling the challenges of the advanced automation system design and implementation described in this thesis. Their input on this project, has been a fusion of academic learning over the years, and practical work experience.

## 8 Algorithm Development

Machine learning is a tool that developed and trained the project's classification system using a python and the Keras package. This part shows the student's algorithm development process and shows what he did.

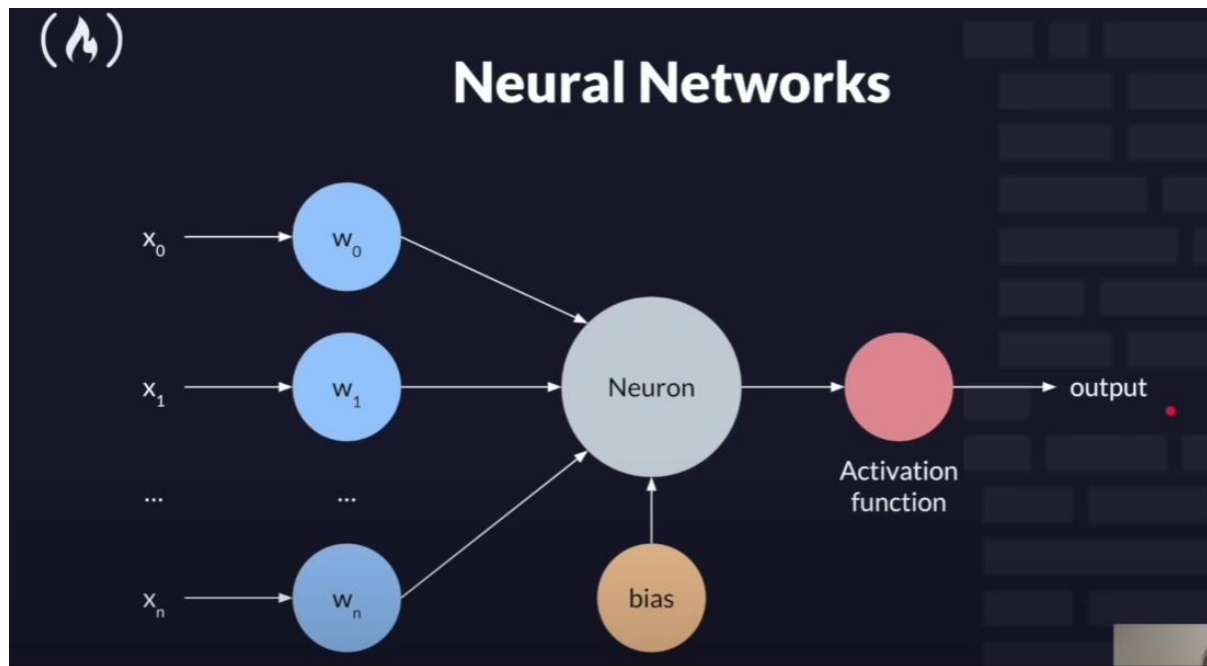


Figure 3 Neural Network

### 8.1 Data Preparation

The dataset had to be prepared in order for training and validation to be carried out. The student organized the dataset into two main directories, one which contained the training data and the other for validation data. Each of the main folders contained subfolders that were the several categories that the model was supposed to classify.

Here is how the directories were defined:

```
train_dir = r'C:\Users\cfarr\Desktop\fyp_pi_code\fyp_realtime\train'
```

Module: Final Year Project

```
validation_dir = r'C:\Users\cfarr\Desktop\fyp_pi_code\fyp_realtime\val'
```

## 8.2 Data Augmentation

The student used data augmentation techniques with Keras's `ImageDataGenerator` to increase the model's ability. Using random transformations like rotations, shifts, shears, and flips on the training images is known as data augmentation. By artificially expanding the dataset, this method better the model's ability to learn more traits and adapt to variances in the real world.

The data augmentation procedure is demonstrated by the following code snippet:

```
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  
  
)
```



### 8.3 Image Loading and Preprocessing

The student then loaded and pre-processed the photos from the directories using the `ImageDataGenerator` objects. The `flow\_from\_directory` function was used to do this, since it automatically identifies the images according to their folder structure and resizes them to the necessary input size of 224x224 pixels. To help the model learn, the photos were also batched and scrambled during training.

The following configuration was used for the image loading procedure for the training and validation datasets:

```
train_generator = train_datagen.flow_from_directory(

    train_dir,

    target_size=(224, 224),

    batch_size=32,

    class_mode='binary',

    shuffle=True

)

validation_generator = validation_datagen.flow_from_directory(

    validation_dir,

    target_size=(224, 224),

    batch_size=32,
```

```
class_mode='binary',  
  
shuffle=False  
)
```

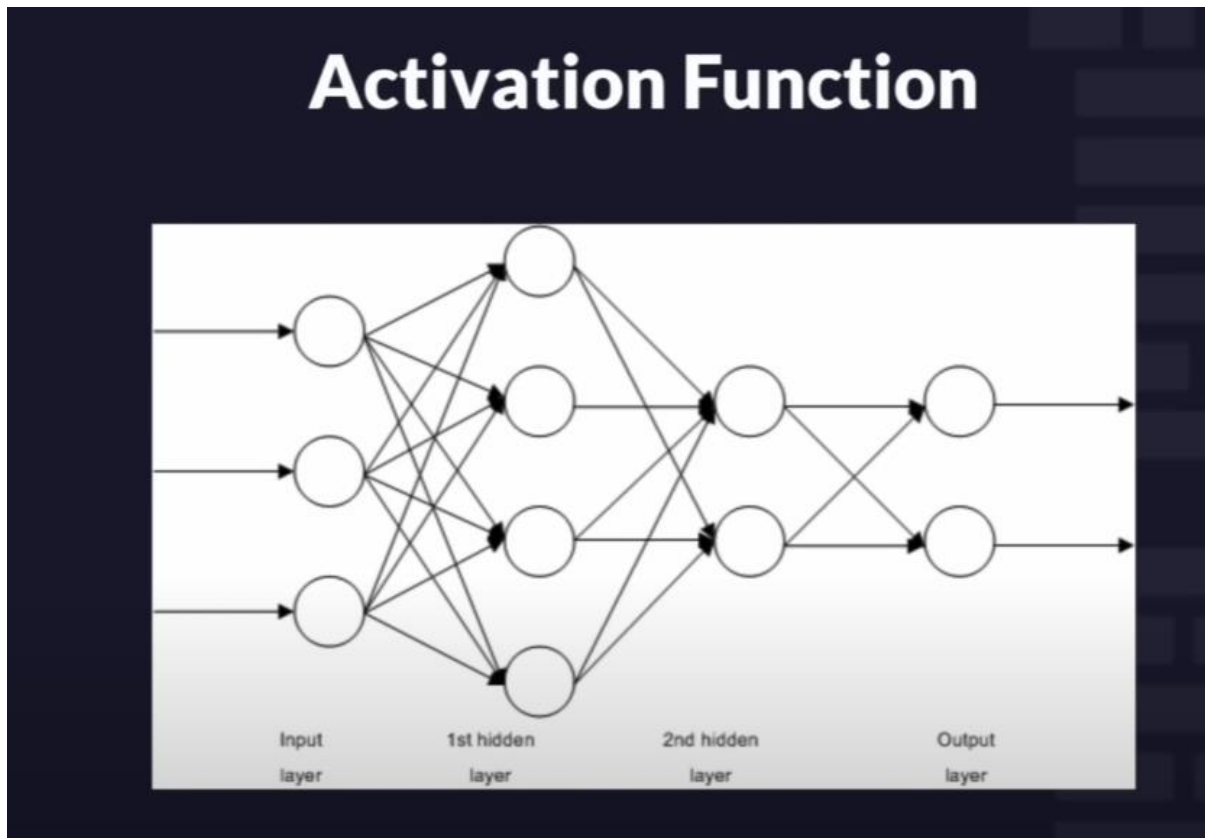


Figure 4 Activation Functions

## 8.4 Model Architecture

The student decided to use the ResNet50V2 model, a convolutional neural network that has already been pre-trained on the ImageNet dataset, in a transfer learning strategy. Transfer learning is a strategy that has improvements in training speed and accuracy, especially when working with limited data.

The top classification layers of the ResNet50V2 model were loaded empty, allowing the student to modify the final layers in accordance with the project's requirements:

```
base_model = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

The student stopped updating the layers of the ResNet50V2 model during training to maintain the features that were learned from the pre-trained model:

```
for layer in base_model.layers:

    layer.trainable = False
```

The frozen base model was then modified by the student by adding more layers to make it suitable for the binary classification task:

```
model = models.Sequential()

model.add(base_model)

model.add(layers.GlobalAveragePooling2D())

model.add(layers.Dropout(0.5)) # Dropout for regularization

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dropout(0.3)) # Dropout for regularization

model.add(layers.Dense(1, activation='sigmoid'))
```

The following custom layers were included:

- **GlobalAveragePooling2D:** This layer created a single long feature vector by reducing the spatial dimensions of the output from the convolutional layers.
- **Dropout Layers:** During training, a portion of the neurons are randomly disabled to prevent overfitting.
- **Dense Layers:** A sigmoid activation function was used by the last dense layers to carry out the binary classification. They produced a probability score that indicated whether the component was rated as "pass" or "fail."

(Keras, 2024)

(Keras, 2024)

(Keras, 2024)

## 8.5 Model Compilation and Training

The binary cross-entropy loss function, which is common for binary classification problems, was chosen by the student to build the model. The model's weights were updated during training using the Adam optimizer.

```
model.compile(loss='binary_crossentropy',  
              optimizer=optimizers.Adam(learning_rate=1e-3),  
              metrics=['accuracy'])
```

To avoid overfitting, the student also put in place an early stopping function. This required keeping an eye on the validation loss during training and stopping if no progress was seen for four epochs in a row.

