POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Flexible manufacturing line

Authors:
Capuana Giorgio 245420
Bellomi Alessandro 245545
Tommaso Panzeri 251423
Gün Kutay Çiçek 991794

Professor: Terzi Sergio
Tutors: Villegas Torres Luis Felipe, Fumagalli Luca
Academic Year: 2023-24

Contents

List of Figures

List of Tables

Introduction

The Management School at Politecnico di Milano has strategically chosen to invest in a substantial piece of equipment to enhance research in advanced manufacturing techniques. This equipment offers a simulated setting that replicates real production scenarios, allowing for direct experimentation and application studies. Each station is equipped with sensors that enable line monitoring. The production system is integrated with a monitoring system that can track various parameters. Control is implemented in real-time through the Manufacturing Execution System (MES), which identifies any assembly errors, and at the end in the manual station, where a Human-Machine Interface (HMI) alerts the operator of any errors detected during the cycle.

Signal	Description
ActivePower	Consumption of energy per unit time
xBG5	Entrance sensor of station
xBG6	Exit sensor of a station
xBG1	To detect if the carrier is in the working position of the station
xSF5	Indicates the status of the emergency button
xCL_BG5	Detects if there is a front/back cover in the station
xCL_BG7	Detects if there is a pallet on the carrier
xCL_BG8	Detects if the front/back cover are on the pallet
iCarrierID	Reference number of the pallet according to the RFID tag

Table 1: Signals table

The assembly line begins with the placement of the front cover onto the pallet. Subsequently, up to four holes are precisely drilled into the cover to accommodate various components. Following this, the product moves through a series of robotic operations, each designed to assemble and secure different parts of the PCBs. At this assembly station, one of the more complex among the seven, a robot places the front cover on a backlit table. A camera mounted above determines the cover's orientation, allowing the robot to correctly position it for the next steps. The robot then places the correct type of PCB

onto the cover and adds the required fuses to their holders, if any. The configuration of the PCB and fuses varies, depending on the product being assembled. This particular task is intricate and might take longer compared to other stages due to the precision needed in aligning and installing the various components.

After the robotic assembly, a camera inspection system checks the quality of the product, ensuring all components are correctly placed and secured. Once the product passes the quality inspection, the back cover is placed onto it. Next, the two covers are firmly pressed together, encapsulating the internal components. This step is crucial for the structural integrity of the phone. Finally, an operator oversees the unloading of the finished product, readying it for the next phase in the manufacturing process.

1. Front cover realising
2. Drilling
3. Robot cell
4. Camera inspection
5. Back cover
6. Pressing
7. Manual station

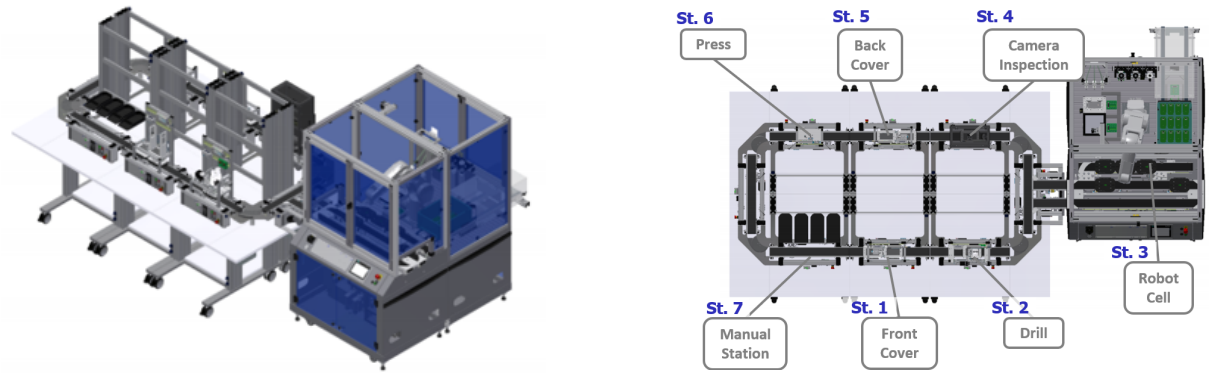


Figure 1: Flexible manufacturing line

1 | Model of the Flexible Manufacturing Line (AS IS) and characterization

1.1. Model of the line on Simulink

The construction of the digital model primarily aims to simulate the behavior of the line, export data, and conduct analysis capable of identifying potential improvements to ensure greater reliability and productivity. At this stage, the line is controlled by the I4.0 block, which has been programmed by the tutors. Specifically, the I4.0 block plays a role in defining the service times of each individual stations while some functions are implemented in order to send signals concerning downtimes.

Just as it is in the reality, the line has been designed with a cyclic geometry.

