ASSIGNMENT 01:
ORCHESTRATION OF INDUSTRIAL EQUIPMENT
Design, Integration and Implementation

Course:

Industrial Informatics (AUT.840-2022-2023-1)

Professor:

Luis Gonzales Moctezuma

Students:

Nadun Ranasinghe (150906193)

Stepan Zalud (151062243)

Ana Sofía Carvajal Argüello (151021213)

Date:

23-11-2022

Index

Introduction

The following report encompasses all the details in regard to the implementation and the background information considered and collected while completing the Assignment 01 for the module Industrial Informatics. The assignment requirement was to control a workstation of the FASTory line to work with multiple orders sent to it by a client. The numerous constraints and hardships that were faced and how they were overcome to introduce the final solution will be discussed briefly.

Objectives

The goal of this project is to solve a given orchestration task with the help of Object Oriented Programming. Also, it will give a good understanding on the role of web services during the integration of automation systems.

Requirements

The requirements to be accomplished to complete the task successfully are as follows:
Conveyor:

- When an order is received, the pallet present at Zone 01 must move either to Zone 02/03 for production or to Zone 04 to be discarded.
- The pallet at Zone 02, must wait until Zone 03 is empty, to be sent for production.
- The pallet at Zone 03 must complete the full drawing execution before being transferred to an empty Zone 05.
- Zone 04 must discard the pallets received to the empty Zone 05 whenever possible.
- Zone 05 must receive completed pallets from Zone 03 and discarded pallets from Zone 04 with no collision.

Robot:

- When the orchestrator receives a phone order, via REST API, the desired parameters of the mobile to be drawn must be extracted from it and they should be used to execute the pen change and the drawing process.
- The three recipes related to the screen, frame and keypad must be drawn sequentially to complete the drawing execution.
- Console logs should notify each step of the process.

Program description

Loop-based method. Event based method

Initially, the program was developed based on the state changes of the conveyors. These state changes can be easily read through the response given by the controller when requested via the REST API. Where the presence of a pallet was shown by the pallet ID and '-1' to show the absence.
This algorithm was designed for a single cycle of a continuously running loop that terminates only at the end of a production. At each cycle, all the states of the zones are read, and if a pallet exists, the relevant algorithm will run and proceed the pallet to its next zone. This system is simpler to develop and code, and also runs with no use of a server. But, the absence of a server and not being able to receive any messages regarding the status of the robot made this loop-based algorithm futile to accomplish all the given requirements.
Therefore, the next approach was taken towards implementing a server via the Flask library to receive notifications from the controller and the user (via REST API), that would act as events to trigger event_handler functions to execute the system. This method not only mitigated the disadvantages shown from the previous method, but also brought in more advantages for the process to execute smoothly.
Although, this system accomplished the tasks of the assignment, various bugs existed, which were removed by the team successfully;

- With no delay existing between the command requests given to draw the three recipes of the mobile, the system was not able to execute the second recipe: This was mitigated by adding time delays between the relevant requests so that the system can identify the relevant commands successfully.
- Not being able to subscribe to the events: The team was able to mitigate this problem by reviewing the code and developing the accurate requests.

Use case diagram

Main part of the whole program is class Orchestrator. (Fig. 1)

Use cases of application:

- Receive order (which is placed from postman)
- Notify final outcome of orders
- Subscribe to events (provided by workstation)
- Process order
- Notify about failure (if there is any)