```
early_stopping = callbacks.EarlyStopping(  
  
    monitor='val_loss',  
  
    patience=4,  
  
    restore_best_weights=True  
  
)
```

Lastly, the model went through 14 epochs of training, and its performance was checked during the training and validation phases:

```
history = model.fit(  
  
    train_generator,  
  
    steps_per_epoch=len(train_generator),  
  
    epochs=14,  
  
    validation_data=validation_generator,  
  
    validation_steps=len(validation_generator),  
  
    callbacks=[early_stopping],  
  
    verbose=1  
  
)
```

The model was saved after training so that it could be used for real-time part categorization on the conveyor belt:

```
model.save('FYP_iteration1.h5')
```

(Youtube, 2020)

(TensorFlow, 2024) (Keras, 2024) (arxiv, 2015)

```

fyp.py x
C: > Users > cfarr > Desktop > fyp_pi_code1 > fyp.py > ...
1  from keras import layers, models, optimizers, callbacks
2  from keras.preprocessing.image import ImageDataGenerator
3  from keras.applications import ResNet50V2
4
5  # Prepare the Dataset
6  train_dir = r'C:\Users\cfarr\Desktop\fyp_pi_code\fyp_realtime\train'
7  validation_dir = r'C:\Users\cfarr\Desktop\fyp_pi_code\fyp_realtime\val'
8
9  # Data augmentation
10 train_datagen = ImageDataGenerator(
11     rescale=1./255,
12     rotation_range=40,
13     width_shift_range=0.2,
14     height_shift_range=0.2,
15     shear_range=0.2,
16     zoom_range=0.2,
17     horizontal_flip=True
18 )
19
20 validation_datagen = ImageDataGenerator(rescale=1./255)
21
22 batch_size = 32
23
24 train_generator = train_datagen.flow_from_directory(
25     train_dir,
26     target_size=(224, 224),
27     batch_size=batch_size,
28     class_mode='binary',
29     shuffle=True
30 )
31
32 validation_generator = validation_datagen.flow_from_directory(
33     validation_dir,
34     target_size=(224, 224),
35     batch_size=batch_size,
36     class_mode='binary',
37     shuffle=False
38 )
39
40 # Load the ResNet-101 model without the top layers
41 base_model = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
42
43 # Freeze the pre-trained layers
44 for layer in base_model.layers:
45     layer.trainable = False
46
47 # Add new layers on top of the pre-trained model
48 model = models.Sequential()
49 model.add(base_model)
50 model.add(layers.GlobalAveragePooling2D())
51 model.add(layers.Dropout(0.5)) # Add dropout for regularization
52 model.add(layers.Dense(128, activation='relu'))
53 model.add(layers.Dropout(0.3)) # Add dropout for regularization
54 model.add(layers.Dense(1, activation='sigmoid'))
55
56 # Use an adaptive learning rate optimizer like Adam
57 model.compile(loss='binary_crossentropy',
58             optimizer=optimizers.Adam(learning_rate=1e-3), # Adjust learning rate
59             metrics=['accuracy'])
60
61 # Add early stopping
62 early_stopping = callbacks.EarlyStopping(
63     monitor='val_loss',
64     patience=4,
65     restore_best_weights=True
66 )
67
68 history = model.fit(

```

Figure 5 Model

The next stage came the task of connecting the Raspberry Pi to a Siemens PLC once the machine learning algorithm had been compiled and trained. This was important in such a way that the whole system worked by the Raspberry Pi communicating the classifying results to the PLC.

## 9 Establishing Communication Between Raspberry Pi and Siemens PLC

The student placed great emphasis on ensuring that the Raspberry Pi and the Siemens PLC could communicate with ease before proceeding to the wiring stage. This was a critical step in the project as the overall performance of the system was largely dependent on the information exchanged between these devices.

The initial stage included getting a rig of the Siemens PLC, which served as a prototyping platform. The primary objective was solving connection of the Raspberry Pi with the PLC.

### 9.1 Learning and Implementing Python Snap7

The first challenge was getting accustomed to the new Snap 7 libraries for Python to be able to communicate with Siemens S7 PLCs. The student had to get used to Snap 7 in order to upload and configure the library on the Raspberry Pi. Snap 7 supports the built-in network interface on PLCs and lets the user read from and write data, via python scripts to Siemens's PLC memory areas.

The Raspberry Pi needed to be set up properly first. This was achieved by making provision for remote access to the Raspberry Pi from a laptop using Visual Studio Code's SSH (Secure Shell) features. The student was able to SSH into the Raspberry Pi and run their development on the laptop, making it easier for those who needed a laptop as a development workstation.



## 9.2 Networking Challenges and Subnet Configuration

The disparity in subnet configuration of the Raspberry Pi and the Siemens PLC caused major issues. The Raspberry Pi network settings for the two devices to work on the same network must be adjusted. The student had to go through some Linux networking commands and change the Raspberry Pi IP address to that of the PLC subnet.

There was a need to learn how to do static configuration of the Raspberry Pi to change the address to one that was in the PLC network. One of the tasks that proved essential for the communication of the Raspberry Pi to the PLC was setting the subnets in harmony.

## 9.3 Troubleshooting Connection Refusals and CPU Configuration

When first attempting to connect the Raspberry Pi to the PLC, numerous "connection refused" issues were encountered, even if the subnets were aligned. This led to further investigation of the details of the hardware configuration of the PLC. The student did some investigation and came to understand that the CPU config for the problem of the PLC was the one at fault. More precisely, the PLC's PUT/GET communication setting had to be set to ON. Therefore, this was not on, which made it impossible for any connections from the Raspberry Pi. On the other hand, whitelisted communication allows external devices to the CPU as long as it is not prevented.

After the enabling of PUT/GET connectivity, there was still the Raspberry Pi which was okay with connecting to the PLC, but more trouble was there waiting—the CPU was unresponsive when it came to the PLC's data block writing. The course of action that included going through the available documentation that Siemens made public and browsing through the stack overflow,

the student came to understand that the reason why the problem occurred is due to there being set as 'optimized data block.' These optimized data blocks allow access to the Siemens PLCs to be easy to store but renders other applications like Snap7 not able to interact as much. Deactivating this optimization permitted the student to write to the data in the memory of the PLC enabling

the Raspberry Pi to transmit the classification.

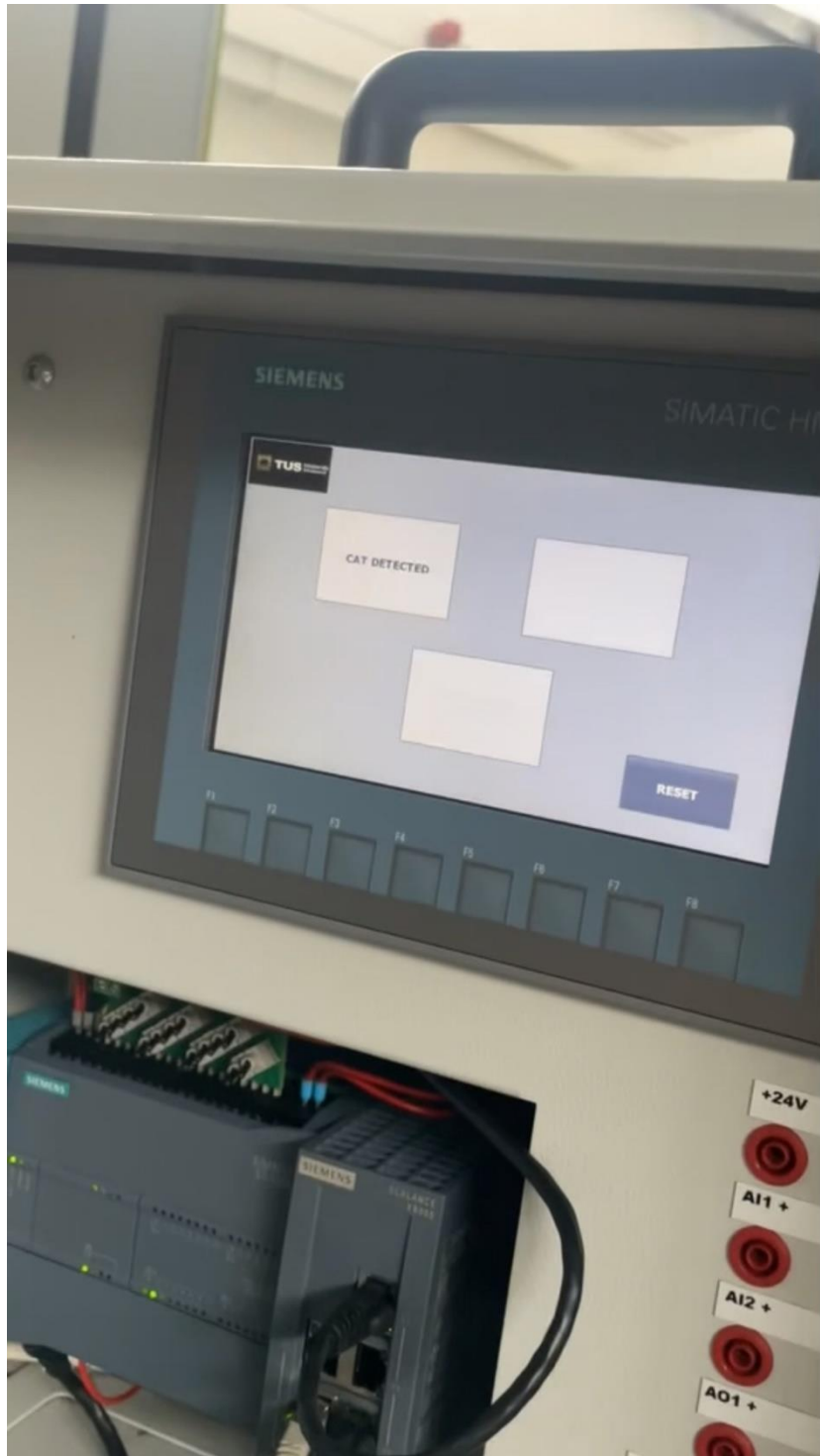


Figure 6 Raspberry Pi Comms

#### 9.4 Refining the Communication Process

Improvements had to be made on the communication and procedure, after the problem of the data block was fixed and the connection established. At this time, the student had a task of taking the Python scripts a little further to ensure that the data transmission between the PLC and Raspberry Pi is accurate. When this was done, the foundation for the real wiring for the project was ready.

## 10 Wiring and Configuring the System

When the Raspberry Pi and Siemens PLC started communicating, the next thing to do was to wire the components of the system. Some of the significant processes that were performed include the wiring of the electrical system. These are further below.

### 10.1 Initial Setup and Powering the PLC

The first task involved the installation of the electrical parts required to energize the PLC and other systems. Thomas supplied the student with the necessary parts including 24V power supply unit (PSU), Siemens S71200 1215 DC DC DC and 4a MCB. The first two actions to carry out before the start of the wiring process were to connect the 24V PSU to the mains and then the PLC. This one was necessary to switch the PLC on, and also connect the laptop and the Pi in an online mode.

As soon as the rig was powered up, it was the flashing of the PLC's error light which was noted. In order to help address this problem, the student used the laptop to connect to the PLC. It was found that there were wrong hardware configurations in the PLC and a full reset of configurations was required. It was also important to PUT/GET communication since it was necessary for the Raspberry Pi to interact with the PLC. After these configurations were appropriately done, communication was successful between the Raspberry Pi PC and the PLC

and this was a great achievement for the project.



*Figure 7 Initial Wiring of PLC*

## 10.2 Wiring the Conveyor Belt and Addressing Issues

Once the communication was confirmed, the next critical job was to wire the single phase 240v conveyor belt's servo motor. Firstly, there was a need to control the direction of movement of the conveyor belt remotely using a contactor. The contactor was intended to close a circuit that would allow 240V flow to the motor when the PLC sends a signal that will measure 24V. Still, a crisis situation arose when it was found out that the contactor was burnt out due to unforeseen reasons, this information surfaced rather late in the course of the project. Since the contactor failed, there were of course problems starting the conveyor system.

To remove the issue at hand, the student resolved to adapt an Altivair 11 and connect its AC power source to the mains supply through the contactor. Some questions remained regarding proper installation of the VSD which prompted research for clarification regarding various VSD wiring processes. This involved perusing through various datasheets as well as video clips that explained how and where the wires go.