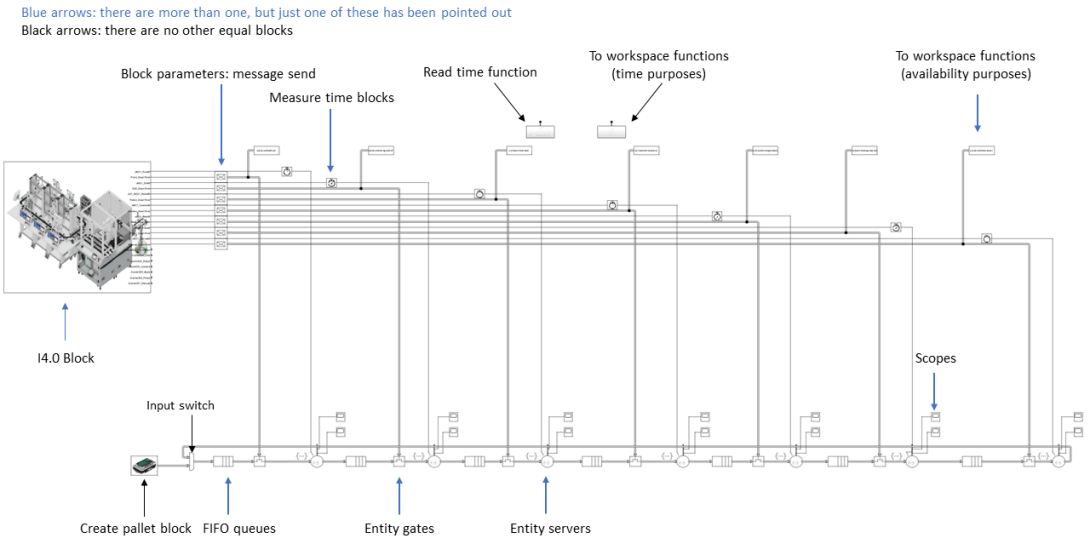


Figure 1.1: Connected AS-IS model.

It all begins with the generation of pallets through a "creator pallet block," an "entity generator" properly set by the tutors to simulate the circulation of a certain number of carriers in line.

To sample the processing times for each unit on each machine (as well as the overall times for a pallet to complete the cycle), the generated pallets are endowed with various attributes. In particular, the following will be measured:

- InitialTime: the moment when a pallet enters the first FIFO queue
- LEADTIME: the total time required to traverse the assembly cycle
- TIN_namestation$_i$: the moment of entry into the i-th station
- TOUT_namestation$_i$: the moment of completion of processing in the i-th station
- TIS_namestation$_i$: time inside i-th station or simply task time

The following table provides a comprehensive overview of the attributes accompanying each pallet.

Attribute Name	Attribute Initial Value
ID	1
initialTime	1
TINFront	1
TOUTFront	1
TISFront	1
TINDrill	1
TOUTDrill	1
TISDrill	1
TINRobot	1
TOUTRobot	1
TISRobot	1
TINCamera	1
TOUTCamera	1
TISCamera	1
TINBack	1
TOUTBack	1
TISBack	1
TINPress	1
TOUTPress	1
TISPress	1
TINManual	1
TOUTManual	1
TISManual	1
LEADTIME	1

Table 1.1: Pallets' attributes initialized at 1

When the simulation begins, there's a very brief delay in the creation of the initial batch of pallets. Specifically, pallets are not generated one at a time, but in a group. The delay and the number of pallets produced can be programmed within the 'create pallet block'.

Adjusting these parameters and running simulations under various working conditions has enabled a thorough study of the system, conducting detailed sensitivity analyses primarily focused on variations in Work in Process (WIP). As we will see later, these analyses have been crucial for understanding the encountered issues and for verifying the consistency and accuracy across different simulations (and therefore the model itself), as well as for fulfilling the tasks required by the project.

Once the pallets are generated, they enter the cycle. During their journey, the attributes contained within the entities change their values, collecting the data of our interest. The first block encountered is a FIFO (First-In, First-Out) queue, which is discussed and explained in detail below. However, it is important to note that the calculation of the overall productive lead time, subject to all the system's variability, begins for each pallet from the moment it enters the first block. The line of code "entity.InitialTime = ReadTime()" has been written within the first queue. It calls the ReadTime function that is programmed to record time instances. This function allows the simulation to progress in real time. It is important to note that all simulations conducted had a duration of 28800 seconds, which is equivalent to 8 hours, matching a standard work shift.

The figures below show what there is inside the "create pallet block" and the "ReadTime function".

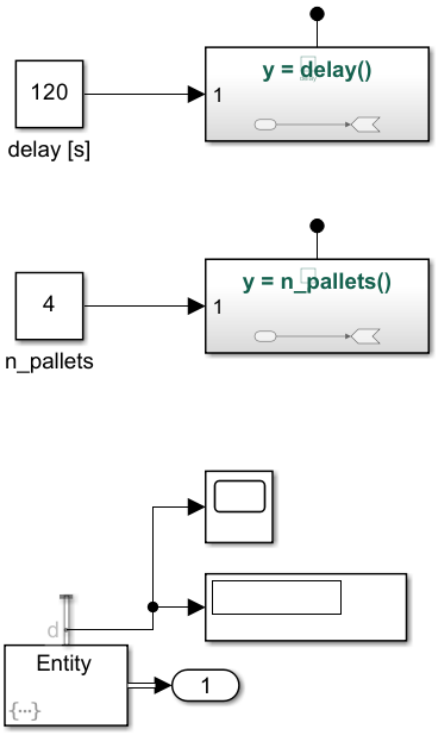


Figure 1.2: Create pallet block

Figure 1.3: Read time function

As it possible to figure, realistically pallets are placed on the line and transported from the first station to the last one by mean of a conveyor. To model the conveyor belt, seven "FIFO queue" blocks have been arranged: one before each stations. They act as a sort of inter-operational buffers based on the "first in, first out" logic. Here, the work in progress

(WIP) may be temporarily stopped in case the machine is unavailable, typically due to a built-up pieces queue or a breakdown. If not, the pallet is allowed to enter the station to undergo the sequence of operations implemented at that particular station.