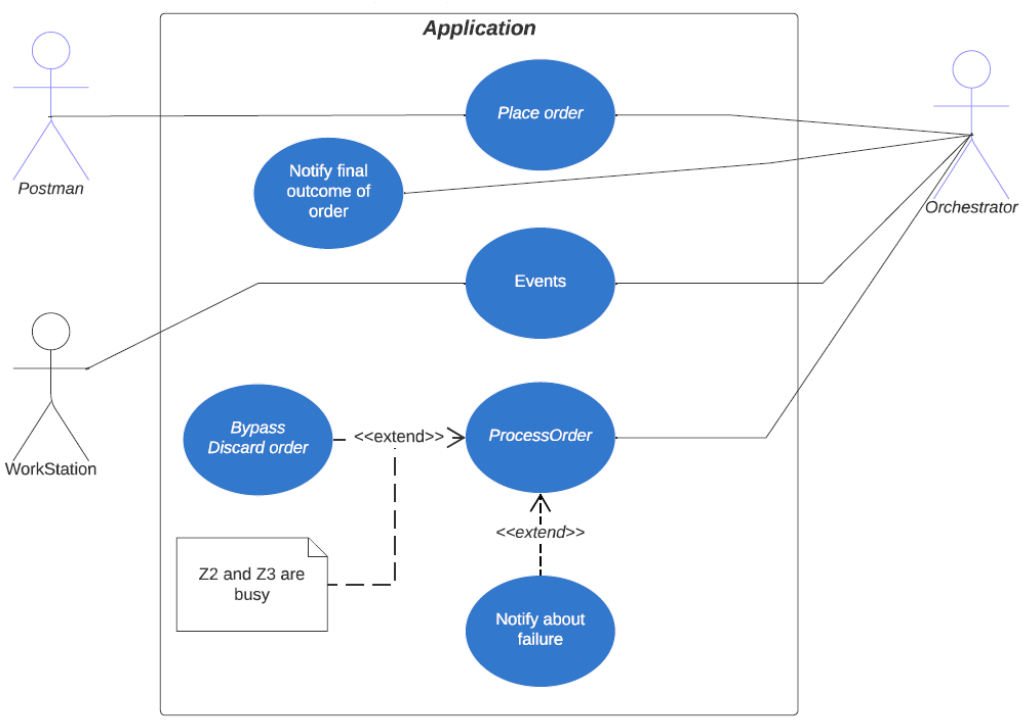

Fig 1. Use Case - Application

OrderProccessing use case:

- Transfer pallets
- Draw mobile
- Get zone state
- Get robot data (working/not working, order finished/not finished) - when needed
- Get order information (when drawing)