The operation included mains power supply execution to a 10a MCB then to the VSD as well as practical installation of a speed dial to control the speed of the motor and work with appropriate external circuits of the VSD. Several cabling set-ups were done at the completion stage so that power could be brought to the control circuit and the motor. And the last thing that was installed was all the electrical wiring of the conveyor unit.



### 10.3 Finalizing Initial Wiring and Testing

The attention was on finalizing the installation of the wires for other components, with the conveyor belt working. This involved the further installation of start, stop, reset, and emergency stop (estop) functionalities. In the beginning, the estop was set so that the PLC would stop the conveyor belt before any power cut off, rather than cut off power altogether, to simplify the control of the system. This would later be changed.

The next major component to be wired in was the Cognex InSight 7801 camera. The reason for this addition was to take pictures of items on the conveyor belt and relayed to the Raspberry Pi for further processing.



Figure 8 Implementation of VSD



## 11 Integrating and Configuring the Cognex INSIGHT 7801 Camera

After the system's initial wiring was finished, the Cognex INSIGHT 7801 camera setup was an essential next step. This camera is needed to take pictures of the parts on the conveyor belt and transfer them to the Raspberry Pi for processing. Attention to detail was necessary throughout the process to guarantee correct image acquisition and appropriate functionality.

### 11.1 Initial Setup and Wiring of the Camera

The breakout cable of the camera was first connected to a 24v power source during setup. The data sheet indicated that the breakout wire was color-coded, with black being 0V and red being 24V. Nevertheless, when connecting the wires to a power supply using crocodile clips, the camera did not turn on as planned.

The student discovered a difference between the colour codes on the breakout cable and the data sheet after conducting additional research. The wire numbers were shown by a key on the cable, and a comparison showed that the colours were different from what was listed on the data sheet. For example, wire number 7 on the breakout cable was black, even though the data sheet said it was red (for power). The following connections were found to be accurate:

- **Power (24V):** Black wire
- **Ground (0V):** Gray wire

Once the connections were adjusted and rewired, the camera was able to power on.



*Figure 9 Cognex 7801 Power*

## 11.2 Connecting the Camera to the Laptop

After turning on the camera, the next steps involved connecting it to the laptop. For this, an Ethernet cable was used. Using VMware the student used Insight Explorer 5.6.0 to gain access to the camera.

For the connection to work, the camera's subnet needed to match the subnets of the laptop, Raspberry Pi, and PLC. To make sure the camera, the laptop, Pi and PLC were on the same network, this required modifying the subnet settings on both devices. When these settings aligned successfully, all devices were able to communicate with one other.

## 11.3 Camera Configuration and Mounting

After the camera was brought online, the aim turned to setting it up both physically and operationally. The rig held the camera firmly in place, and the panel received direct power and ground connections (24V).

The next stage was to adjust the camera's settings to take better pictures:

- **Focusing:** The camera was adjusted to ensure a clear view of the parts.
- **Light Intensity:** The lighting was changed to give proper lighting for accurate image acquisition.
- **Exposure:** The exposure settings were fine-tuned to balance image brightness and detail.

These modifications were necessary to make sure the camera consistently and clearly caught images.

## 11.4 Camera Triggering and Image Capture

At first, the student tried to turn on the camera by connecting one of the breakout lines (trigger) to an output from the PLC. The setup did not function as planned. As a result, a different strategy was put into place using the camera's internal software.

The camera was set up to fire off every 300 milliseconds constantly. Regular acquisition of photos of the parts moving on the conveyor belt was guaranteed by this frequent triggering. The student used the software's "Patterns Max" tool to establish when a part was positioned correctly. This function was trained to identify parts features, like the visible block section.

The camera displayed a "pass" result when it found a block segment in the field of view and determined that it met the pattern recognition requirements. The procedure of taking the picture and forwarding it for additional examination was started by this outcome. After that, the image data was delivered to the Raspberry Pi for processing, which was necessary for the system's general operation.



*Figure 10 Cognex camera set up and wired in*

## 12 Integrating Camera Image Transfer with FTP and Training the Algorithm

The next major task was to enable the Cognex INSIGHT 7801 camera to communicate photos of detected parts to the Raspberry Pi via FTP after it had been successfully wired and setup. The machine learning-based classification system's smooth functioning depended heavily on this stage.

### 12.1 Setting Up FTP Image Transfer

The student used VMware on a laptop without an Ethernet port instead the laptop was connected using an Ethernet extension cable, and it was possible to direct the connection to the virtual machine or the host PC. The virtual machine was the destination of the connection in this configuration. Within the Insight program, an FTP job was configured to start when the Pattern Max function identified a pass.

Nevertheless, there were a few difficulties along the way. Many errors were produced at first because the FTP server could not establish a connection with the Raspberry Pi. After extensive investigation, it was determined that the problem between the VMware configuration and the FTP server's subnets was the root of the problem. Communication problems were caused by the VMware's subnet not being compatible with the FTP server's subnet.

The student moved and configured the FileZilla FTP server software within the VMware environment to fix issue. VMware and the Raspberry Pi were able to communicate directly by assigning the FTP server IP directly, as opposed to utilizing the Raspberry Pi's name. The image could be successfully transferred over FTP with this arrangement.



## 12.2 Handling Multiple Images and FTP Queue Management

When it was working, every 300 milliseconds the camera was told to fire and take a lot of pictures transmitting those for every movable element on the assembly line. Shot like this routinely creating a file on the FTP site created a problem of multiple duplication of the photos. To counter this problem, some adjustments were made:

1. **Configuration Adjustments:** The student changed the speed of the conveyor belt in relation to the speed of the pictures taken too in order to minimize the number of pictures sent.
2. **FTP Queuing:** Multiple pictures had to be taken of a single part too many times, queuing images for transfer was prevented by disabling ftp queuing.
3. **Single Image Capture:** The systems configuration only allowed one picture of each part to be transferred to ensure no unnecessary data was processed and that server congestion was avoided.

Using these means, it was assured that only one picture of each part was received by the Raspberry Pi and the FTP server would not have excess demands placed upon it.

## 12.3 Training the Machine Learning Algorithm

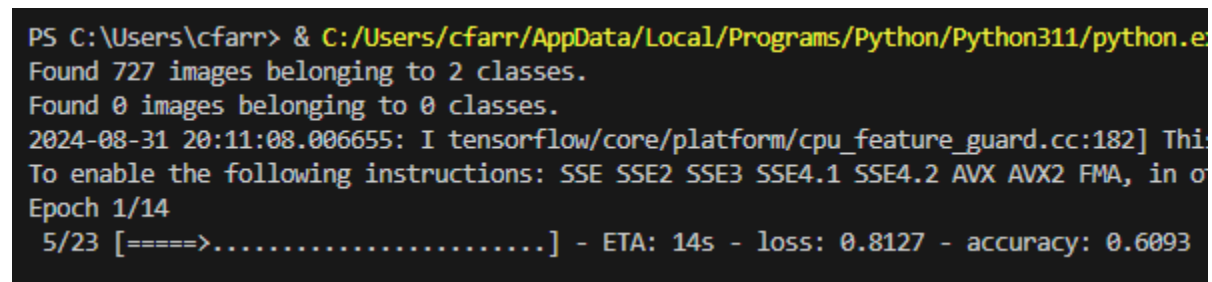
Once the image transmission difficulties were addressed, the focus of training the machine was laid solely upon training the machine learning algorithm. There were many important steps throughout the entire procedure:

1. **Image Collection:** Camera setup allowed to photograph the parts moving on the conveyor belt. The purpose was to take clear pictures of the parts in different positions:
  - **Pass Parts:** Collection of 300 parts with a visible tick identified as with a pass was made.
  - **Fail Parts:** Special images of the parts were taken such that they were turned sideways- upside down and in the standing position. This way any part which had visible traces of defect such as 'X' or incorrect angular position was considered a fail.
2. **Image Preparation:** The images were uploaded to FTP server then shift to Raspberry Pi. This dataset consisted of samples with different angles and positions of the scanned components within the enclosure.
3. **Algorithm Training:** The photograph collected was used to instil and train the machine learning system. It took approximately 15 minutes to finish this particular step when the algorithm was shown pictures of pass and fail conditions. The training intended to refine the existing image processing algorithm for real-time application in component sorting.
4. **Testing and Validation:** In order to test the accuracy and the reliability of the algorithm in the practical situation, extensive testing was conducted at the end of training using real time image data. At this stage, the effectiveness of the algorithm in classifying components as per the training parameters was tested.



## 12.4 Summary

Within the featured approach, it has to be noted that there were a few technical challenges connected to the process of integrating and configuring the camera Cognex Insight 7801 for the purposes of transferring the obtained pictures via FTP. The student managed resolve these issues and carry on.