As it can be seen from the AS IS model of the FML (Figure 1.1), each of the these inter-operational buffers is followed by an "entity gate." This block is connected to the I4.0 block by means of a "message send block," which creates and sends a message by reading the input signal at the input signal source block sample time. The block just discussed is replicated once before each "entity server", as it has been done for the "FIFO queue". The purpose of the "entity gate" is to stop the conveyor belt in case of a machine failure. For this reason, it was decided to insert the monitoring block between the "FIFO queue" and the respective machine on which the gate acts.

Afterwards, the machine setting has been crucial for consistency simulate the behaviour of the system. Firstly, as mentioned before, each machine has been arranged after its related entity server, thus creating a seven times repeated structure made as follow:

`"FIFO queue" => "entity gate" => "entity server".`

- Main
  - Capacity: 1
  - Service time source: Signal port
- Event action
  - Entry*:

    ```
    entity.TINFront = ReadTime();
    ```

  - Service complete*:

    ```
    entity.TOUTFront = ReadTime();

    entity.TISFront = entity.TOUTFront - entity.TINFront;
    ```

- No preemption
- Statistics (not compulsory but useful to quickly check the simulation output)
  - Number of entities departed
  - Number of pending entities
  - Utilization

A machine can process one pallet at a time, and the processing times are still determined by the I4.0 block, which communicates with the station through the "measure time" block and a signal port that enables the exchange of information. When a pallet enters the machine and when the service is completed, the respective timestamps are recorded. Additionally, the time the machine takes to perform its operations is calculated when the pallet exits. This calculation is simply the difference between the two timestamps, which, for greater clarity, has been termed "time in the system" (TIS). This mechanism takes place on every machine and for each pallet processed over the course of the 28800 seconds simulation. Thanks to these lines of code, our simulation is capable of collecting essential data on service times. Afterwards, the data can be exported to the workspace for further processing, cleaning, and aggregation, in preparation for the next stages of the project.

The statistics can either be displayed graphically through 'Scope' or exported to Matlab as time series using the original 'To Workspace' blocks provided by Simulink.

Scopes did not play a crucial role in our work, apart from buffer sizing (last section of the redesign), nonetheless they were extremely helpful in verifying the quality of the simulations, the consistency among the different phenomena occurring in the cycle (downtime, queues, etc.), as well as in monitoring the correct parallelism between our ideas and their implementations.

In order to let the reader aware about the worth of monitoring by scopes, the following figure was captured during one of our firsts simulation run:

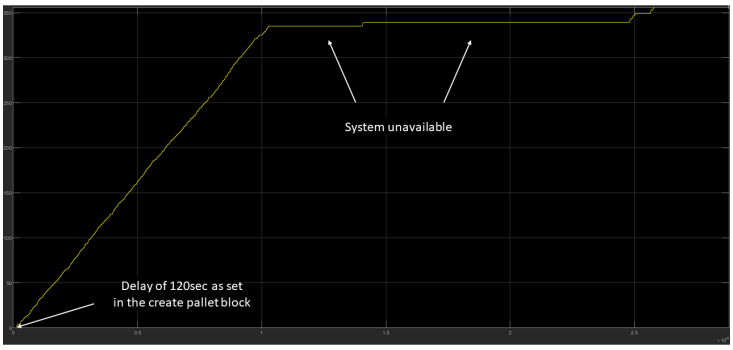


Figure 1.4: Example of statistics exploitation

Lastly, the final station received a similar setup, but with the addition of two lines of code: the first for measuring the overall production LeadTime of each pallet, and the second for exporting to the workspace the data carried by each pallet.

A figure is attached below

Figure 1.5: Event action service completed on the 7th station

To complete the construction of the digital model, let's consider the approach used for exporting the data sets of our interest. For the service times of each machine (durations of individual operations at each station for every processed pallet), a 'To Workspace' function was created. This function encompasses the functions for exporting the individual vectors formed by the samplings.

It's important to note that the output is not a time series, but a set of samplings on the durations of the operations. By creating a subfunction for each machine, the output will be 7 column vectors of equal length. Besides the essential service times, another crucial piece of information is the length of the vector. This indicates the number of sampled pallets, which can be assumed to be the total number of pallets that have entered and exited the production cycle.

The picture below shows the inside of the 'To Workspace' function (time purposes).

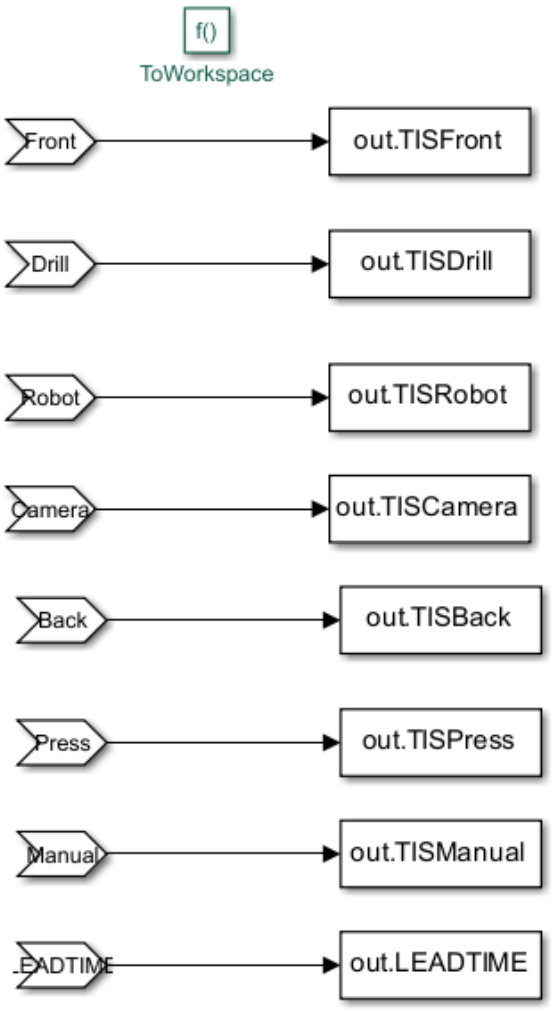


Figure 1.6: To Workspace function (time purposes)