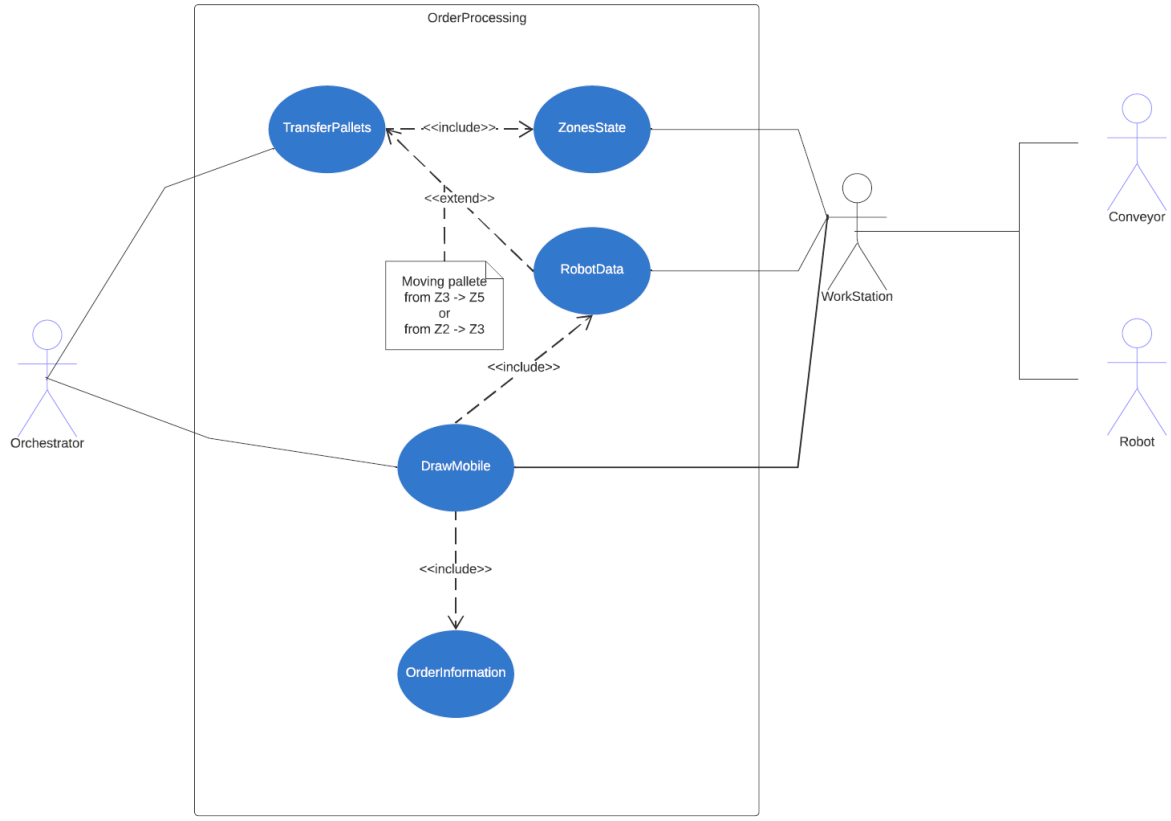

Fig 2. Use case - Order processing

Class diagram

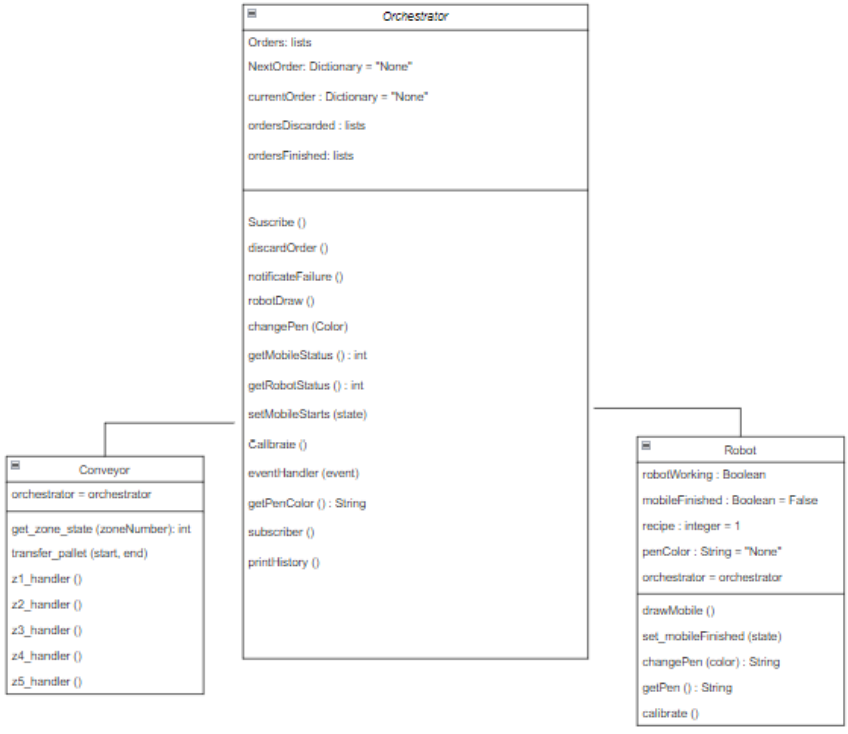

Fig 3. Class diagram

Events

The events are defined according to the status of the workstation, which includes the conveyor and the robot. The events will be triggered via the notifications received by the Flask server from the subscribed services of the server and the requests sent via the REST API by the client PC.
The triggers that activate the given event_handler functions can be divided into two main parts:
1. Controller-given Notifications: Using the FASTory line event notification capabilities, it is possible to subscribe to receive notifications of the changes happening in the workstation. Therefore, these notifications are being used to trigger the relevant event handler functions as follows:

a. Z1_changed |Z2_changed| Z3_changed| Z4_changed | Z5_changed : These notifications are received when there is a change in the 5 zones of the conveyor, i.e, when a pallet arrives/leaves the given zone. These notifications will activate their relevant zone event handlers. (It is shown in Figures 4-8.)