```
PS C:\Users\cfarr> & C:/Users/cfarr/AppData/Local/Programs/Python/Python311/python.e
Found 727 images belonging to 2 classes.
Found 0 images belonging to 0 classes.
2024-08-31 20:11:08.006655: I tensorflow/core/platform/cpu_feature_guard.cc:182] This
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in o
Epoch 1/14
5/23 [====>.....] - ETA: 14s - loss: 0.8127 - accuracy: 0.6093
```

Figure 11 Training terminal demonstration

## 13 Implementation of Safety Features and System Integration

Installing safety features and making sure the system worked was important after setting up the wiring different parts. This was done by making sure the system was tested for safety and as well as integrating an Emergency Stop (E-Stop) relay.

### 13.1 Implementing Safety Features

#### 13.1.1 Emergency Stop (E-Stop) Relay:

1. **Purpose:** To offer a physical safety mechanism that, in an emergency, cuts power to parts, including the Variable Speed Drive (VSD), system shutdown.
2. **Setup:** A physical E-Stop relay was included in place using the PLC to operate E-Stop software. When this relay was triggered, the VSD's power would be cut off, stopping all right away.
3. **Components:**
  - **E-Stop Relay:** Chosen based on documentation from the manufacturer, ensuring it matched the required specifications.
  - **MCBs (Miniature Circuit Breakers):** Two MCBs were installed:
    - **4A MCB:** For the power supply.
    - **10A MCB:** For the main supply to the contactor and subsequently to the VSD.

4. **Wiring and Configuration:** Wiring and Configuration: When the relay was activated, it was wired to cut off all power to the VSD. To wire and configure the relay appropriately, the student consulted the manufacturer's literature.
5. **Testing:** To make sure the E-Stop feature cut power and stopped the system as intended, it underwent testing.

#### 13.1.2 JSBR4 E-Stop Relay into the Automated System

This section focuses on how the system intended for conveyor belt control and part classification uses the JSBR4 Emergency Stop (E-Stop) relay. The JSBR4 relay is used to maintaining safety since it offers a physical way to physically stop the system in an emergency. This section describes the steps involved in choosing, wiring, and testing the JSBR4 relay and shows how crucial it is to create a safe system.

##### *13.1.2.1 Selection of the JSBR4 E-Stop Relay*

The JSBR4 was chosen for the system because of its safety features. When this relay is triggered, it is intended to cut off power to parts, like the variable speed drive (VSD). The JSBR4 has the following important features:

- **Contact Configuration:** The relay has normally closed (NC) contacts, meaning that power flows through the relay under normal operation. When the E-Stop button is pressed, these contacts open, stopping power flow.
- **Electrical Ratings:** The JSBR4 is rated to handle 240v and 1.5A current and voltage loads required by the VSD and other components in the system.

- **Safety Compliance:** The relay complies with industrial safety standards, providing a reliable means of emergency shutdown.

#### *13.1.2.2 Wiring and Installation*

##### **Power Supply Connections:**

The JSBR4 relay requires a power supply, typically 24V DC. The power supply lines are connected to the coil terminals of the relay, A1 for 24v+ and A2 for 24v-.

##### **Control Circuit Integration:**

The E-Stop button is wired to the relay's control circuit (S13,14 & S24,23). When pressed, the relay's internal contacts open, cutting power to the VSD. The reset button is wired to X2 & X3 as well as S23 is wired to X3 to allow the relay to be reset when the estop is pressed.

**Wiring Diagram:** The E-Stop button is connected to the relay's coil, activating it to open the normally closed contacts when engaged.

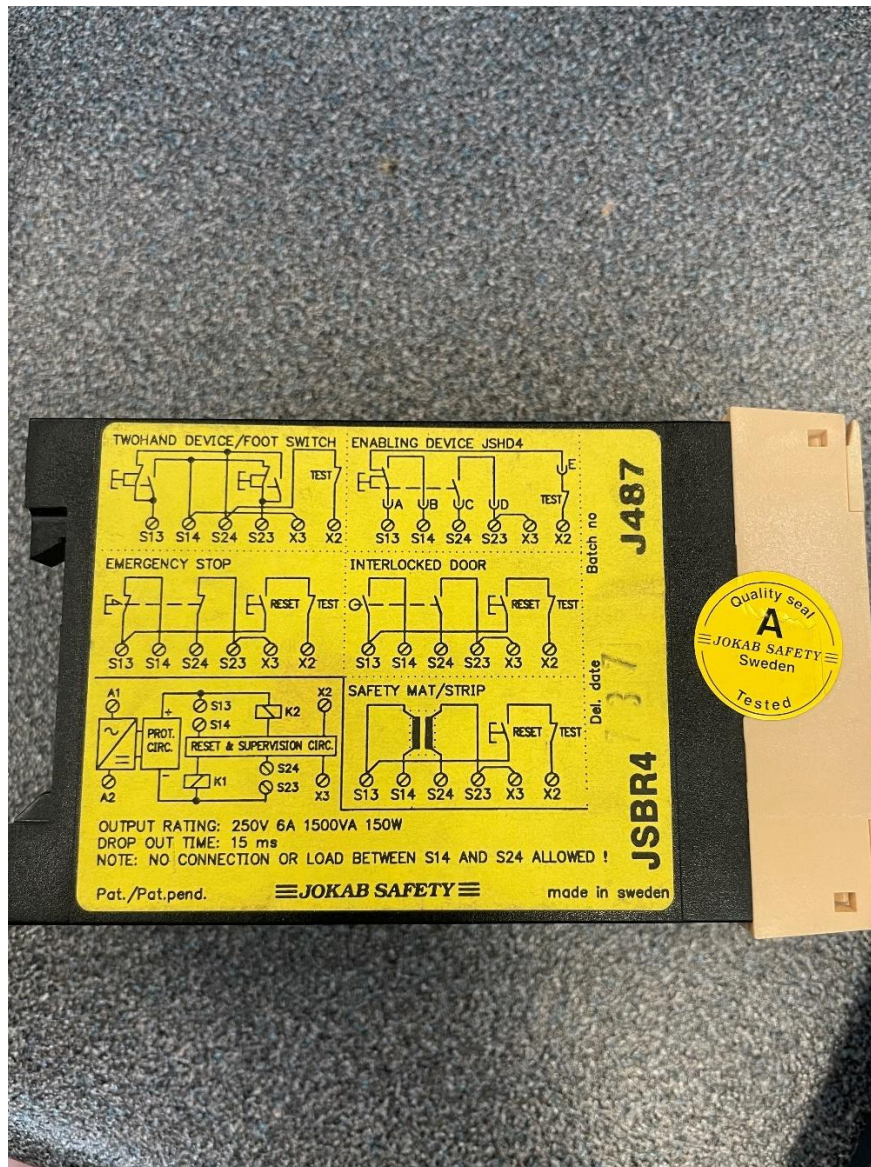


Figure 12 Estop Relay

### Power Circuit Connections:

The NC contacts of the relays are connected in series to the power input of the VSD. This arrangement is to ensure that the moment the relay is energized, all voltage to the VSD is eliminated.

### 13.1.3 Incorporation of Miniature Circuit Breakers (MCBs):

There is an MCB with 4A ratings for the power supply and another MCB of 10A ratings meant for the primary supply to the contactor and the VSD. These breakers, in preventing overcurrent and short circuiting, ensure the safe functioning of the relay.

#### *13.1.3.1 Functionality Testing:*

Once the JSBR4 relay is wired up, the E-Stop button is also found to effectively cut power to the VSD. This includes pressing the button and confirming that every component that was turned on is turned off within seconds.

Also, tests are done with the power restored immediately after the E-Stop button is released and the reset button is pressed.

#### *13.1.3.2 Safety Verification:*

Comprehensive tests are performed so that the relay operates reliably in case of an emergency. As a part of this, the situations of emergencies are created, and the system is tested to fall as it was designed to do. Wires are therefore inspected to ensure there are no loosed connections so as to avoid false bearing or failure.

#### *13.1.3.3 Documentation and Compliance:*

The documentation regarding the manufacturing process of the JSBR4 relay is scrutinized in order to ensure that the right wiring practice and safety measures are followed. Some warning regarding the safety measures taken and external & internal connections are mentioned.

#### 13.1.4 Panel and Wiring Organization:

1. **Panel Wiring:** To refrain from the fall of any disconnections, all wires and connections were made sure to be painfully jacked in. The enclosure was cleaned perfectly to ensure that there is no dust for safety and clear view.
2. **Wiring Tidiness:** To prevent ever reaching that degree of muddle and to facilitate the comprehension of the panel, wires were organized and tied using zip. This was important during both debugging and service.
3. **Pass and Fail Indicators:** Pass and fail lights were added in to other wires for light indicators. The indicators which provided a pass light on objects which have been positively classified by the PLC and the failure light on objects which a wrongly classified, illustrated the reaction of the PLC perfectly.





Figure 13 Final Panel Fully Wired



## 14 System Integration and Final Testing

### 14.1.1 System Workflow:

*Component Integration:* Integrated system included conveyor belt, camera, PLC, and raspberry pi. Parts were placed on the conveyor belt and a sensor was triggered to manage the stream of parts.

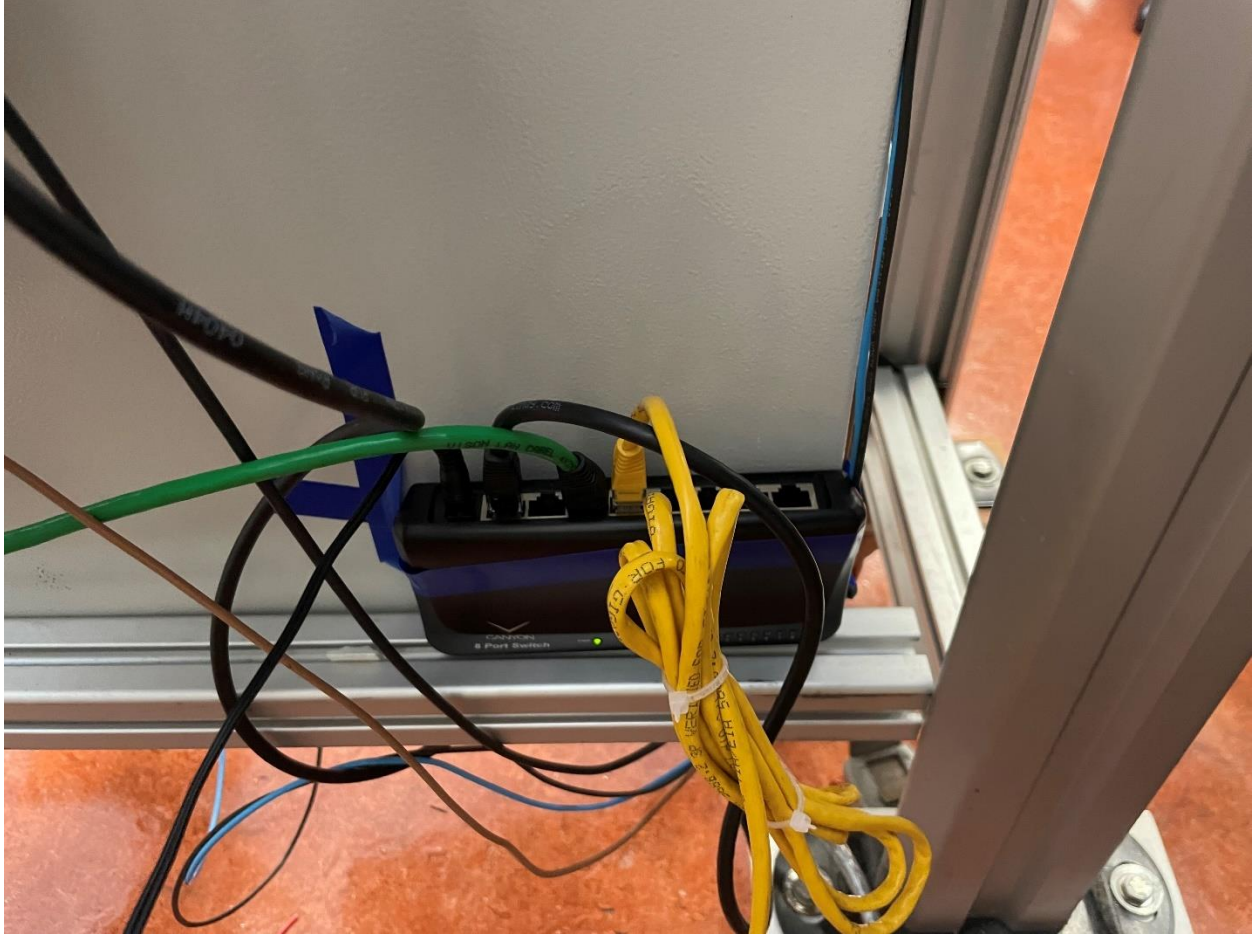
#### 14.1.1.1 Image Capture and Processing:

- **Triggering:** This also involved the use of a camera that took pictures of the parts moving beneath it every 300 milliseconds.
- **Image Transfer:** The problem of retrieving images from an FTP server on a RaspberryPi was solved by first sending images to an FTP server where they were later collected by the Raspberry Pi.

#### 14.1.1.2 Image Processing and Classification:

- **Code Functionality:** Processing of images was done on code present in the Raspberry Pi. It involved multiple important sub processes which include:
  - **Image Transfer:** On the Raspberry Pi, images that were processed from an FTP folder, were moved to a processing folder.
  - **Image Processing:** A trained algorithm on the TensorFlow/Keras model sorted the images into ‘pass’ and ‘fail’ categories.

- **Result Reporting:** The results of the classification were sent to the PLC, which controlled the conveyor belt and the pass and fail lights.



*Figure 14 Network Switch for System*

## 15 Detailed Breakdown of the Python Code for Image Processing and PLC Communication

### 15.1.1 Overview

The given Python code is responsible for images processing within the limits of a TensorFlow/Keras model, while providing a bidirectional interaction between the Raspberry Pi, FTP server and Siemens PLC. This section provides details on the code and system where it highlights the role of each part and the relative location in the system.

### 15.1.2 Importing Libraries