In terms of sampling the time instances of good production on each machine, 'To Workspace' functions were also used in this case. The difference from the previous case lies in the intention to obtain a time series, that is, a vector where the first column represents the sampling time instances (timestamps), and the second column contains binary numbers where:

- 1 = if the station is on uptime status
- 0 = if the station is on downtime status

Now that the model is ready, the simulation can be initiated, and it's possible to work on the output data from Simulink, which are transferred as input to Matlab.

1.2. Statistical analysis coding on Matlab

1.2.1. Initial hypotesis and highlights

Once the data has been exported from Simulink to Matlab, it is possible to begin studying, processing, and cleaning it.

Firstly, seven vectors were created (Data 1, ..., Data 7), each necessary to contain the performance times of each machine, sampled during the processing of each pallet. These column vectors have the same length and are therefore combined into a matrix.

Subsequently, a cyclic construct is used to create a histogram for each column of the matrix. The aim is to visually show how the samples are distributed.

In analyzing the system, it might have been that the duration of operations for each machine follows a normal, or Gaussian, distribution pattern. It can be hypothesized that the processing times sampled on the same machine are independent of each other. Moreover, the number of samples typically detected in an 8-hour simulation is considerably large: in a generic simulation chosen as a reference model, a total of 320 samples were obtained, including clean data and outliers.

The numerosity and independence of the data are the hypotheses behind the central limit theorem, according to which the data that make up the dataset tend to be distributed following a normal distribution regardless of the underlying distribution.

We wanted to verify that this was also valid for our set of data and chose to plot histograms. In order to plot them, it was necessary to define the optimal number of columns (k) that would enable a better visualization of the distribution. For this task, the 'Sturges' rule' was used: $K = 1 + \log_2(N)$ Subsequently, an experimental corrective coefficient (chosen from a series of different attempts) was applied with the aim of further improving the visualization.

The results can be seen in "subsection 1.1.6.: Visual outcome."

Matlab Code:

```
Data1 = out.TISFront;
Data2 = out.TISDrill;
Data3 = out.TISRobot;
Data4 = out.TISCamera;
Data5 = out.TISBack;
Data6 = out.TISPress;
Data7 = out.TISManual;
DataAll = [Data1 Data2 Data3 Data4 Data5 Data6 Data7];
% plot istogrammi
for i=1:width(DataAll)
    data = DataAll(:,i);
    n = length(data);
    k = ceil(1.3*ceil(log2(n) + 1));
    figure(i)
    histogram(data, k);
    title(sprintf('histogram station %d', i))
    xlabel('Time(s)')
    ylabel('Sample')
end
```

1.2.2. Best statistical distributions fitting on the whole dataset

At this point, we have histograms representing the data distribution. From these, it's evident that not all stations operate with normally distributed service times. Rather, it should be noted that **the majority of the seven stations do not exhibit normal behavior at all**, but instead, the distributions take on shapes very different from the classic bell curve.

For this reason, the assumption of normality of the data was discarded. Therefore, a sequential search was conducted for each station to find the 'distribution that best approximates the behavior of the machine'.

A pool of potential notable distributions was defined to determine which of these is the distribution model that best fits the data. The following distributions were tested: Gaussian, Gamma, Weibull, Lognormal, Exponential, Poisson. The script shown below uses mathematical methods to fit different distribution models to a set of data and to select the model that best fits. To perform this operation, the Akaike Information Criterion (AIC) was been applied:

- The AIC is calculated to quantify the quality of each model, taking into account both the model's fit to the data and the number of parameters used.
- The AIC penalizes models with more parameters to avoid overfitting, favoring models that offer a good balance between complexity and fit to the data.
- `aicbic(-loglik, num_params)`: `loglik` is the negative log-likelihood of the model, and `num_params` is the number of parameters in the model.

Thanks to this method, it is possible to create a 'struct', an array capable of accommo-

dating different formats, named fitResults, constructed as shown in the example below (Figure 1.7).

Each 'Column' represents a station, where 1 = Front cover, ..., 7 = Manual.

Within these, all the information on the tried distributions (with their respective parameters) are reported, and the 'best model' is shown as the last record of the struct. The image below clearly demonstrates what we had hypothesized: the model is capable of identifying the best distributions, and in most cases, these are not Gaussian (readers are invited to pause and visually verify this by referring to 'subsection 1.1.6.: Visual outcome').

The image below shows the result for only two stations due to space limitations.