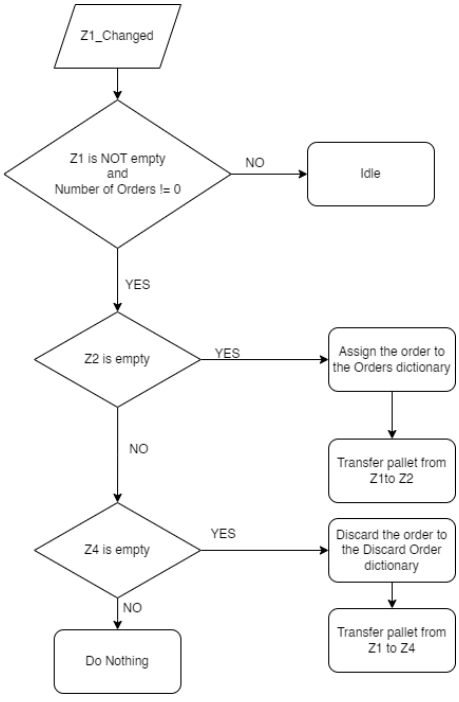

Fig 4. Z1_changed

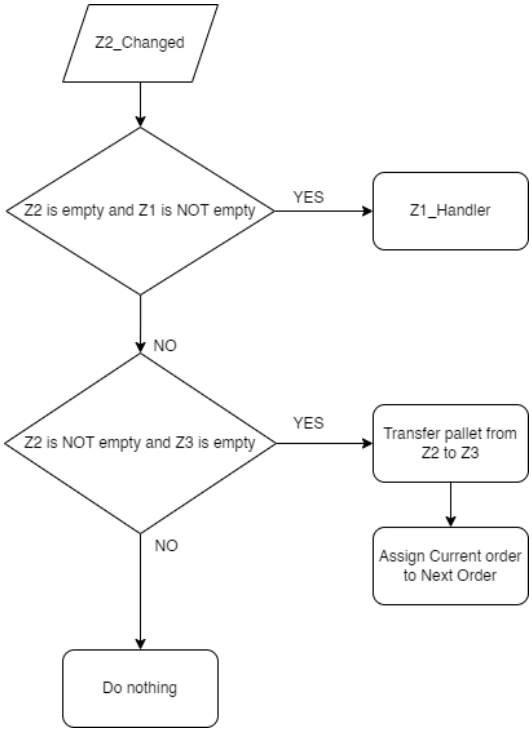

Fig 5. Z2_changed

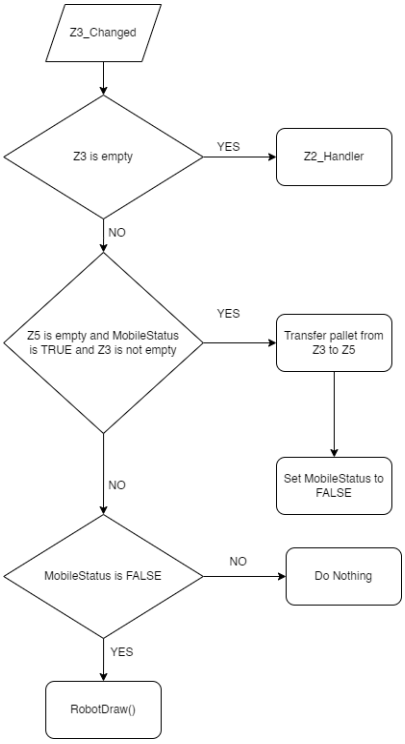

Fig 6. Z3_changed

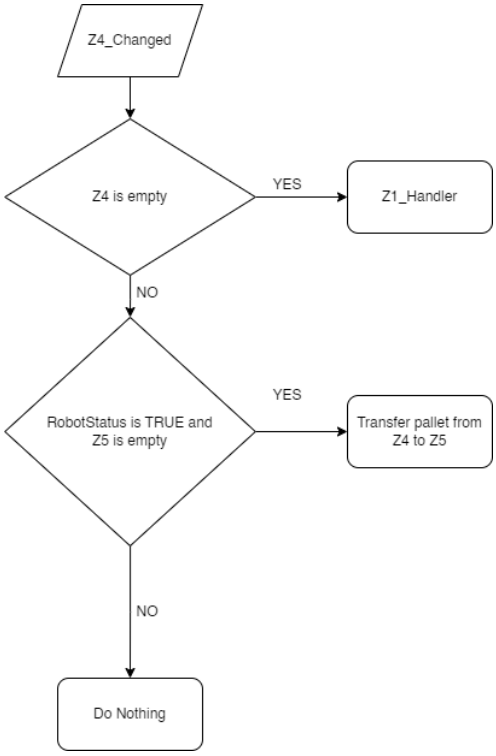

Fig 7. Z4_changed

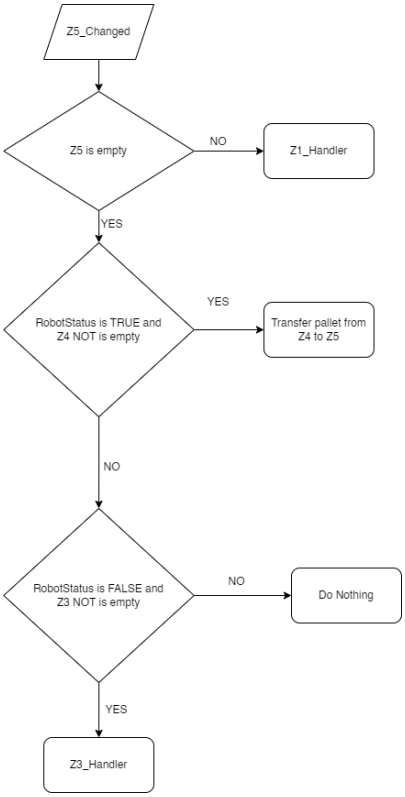

Fig 8. Z5_changed

b. PenChangeStarted | PenChangeEnded: These notifications are received at the start and end of each pen change execution by the robot. While the PenChangeStarted won't be activating any function, other than saving the state that the pen change is in process. The penChangeEnded will trigger the event 02 handler to send any pallet existing to zone 03 for production.