```
import tensorflow as tf

from tensorflow import keras

from PIL import Image, ImageFile

import os

import shutil

import time

import snap7

from snap7.util import set_bool

from snap7.type import Areas
```

- **TensorFlow/Keras:** Involves loading and making predictions using the already trained deep learning model.
- **PIL (Pillow):** For image processing and verification.
- **os:** To work with the OS file structure.
- **shutil:** To handle files considering their hierarchy.
- **time:** To introduce waiting periods within the processing cycle.
- **snap7:** For connection with Siemens PLC using s7 communication protocol.

### 15.1.3 Model Loading

```
model = keras.models.load_model('FYP_test1.h5')
```

- **`keras.models.load\_model()`:** Recovers already trained TensorFlow/Keras reshaped in a file provided as a parameter ({FYP\_iteration1.h5}). This model is further used to classifies images to either “pass” or “fail”.

### 15.1.4 Directory Setup

```
ftp_folder = '/home/ftpuser'
```

```
input_folder = '/home/pi/Desktop/fyp_pi_code/fyp_realtime/images'
```

```
processed_folder = '/home/pi/Desktop/fyp_pi_code/fyp_realtime/scanned'
```

- **`ftp\_folder`:** Folder, in which the uploaded photographs should be stored by the FTP server.

- **`input\_folder`**: A folder, into which images are relocated in order to be processed.
- **`processed\_folder`**: A folder containing images that have already been classified for further processing or archiving.

#### 15.1.5 PLC Communication Setup

```
PLC_IP = '192.168.0.1' # PLC's IP address

client = snap7.client.Client()

try:

    client.connect(PLC_IP, 0, 1)

    if client.get_connected():

        print("Connected to PLC")

    else:

        print("Failed to connect to PLC")

        client = None

except Exception as e:

    print(f"Failed to connect to PLC: {e}")

    client = None
```

- **`snap7.client.Client()`**: Creates a new instance of the Snap7 client.
- **`client.connect()`**: Using the PLC's IP address to establish a connection. The DB slot numbers are denoted by the parameters {0} and {1}, respectively.

#### 15.1.6 File Readiness Check

```
def is_file_ready(file_path):  
  
    initial_size = os.path.getsize(file_path)  
  
    time.sleep(0.5) # Short delay to ensure file writing is complete  
  
    return initial_size == os.path.getsize(file_path)
```

- **`is\_file\_ready()`**: compares the file size before and after a brief wait to make sure it is fully written and prepared for processing.

#### 15.1.7 Image Processing

```
def process_image(image_path):  
  
    retries = 3 # Number of retries  
  
    for attempt in range(retries):  
  
        try:  
  
            # Check if the file is ready  
  
            if not is_file_ready(image_path):  
  
                raise IOError("File is not ready for processing.")  
  