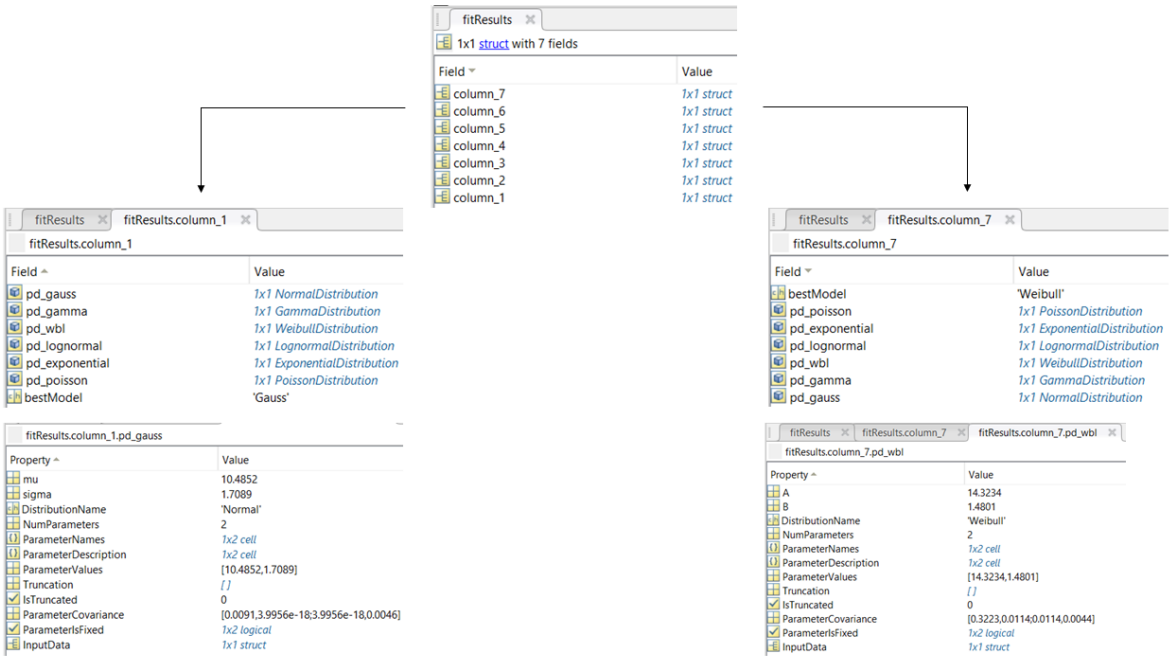


Figure 1.7: Distribution analysis output

Matlab Code:

```
% Inizializza la struttura esterna
fitResults = struct();

for i = 1:7
    % Estrai i dati della colonna i-esima
    data = DataAll(:,i);

    % Inizializza una struttura interna per ogni colonna
    fitResults.(sprintf('column_%d', i)) = struct();

    % Adatta e memorizza i diversi modelli di distribuzione
    fitResults.(sprintf('column_%d', i)).pd_gauss = fitdist(data, 'Normal');
    fitResults.(sprintf('column_%d', i)).pd_gamma = fitdist(data, 'Gamma');
    fitResults.(sprintf('column_%d', i)).pd_wbl = fitdist(data, 'Weibull');
    fitResults.(sprintf('column_%d', i)).pd_lognormal = fitdist(data, 'Lognormal');
    fitResults.(sprintf('column_%d', i)).pd_exponential = fitdist(data, 'Exponential');
    fitResults.(sprintf('column_%d', i)).pd_poisson = fitdist(data, 'Poisson');
    % Calcola l'AIC per ogni distribuzione
    aic_vals = [
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_gauss.negloglik, numel(fitResults.(sprintf('column_%d', i)).
            pd_gauss.Params)),
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_gamma.negloglik, numel(fitResults.(sprintf('column_%d', i)).
            pd_gamma.Params)),
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_wbl.negloglik, numel(fitResults.(sprintf('column_%d', i)).
            pd_wbl.Params)),
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_lognormal.negloglik, numel(fitResults.(sprintf('column_%d', i))
            .pd_lognormal.Params)),
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_exponential.negloglik, numel(fitResults.(sprintf('column_%d', i
            )).pd_exponential.Params)),
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_poisson.negloglik, numel(fitResults.(sprintf('column_%d', i)).
            pd_poisson.Params))
    ];

    % Determina la distribuzione con il minimo AIC
    [~, bestModelIndex] = min(aic_vals);
    bestModelNames = {'Gauss', 'Gamma', 'Weibull', 'Lognormal', 'Exponential', 'Poisson'};
    bestModel = bestModelNames{bestModelIndex};

    % Memorizza il nome del modello migliore per questa colonna
    fitResults.(sprintf('column_%d', i)).bestModel = bestModel;
end
```

1.2.3. Detection and removal of outliers

Knowing that the distributions are not normal makes the process of removing outliers more complex.

It becomes necessary to carefully choose the method to use in order to find the right compromise between removing dirty and clean data, defining thresholds such that a data point is recognized as an outlier.

The interquartile range method, commonly abbreviated as IQR, is a statistical technique used to measure dispersion and potentially identify outliers in a data set. The beauty of the IQR lies in its simplicity and robustness, especially in the presence of skewed distributions or data with extreme values.

To calculate the IQR, we start by determining the quartiles of a data set. Quartiles divide the ordered data set into four equal parts. The first quartile (Q1) represents the value

below which 25% of the data falls, while the third quartile (Q3) represents the value below which 75% of the data falls. The IQR is the difference between these two quartiles (Q3 - Q1). This range represents the dispersion of the data around the median, providing a clear idea of the internal variability of the data, excluding the extremes.