c. DrawStartExecution | DrawEndExecution: These notifications are received at the start and end of each draw execution by the robot. While the DrawStartExecution will trigger the zone 04 handler, the DrawEndExecution will trigger the Zone 03 handler.
2. User-given requests: These requests define the orders to be produced by the system and the end of production. These requests are made by the user/ client of the system. (Fig. 9,10)

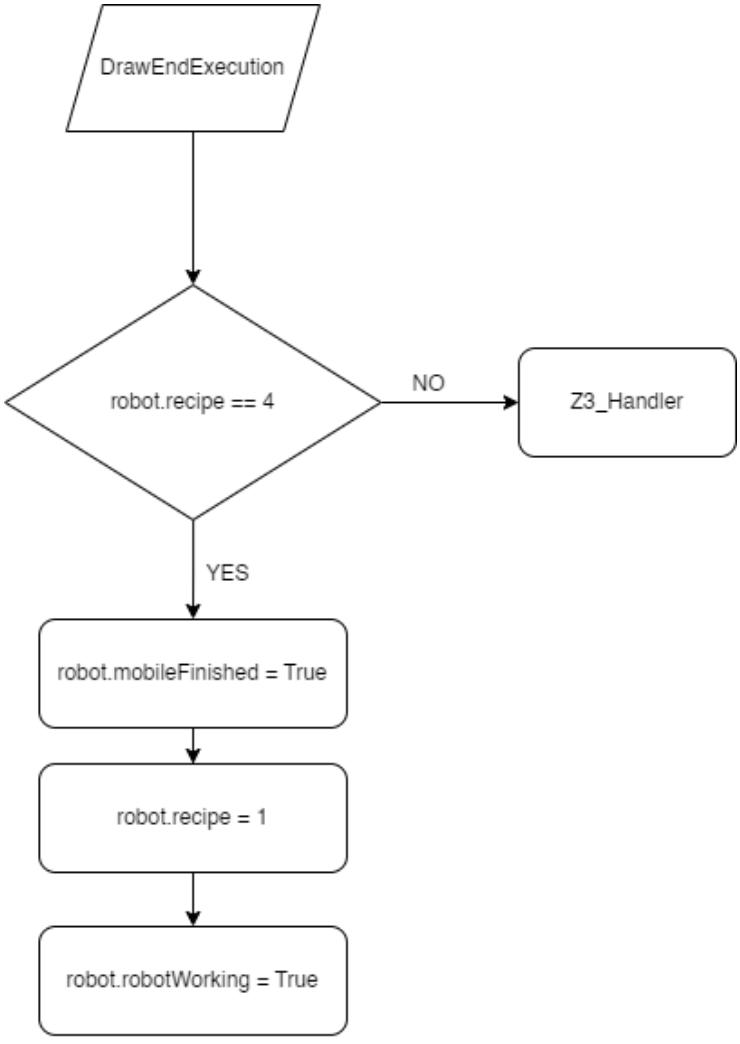

Fig 9. DrawEndExecution

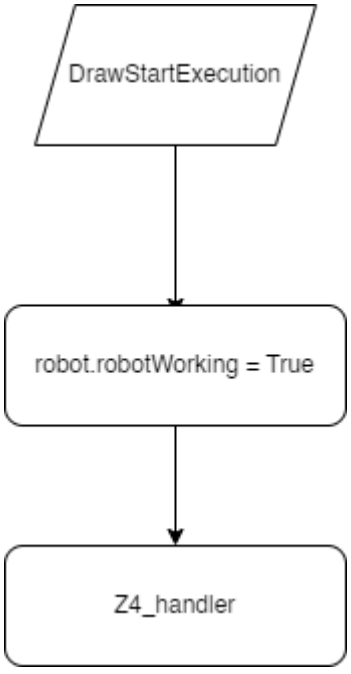

Fig 10. DrawStartExecution

d. Order_handler | System Close : These notifications handles all the orders received and initiates the printHistory function to display a summary of all orders completed, respectively. (Fig. 11 and 12)