            # Open and verify the image  
  
            with open(image_path, 'rb') as f:
```

```
img = Image.open(f)

img.verify() # Verify that the image is complete

# Reopen the image for processing after verification

img = keras.utils.load_img(image_path, target_size=(224, 224))

img_array = keras.utils.img_to_array(img)

img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)

confidence = predictions[0][0]

# Determine pass or fail based on the confidence

if confidence > threshold:

    return 'pass', confidence

else:

    return 'fail', confidence

except (IOError, tf.errors.InvalidArgumentError,
Image.DecompressionBombError) as e:

    print(f"Error processing image {image_path}: {e}")

    if attempt < retries - 1:

        print("Retrying...")

        time.sleep(0.5) # Short delay before retrying

    else:

        return 'error', 0
```

```
return 'error', 0
```

- **`process\_image()`**: Manages the entire image classification and verification process.
- **Image Verification**: Verifies the integrity and completeness of the picture file.
- **Image Processing**: Resizes the picture, turns it into an array, and makes predictions about the result using the TensorFlow/Keras model.
- **Error Handling**: If an error occurs (such as a file not being ready or problems with picture decompression), try the process again.

#### 15.1.8 Sending Results to PLC

```
def send_result_to_plc(filename, category, confidence):  
  
    if client and client.get_connected():  
  
        print(f"Sending '{category}' result for {filename} with confidence  
{confidence} to PLC")  
  
        try:  
  
            # Read the existing data from DB1 (1 byte, for example)  
  
            db_number = 1  
  
            start_address = 0  
  
            size = 1  
  
            data = client.read_area(Areas.DB, db_number, start_address, size)  
  
            # Clear previous bits
```



```
    set_bool(data, 0, 0, False) # Clear pass_detected

    set_bool(data, 0, 1, False) # Clear fail_detected

    # Set the corresponding bit based on category

    if category == 'pass':

        set_bool(data, 0, 0, True) # Set pass_detected

    elif category == 'fail':

        set_bool(data, 0, 1, True) # Set fail_detected

    # Write the modified data back to the PLC

    client.write_area(Areas.DB, db_number, start_address, data)

except Exception as e:

    print(f"Error sending data to PLC: {e}")

else:

    print(f"Failed to connect to PLC. File: {filename}, Category: {category},
Confidence: {confidence}")
```

- **send\_result\_to\_plc()**: Sends the classification result to the PLC.

- **Data Handling**: Reads the current data block, modifies the specific bits for pass/fail results, and writes it back to the PLC.

- **Error Handling**: Catches and reports errors related to PLC communication.

### 15.1.9 Main Loop for Monitoring and Processing Images

```
while True:

    found_new_image = False

    # Move new images from FTP folder to the input folder

    for filename in os.listdir(ftp_folder):

        if filename.endswith(".jpg") or filename.endswith(".jpeg") or
filename.endswith(".png"):

            ftp_image_path = os.path.join(ftp_folder, filename)

            dest_image_path = os.path.join(input_folder, filename)

            try:

                shutil.move(ftp_image_path, dest_image_path)

                print(f"Moved {filename} from FTP folder to processing folder")

                images_to_process.append(filename)

                found_new_image = True

            except Exception as e:

                print(f"Failed to move {filename} from FTP folder to processing
folder: {e}")

    # Process images in the list

    for filename in images_to_process:

        img_path = os.path.join(input_folder, filename)

        if os.path.exists(img_path): # Ensure image still exists in input_folder
```

```
category, confidence = process_image(img_path)

print(f"File: {filename}, Classified as: {category}, Confidence:
{confidence}")

send_result_to_plc(filename, category, confidence)

# Move processed image to the processed folder

shutil.move(img_path, os.path.join(processed_folder, filename))

# Remove from images_to_process to avoid re-processing

images_to_process.remove(filename)

if found_new_image:

    time.sleep(0.1) # Wait until next image appears

else:

    time.sleep(0.1) # Adjust sleep time if needed for faster response
```

**-Image Handling:** Transfers fresh pictures to the input folder for processing from the FTP folder.

**-Image Processing and Classification:** Each image is processed, then it is classified, the findings are sent to the PLC, and the processed images are moved to the scanned folder.

**Loop and Delay:** Processes photos and keeps an eye on the FTP folder continuously, pausing briefly to adjust the rate of image processing.

## 15.1.10 Summary

The provided code solves the problem of integrating PLC communication with that of image categorization in full. It provides almost real-time image processing and classification, which communicates the results to the PLC for action. Such a comprehensive analysis involves putting together all the features and explaining them individually and to give the readers an overview of how each of the elements contributes towards the performance of the system.

## 16 Detailed Breakdown of the PLC Program

### 16.1 PLC Program Overview

The PLC software, which is as the system's CPU, handles inputs from start/stop buttons, controlling the conveyor motor, and reacting to signals from the Raspberry Pi and sensor. Within the PLC, the program is organized into several networks, each with a specific function. The functioning of each PLC program component is described in depth in the sections that follow.

#### 16.1.1 1. Initial Setup and Memory Bit Management

##### *16.1.1.1 Network 1: Start Button and Memory Bit Initialization*

The first network is intended to manage the early startup circumstances of the system. Several important processes are started when the operator hits the start button:

- **Conveyor Running Memory Bit:** When the conveyor is in operation, a memory bit (M) is set to show it.
- **Run On Memory Bit:** A different memory bit is configured to turn on the run light, which serves as a visual cue that the system is on.
- **Stop On Memory Bit Reset:** To make sure that the stop light is off while the system is operating, the memory bit linked to the stop condition is reset simultaneously.

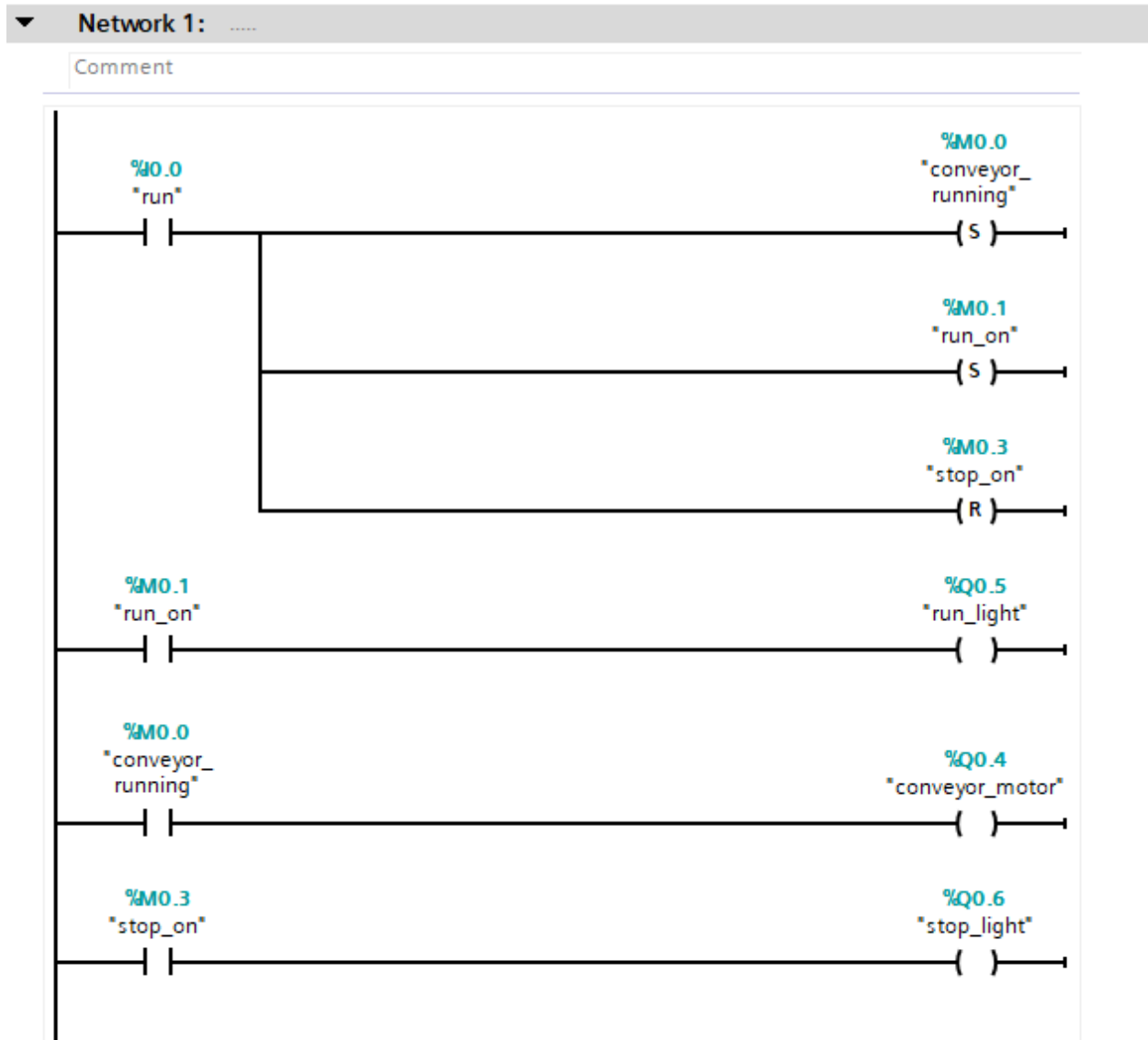


Figure 15 PLC Network 1

#### 16.1.1.2 Network 2: Stop Button Logic

When the stop button is pressed, the logic is controlled by the second network. This network is essential for stopping the system safely:

- The input for the start button is set to normally closed (NC). Pressing the stop button causes the following to happen:

- The conveyor motor is essentially stopped when the Run On Memory Bit and Conveyor Running Memory Bit are reset.
- After that, the stop light is turned on to confirm visually that the system is off. This is done by setting the Stop On Memory Bit.

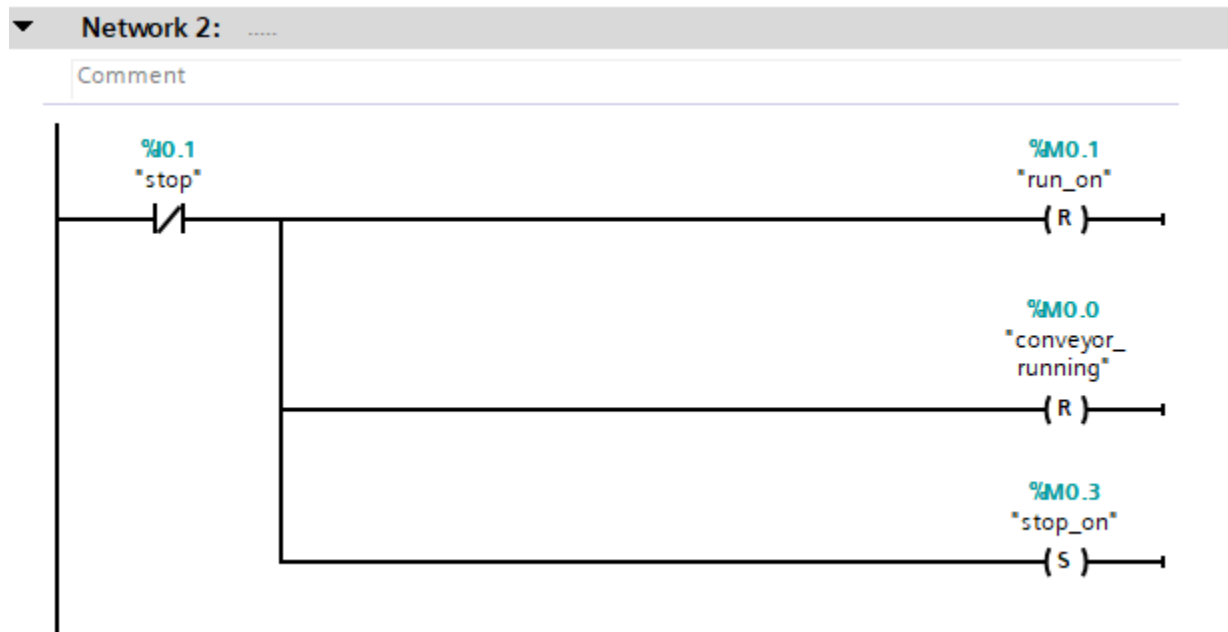


Figure 16 PLC Network 2

### 16.1.2 2. Sensor Integration and Conveyor Control

#### 16.1.2.1 Network 3: Sensor Input Handling

- The third network looks at reacting to data received from the conveyor-side sensor:
  - The Conveyor Running Memory Bit is reset when the sensor is activated, signalling the presence of a part on the conveyor.
- This network is used for coordinating the movement of the conveyor with the inspection process.

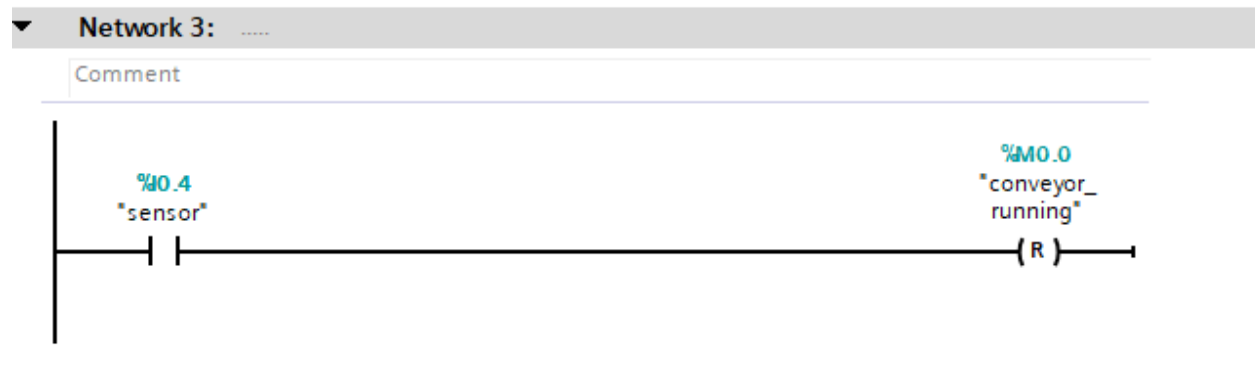


Figure 17 PLC Network 3

### 16.1.3 3. Communication with Raspberry Pi and Result Handling

#### 16.1.3.1 Network 4: Pass Detection and Conveyor Operation

- The fourth network manages comms between the Raspberry Pi and the PLC, in cases where a component is labelled as "pass":
  - **DB Pass Detected:** This data block is set when the Raspberry Pi sends a signal indicating that the component has passed inspection following image analysis.
  - If the **Stop On Memory Bit** has not been activated, the following occurs:
    - A timer is used to trigger the Pass Output, which turns on the pass light for a set amount of time (two seconds).
- The DB Pass Detected data block is reset following the activation of the pass output. The data block is latched; hence this reset is required.



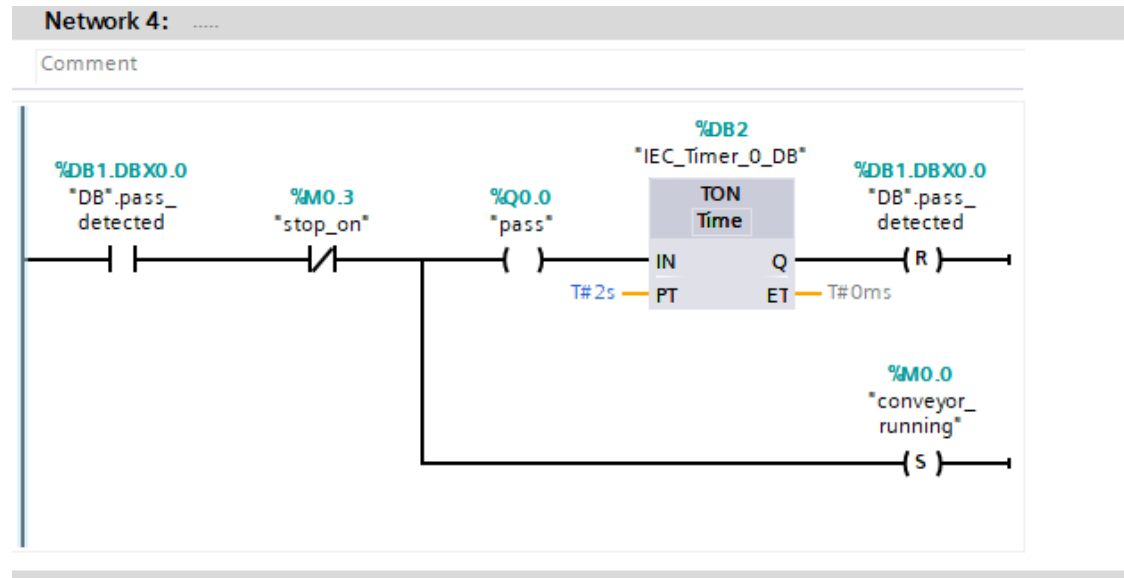


Figure 18 PLC Network 4

#### 16.1.3.2 Network 5: Fail Detection

- The fifth network mirrors the logic of Network 4 but is activated when the Raspberry Pi detects that a part has failed inspection:

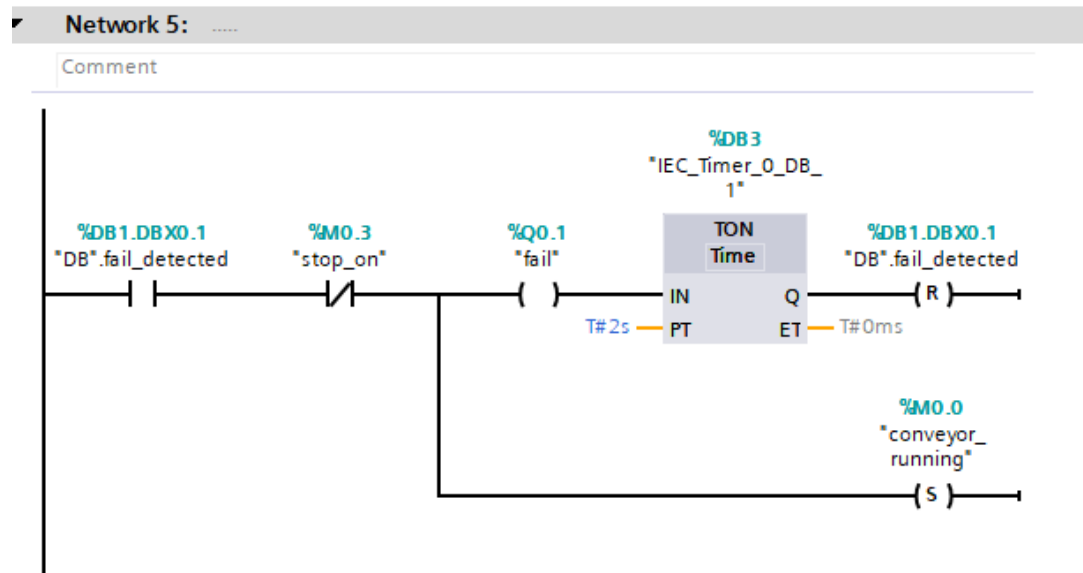


Figure 19 PLC Network 5

## 17 Cognex In-Sight Software Configuration and Troubleshooting

Through its processing of images from the camera positioned above the conveyor belt, the Cognex In-Sight software was important in the system's operation. The objective of the setup was to set up the Patterns Max tool, which oversaw identifying and learning patterns on the parts that were being examined. When an object moved in front of the camera, the program used the Patterns Max tool to determine if it satisfied the requirements for a "pass."

### 17.1 Configuration of the Patterns Max Tool

The Patterns Max tool was taught to identify the proper part orientation and marks. The tool would recognize a part as a pass when it was properly positioned beneath the camera. The software was set up to transfer the part image to a specified location on the Raspberry Pi, specifically at IP address 192.168.0.100, as soon as a pass was detected via an FTP server.

### 17.2 FTP Server Setup and Troubleshooting

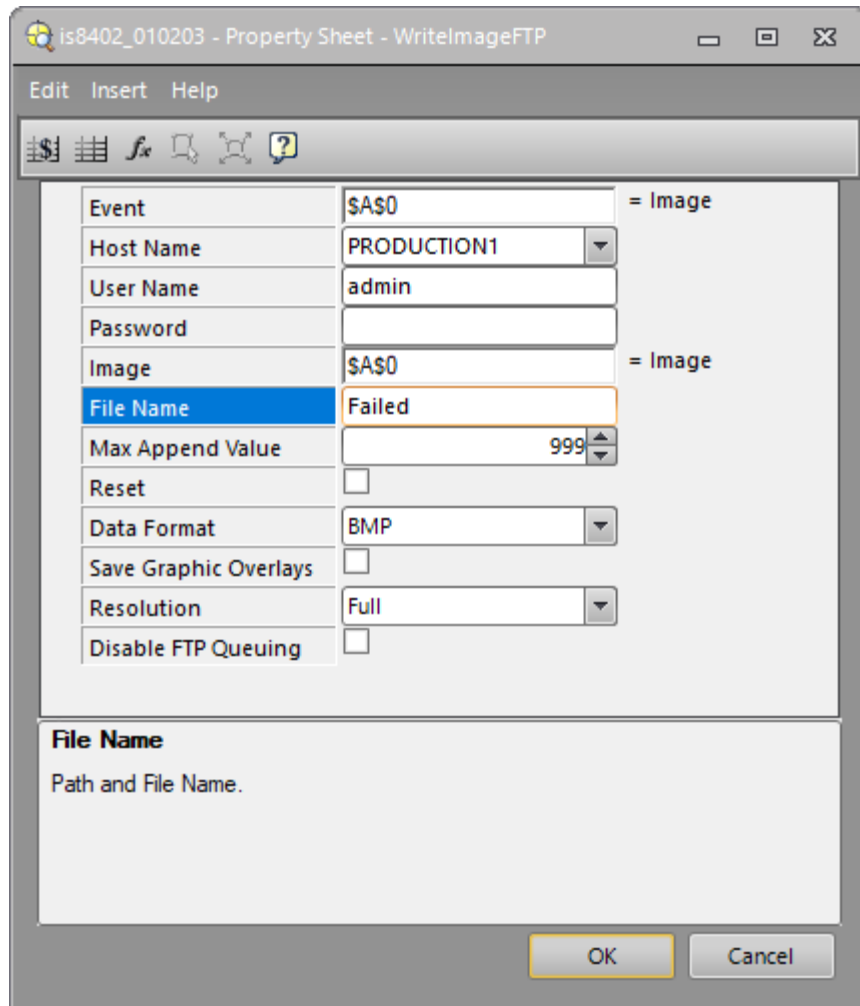
One of the trickiest parts of the project was configuring the FTP server inside the In-Sight program. At first glance, setting up the FTP server seemed simple, but the program had a lot of trouble getting pictures to the Raspberry Pi. Several steps were included in the troubleshooting process:

1. **Initial Setup Failures:** The In-Sight program kept failing to connect to the Raspberry Pi, even after the FTP settings were correctly configured. There were several attempts to reconfigure the server, but this problem remained, which resulted in long delays.

2. **Network Configuration:** Following a thorough debugging process, it was determined that the network configuration was the source of the issue. Successful communication was not possible since the virtual machine (VMware) running the Cognex software, and the Raspberry Pi were not on the same network.
3. **FileZilla and VMware Adjustment:** FileZilla was moved to the VMware environment to fix this. The FTP connection was eventually established by rearranging the subnets so that the Raspberry Pi and the virtual machine were connected to the same network. Ensuring that photos could be sent without network interruption required the completion of this crucial phase.
4. **Handling Multiple Image Transfers:** After the FTP connection was established, a fresh problem emerged: each trigger was causing the sending of numerous photos. Every 300 milliseconds, the camera would take and send an image, flooding the system with unnecessary information. By changing the FTP task settings in the In-Sight program, this issue was fixed. To make sure that the Patterns Max tool only sends one image every time a pass was found, the settings were changed.