The next step in using the IQR is identifying outliers. Generally, any value that falls outside the range defined by `Q1 - 1.5IQR` and `Q3 + 1.5IQR` is considered an outlier. This rule, known as the '1.5*IQR rule', is flexible and can be adapted depending on the context and the needs of the analysis. This characteristic of the IQR makes it particularly useful in situations where outliers might distort the outcome of an analysis.

The IQR is often displayed in box plots, a type of graph that illustrates the distribution of data by highlighting the interquartile range. In a box plot, the 'box' represents the IQR, and the 'whiskers' extend from the quartiles to the furthest values that are not considered outliers. The use of box plots provides an immediate visual representation of data dispersion, making the IQR a valuable tool not only for statistical analysis but also for data visualization.

Given the specifics, the decision was made to use Box Plots to more clearly visualize the outliers. In this regard, the Box Plot is a clearer and more immediate tool compared to creating histograms.

The cleaned data are stored internally in the dataClean vectors: DataClean1, ..., DataClean7, each neatly representing a respective station (the convention remains the same: 1 = Front Cover Station, ..., 7 = Manual Station). Since the number of outliers removed is not equal among all the Data1, ..., Data7 vectors, the DataClean vectors will have different dimensions from one another. Therefore, they are studied individually, as constructing a matrix (at least one structured similarly to the DataAll matrix) is not feasible.

To visualize how the cleaned data are distributed, the previous "for loop" is reused to plot the histograms again, with the difference that outliers have been removed. Finally, it is possible to observe the actual behavior of each station, and once again we can affirm that assuming normality of the data (although theoretically acceptable) may be a rough approximation.

```
% Inizializza i cell arrays per i dati puliti
for i = 1:7
    data = DataAll(:,i);

    % Calcola l'IQR
    Q1 = quantile(data, 0.25);
    Q3 = quantile(data, 0.75);
    IQR = Q3 - Q1;

    % Identifica gli outlier
    outlierIndices = data < (Q1 - 1.5 * IQR) | data > (Q3 + 1.5 * IQR);

    % Rimuovi gli outlier
    dataClean = data(~outlierIndices);

    % Assegna i dati puliti al cell array corrispondente

    eval(['Dataclean' num2str(i) ' = dataClean;']);


    %(Opzionale) Visualizza un box plot per vedere gli outlier
    figure(i);
    boxplot(data, 'whisker', 1.5);
    title(sprintf('Box Plot with Outliers for Column %d', i));
end

% Ora puoi accedere ai dati puliti da ogni stazione tramite Dataclean{index}

for i = 1:7
eval(['data = Dataclean' num2str(i) ';']);
n = length(data);
k = ceil(1.3*ceil(log2(n) + 1));
 figure(i)
 histogram(data, k);
 title(sprintf('Histogram for Clean Data of Station %d', i))
 xlabel('Time(s)')
 ylabel('Sample')
 % nota: il ciclo non    chiuso perch  utilizzato anche per la successiva porzione di codice
```

1.2.4. Best statistical distribution fitting on the clean dataset

Now that the data has been cleaned and there is visual confirmation that fitting the data with a normal distribution is not feasible, there arises the need to once again find the best distribution that approximates the behavior of the cleaned data.

As one might imagine, it would not be correct to assume that clean data and data with outliers can be represented by the same probability density function. Furthermore, even if this were true, the removal of extraneous data would result in a change (albeit minimal) in the expected values, variances, and especially in the parameters that characterize the distributions.

For this reason, a script similar to the one presented in 'subsection 1.2.2.' runs through the various DataClean arrays using the loop from the previous paragraph. The output is again a struct where the most suitable distributions and their respective parameters are declared. These will be of vital importance at the time of disconnecting the line from the I4.0 block.

The results confirm the predictions:

- There's a possibility that the type of distribution remains unchanged, but then it would be the parameters, the expected value, and the variance that change.
- There's a possibility that the model chooses a different distribution to fit the data.

The Drill station shown in the figure below is just one example, but in reality, the entire cleaned dataset exhibits a behavior that is similar yet different from the original one.