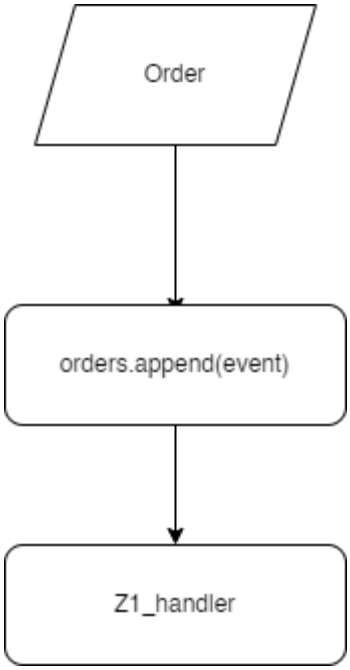

Fig 11. Order_handler

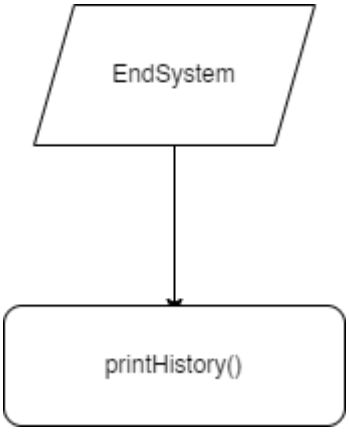

Fig 12. End System

Implementation

The workstation

The whole workstation is divided into two main parts: The conveyor and the robot. Each part is defined under separate classes for smooth control and handling in the code.
The conveyor is further divided into 5 different zones, as it is shown in Fig 13:

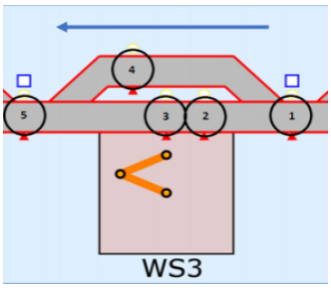

Fig 13. Workstation

1. Zone 01: The initial point at which the pallets are given into the workstation for processing.
2. Zone 02: The waiting area of the pallet prior to production and pen change.
3. Zone 03: The production area of the workstation where the drawing is being executed.
4. Zone 04: The discarding area of the workstation, where the pallet will be placed prior to being discarded.
5. Zone 05: The exit point of the pallet at the end of drawing execution or discarding.

The Robot

The functions of the robot can also be divided into 2 different categories:

1. Draw Execution: Defines the function of drawing the recipes assigned on the pallet paper. This function is capable of drawing the full phone, which consists of three recipes sequentially. This is completed by the use of a single integer variable that counts the number of recipes drawn per pallet and informs the system to send the completed project out. This method makes the system more simple and decreases the complexity of the code.
2. Pen Change: Defines the function of changing the pens according to the user-defined color.

The User

Externally to the workstation, the client PC is defined. The client is capable of controlling the following aspects of the Production process:

1. Orders: The client is capable of sending orders to be executed by the Workstation with the required parameters. But, the discarding decision is taken by the workstation depending on its workload.
2. System End: The client is able to end the whole production process and receive a full report on the completed orders and the discarded orders that the system executed from the start of time.

All the above given sub categories under the Conveyor, Robot and the Client are controlled by unique event handler functions that are activated by the notifications received into the Flask server.

Improvements

The team had observed that following could be developed in the future to improve the system:

- Currently, the pen change function, although it had been implemented, shows bugs related to physical restrictions of the workstation. The team hopes to design and implement new methods to overcome these issues, so that the robot can change colors and draw mobiles efficiently.
- The ability to work with more than one workstation with segregation of work, to make the execution more efficient.
- Create an interactive user interface, to give a clear idea on the process of the system to the clients.

Conclusion

The complexity of working with industrial processes was undertood. The team also realised how teamwork can be beneficial in debugging the system to work effectively as well. Also, by implementing a simple system, that did not complete the tasks, the team realised the importance of Flask in the integration of Automation systems. With the introduction to Python it was also realised about the vast area of opportunities that this programming languages give in the aspects of REST API and OOP.

Appendix

Code: (Github)
Demo: (Video)