#### 17.2.1 Final Implementation

These difficulties were eventually resolved, allowing the In-Sight program to connect with the Raspberry Pi. To guarantee that the Raspberry Pi's machine learning algorithm analysed only pertinent data, the system was adjusted to send a single image per detection. The achievement of a fully integrated and operational automated inspection system was largely dependent on this exact arrangement and the thorough troubleshooting procedure. (Cognex, n.d.)

*Figure 20 FTP job*

## 18 Risk Assessment and Health & Safety Considerations

### 18.1 Risk Assessment

There were several risks identified during the development and implementation of the automated inspection system mainly the use of mechanical parts, the software and electrical safety. The main dangers were as follows:

#### *18.1.1.1 Electrical Shock and Burns:*

- **Risk:** There was a possibility of sustaining electrical shock when handling parts associated to the Variable Speed Drive (VSD) and the 24V electrical connections.
- **Mitigation:** Proper precautionary measures had to be taken including the use of the right protective gear, insulating appropriate equipment, and ensuring that power outlets were turned off before doing any electrical wiring. There was also a provision to add an E-stop relay such that the power to the Variable speed drive (VSD) would be cut off immediately to prevent further usage in the event of an emergency.

#### *18.1.1.2 Mechanical Hazards:*

- **Risk:** The collection chamber and the conveyor belt had moving parts and there was a risk of getting jammed or hurt with these parts.
- **Mitigation:** Electrical emergencies stop buttons were fixed to the conveyor and other peripherals in order to end the operations within a fraction of second in case of an emergency. Other devices such as guards were fixed around most of the

moving parts to prevent injury to their users. There were other ways to reduce risk of injury that were also effective. One of them was to control the maximum allowable speed of the conveyor belt.

#### *18.1.1.3 Software Failures:*

- **Risk:** The image processing technique or the connection among the Raspberry Pi, PLC, and the camera can be deficient, resulting in erroneous sorting or system failures.
- **Mitigation:** The software has built in measures for thorough testing and treatment of mistakes.

## 19 Health & Safety Considerations

Safety matters were incorporated in every aspect of design and implementation of the project.

The other notable aspect to be considered was the Emergency Stop (E-Stop) integration.

### 19.1 Emergency Stop (E-Stop) Integration:

- There was a unique E-stop relay system (JSBR4) which was set such that once turned on the power to the VSD is switched off immediately stopping further operations of the conveyor belt. Luckily, even with the combination of physical restraints and advisory, this electric lock-out has been designed in a way that in the case of a disaster that the whole system would be completely powerless.

## 20 Sustainability Considerations

Some of the factors taken into consideration when ascertaining the sustainability of the project long-term operability, material usage, and energy efficiency were:

### *20.1.1.1 Energy Efficiency:*

- This system has a number of energy efficient features including Raspberry Pi. As a result, total energy consumption is minimized. Furthermore, VSD was added for the purpose of improving motor speed but limiting the over excessive use of power.

### *20.1.1.2 Longevity and Reusability:*

- The configuration of the system's design permits ease of upgrade and replacement of components within the system. Part of this design decision was to extend the useable life of the system and such expensive and time-consuming replacement parts.



## 21 Conclusions and Further Developments

### 21.1 Conclusions

An automated inspection system that could sort pass, fail or unrecognized types of industrial products as a full-fledged system was developed as a part of the project since all objectives were achieved. The system proved the feasibility of the application of machine learning in factories in real time using a conveyor, a raspberry pi, plc, and a Cognex camera. The project was able to address some of the issues, particularly network access and ftp configuration issues, through patience and careful troubleshooting of the issues.

### 21.2 Further Developments

Advancement on further automation of the collection chamber in terms of the use of a Festo controller that controls the rotation of the collection chamber has been cited as one aspect that lacks on the project. The first attempts to integrate this controller were unsuccessful because of interface problems that existed between the laptop and the Festo controller. Although the controller was powered on and indicated readiness to work with the P000 homing signal after turn-on, completion of full integration was not feasible by the time the project was due.

In order to make the part sorting process more efficient, new ideas in the future should focus on resolving these communication problems, perhaps by using different interfacing hardware or software. In addition, the accuracy and performance of the system could be enhanced by incorporating more advanced machine learning algorithms or expanding the system's capabilities to handle more complex sorting criteria.

## 22 References

arxiv, 2015. *Deep Residual Learning for Image Recognition*. [Online]

Available at: <https://arxiv.org/abs/1512.03385>

[Accessed 2024].

Cognex, n.d. *Configure the WriteImageFTP Function*. [Online]

Available at:

[https://support.cognex.com/docs/is\\_632/web/EN/ise/Content/How\\_To/Configure\\_WriteImageFTP.htm](https://support.cognex.com/docs/is_632/web/EN/ise/Content/How_To/Configure_WriteImageFTP.htm)

Keras, 2024. *Developer guides / The Sequential model*. [Online]

Available at: [https://keras.io/guides/sequential\\_model/](https://keras.io/guides/sequential_model/)

[Accessed 2024].

Keras, 2024. *Keras 3 API documentation / Callbacks API / EarlyStopping*. [Online]

Available at: [https://keras.io/api/callbacks/early\\_stopping/](https://keras.io/api/callbacks/early_stopping/)

[Accessed 2024].

Keras, 2024. *Keras 3 API documentation / Keras Applications / ResNet and ResNetV2*. [Online]

Available at: <https://keras.io/api/applications/resnet/>

[Accessed 2024].

Keras, 2024. *Keras 3 API documentation*. [Online]

Available at: [https://keras.io/api/data\\_loading/image/](https://keras.io/api/data_loading/image/)

[Accessed 2024].

TensorFlow, 2024. *tf/keras/optimizers/Adam*. [Online]

Available at: [https://www.tensorflow.org/api\\_docs/python/tf/keras/optimizers/Adam](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam)

[Accessed 2024].

Youtube, 2020. *TensorFlow Tutorial 11 - Transfer Learning, Fine Tuning and TensorFlow Hub*.

[Online]

Available at: <https://www.youtube.com/watch?v=WJZoywOG1cs>

[Accessed 2024].

## 23 Appendices

### Additional Supporting Information