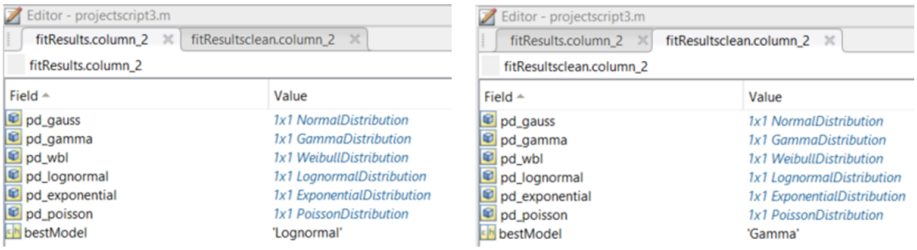


Figure 1.8: Comparison between distributions on dirty and clean datasets for Drill station

```
% continua precedente ciclo for
...
ylabel('Sample')

    % Inizializza una struttura interna per ogni colonna
    fitResultsclean.(sprintf('column_%d', i)) = struct();

    % Adatta e memorizza i diversi modelli di distribuzione
    fitResultsclean.(sprintf('column_%d', i)).pd_gauss = fitdist(data, 'Normal');
    fitResultsclean.(sprintf('column_%d', i)).pd_gamma = fitdist(data, 'Gamma');
    fitResultsclean.(sprintf('column_%d', i)).pd_wbl = fitdist(data, 'Weibull');
    fitResultsclean.(sprintf('column_%d', i)).pd_lognormal = fitdist(data, 'Lognormal');
    fitResultsclean.(sprintf('column_%d', i)).pd_exponential = fitdist(data, 'Exponential');
    fitResultsclean.(sprintf('column_%d', i)).pd_poisson = fitdist(data, 'Poisson');
    % Calcola l'AIC per ogni distribuzione
    aic_vals = [
        aicbic(-fitResultsclean.(sprintf('column_%d', i)).pd_gauss.negloglik, numel(fitResultsclean.(sprintf('column_%d
            ', i)).pd_gauss.Params)),
        aicbic(-fitResultsclean.(sprintf('column_%d', i)).pd_gamma.negloglik, numel(fitResultsclean.(sprintf('column_%d
            ', i)).pd_gamma.Params)),
        aicbic(-fitResultsclean.(sprintf('column_%d', i)).pd_wbl.negloglik, numel(fitResultsclean.(sprintf('column_%d',
             i)).pd_wbl.Params)),
        aicbic(-fitResultsclean.(sprintf('column_%d', i)).pd_lognormal.negloglik, numel(fitResultsclean.(sprintf('
            column_%d', i)).pd_lognormal.Params)),
        aicbic(-fitResultsclean.(sprintf('column_%d', i)).pd_exponential.negloglik, numel(fitResultsclean.(sprintf('
            column_%d', i)).pd_exponential.Params)),
        aicbic(-fitResults.(sprintf('column_%d', i)).pd_poisson.negloglik, numel(fitResultsclean.(sprintf('column_%d',
            i)).pd_poisson.Params))
    ];

    % Determina la distribuzione con il minimo AIC
    [~, bestModelIndex] = min(aic_vals);
    bestModelNames = {'Gauss', 'Gamma', 'Weibull', 'Lognormal', 'Exponential', 'Poisson'};
    bestModel = bestModelNames{bestModelIndex};

    % Memorizza il nome del modello migliore per questa colonna
    fitResultsclean.(sprintf('column_%d', i)).bestModel = bestModel;
end
% nota: l'end chiude il ciclo for per paragrafo precedente
```

1.2.5. Computation of expected values and variances for each station's service time

To conclude the statistical part and proceed to the subsequent phases of the project, it was first necessary to organize all this information.

As a first step, a new "for loop" is implemented with the goal of creating a 1x7 dimensional array of structs that contains all the information about the "best models". To do this, the loop retrieves the name of the best distribution, selects it, and stores all its information.

An error is returned in case the name of the best distribution is not found or recognized due to potential errors in the code or issues during execution (like a lack of data if the Simulink simulation had not been run previously).

The final phase involved calculating the expected values and variances, parameters that will be displayed, saved, and utilized in subsequent analyses.

[seconds/piece]	ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7
Expected values	**10.47**	**11.04**	**26.70**	**9.22**	**10.56**	**10.09**	**12.49**
Variances	2.71	14.68	81.13	14.89	3.48	17.29	68.33

Table 1.2: Expected values and variances of service times

By carefully studying the behavior of the individual stations, it can be observed that the Robot Cell is the slowest, as well as the one subject to the most variability! This station will be the focus in the analysis of throughput, in the study of the overall system, and most importantly, in the decisions that the group has chosen to make with the goal of improving the performance of the entire line.

Figure 1.9: Robot Cell

```
% Inizializza la cella per le distribuzioni migliori
bestFitDistributions = cell(1, 7);

for i = 1:7
    % Ottieni il nome del modello migliore per la colonna i-esima
    bestModel = fitResultsclean.(sprintf('column_%d', i)).bestModel;

    % Seleziona e memorizza la distribuzione corrispondente al modello migliore
    switch bestModel
        case 'Gauss'
            bestFitDistributions{i} = fitResultsclean.(sprintf('column_%d', i)).pd_gauss;
        case 'Gamma'
            bestFitDistributions{i} = fitResultsclean.(sprintf('column_%d', i)).pd_gamma;
        case 'Weibull'
            bestFitDistributions{i} = fitResultsclean.(sprintf('column_%d', i)).pd_wbl;
        case 'Lognormal'
            bestFitDistributions{i} = fitResultsclean.(sprintf('column_%d', i)).pd_lognormal;
        case 'Exponential'
            bestFitDistributions{i} = fitResultsclean.(sprintf('column_%d', i)).pd_exponential;
        case 'Poisson'
            bestFitDistributions{i} = fitResultsclean.(sprintf('column_%d', i)).pd_poisson;
    end
end

% Ora la cella bestFitDistributions contiene le distribuzioni migliori per ogni colonna
% Inizializza gli array per memorizzare i valori attesi e le varianze
expectedValues = zeros(1, 7);
variances = zeros(1, 7);
for i = 1:7
    % Ottieni la distribuzione corrente
    pd = bestFitDistributions{i};

    % Calcola il valore atteso e la varianza in base al tipo di distribuzione
    switch class(pd)
        case 'prob.NormalDistribution'
            expectedValues(i) = pd.mu; % Media per la distribuzione normale
            variances(i) = pd.sigma^2; % Varianza per la distribuzione normale
        case 'prob.GammaDistribution'
            expectedValues(i) = pd.a * pd.b; % Media per la distribuzione gamma
            variances(i) = pd.a * pd.b^2; % Varianza per la distribuzione gamma
        case 'prob.WeibullDistribution'
            expectedValues(i) = pd.A * gamma(1 + 1/pd.B); % Media per la distribuzione Weibull
            variances(i) = pd.A^2 * (gamma(1 + 2/pd.B) - (gamma(1 + 1/pd.B))^2); % Varianza per la distribuzione
                Weibull
        case 'prob.LognormalDistribution'
            expectedValues(i) = exp(pd.mu + pd.sigma^2 / 2); % Media per la distribuzione lognormale
            variances(i) = (exp(pd.sigma^2) - 1) * exp(2 * pd.mu + pd.sigma^2); % Varianza per la distribuzione
                lognormale
        case 'prob.ExponentialDistribution'
            expectedValues(i) = pd.mu; % Media per la distribuzione esponenziale
            variances(i) = pd.mu^2; % Varianza per la distribuzione esponenziale
        case 'prob.PoissonDistribution'
            expectedValues(i) = pd.lambda; % Media per la distribuzione di Poisson
            variances(i) = pd.lambda; % Varianza per la distribuzione di Poisson

        otherwise
            warning('Tipo di distribuzione non riconosciuto.');
    end
end

% Arrotonda i valori a 2 cifre decimali
expectedValues = round(expectedValues, 2);
variances = round(variances, 2);
% Stampa i valori attesi e le varianze per ciascuna distribuzione
for i = 1:7
    fprintf('Colonna %d - Valore atteso: %.2f, Varianza: %.2f\n', i, expectedValues(i), variances(i));
end
```

1.2.6. Visual outcome

In this paragraph, we aim to show the histograms and boxplots created using Matlab.

On the left, we find the histograms created with the data including outliers, in the center are the boxplots, and on the right are the histograms created from the cleaned data.

Detailed information about the distributions are provided in Figure 1.17.

Front Cover Station

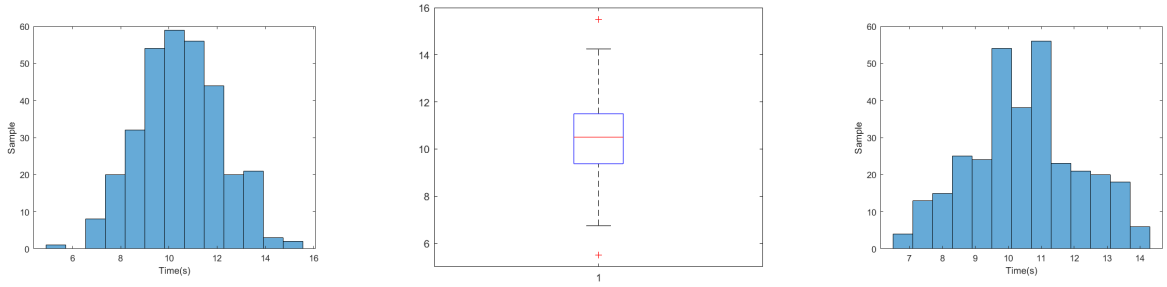


Figure 1.10: Distributions and outliers boxplot (Front Cover Station)

Drilling Station

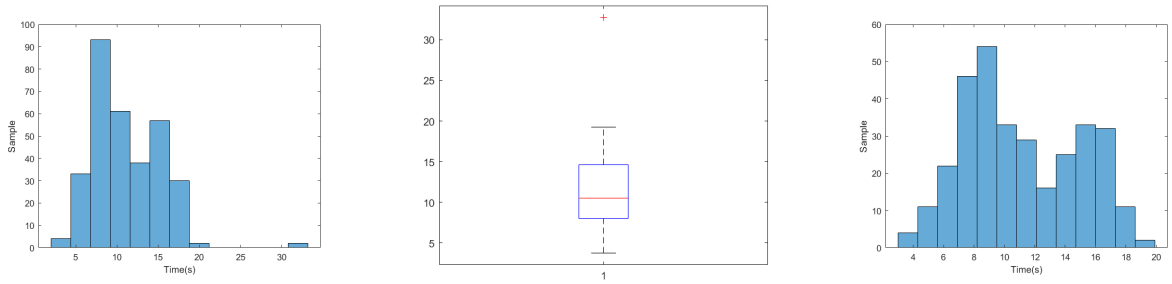


Figure 1.11: Distributions and outliers boxplot (Drilling Station)

Robot Cell Station

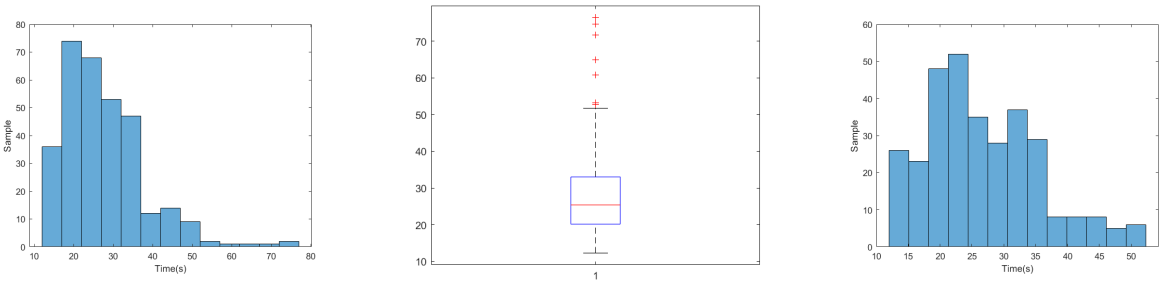


Figure 1.12: Distributions and outliers boxplot (Robot Station)

Camera Inspection Station

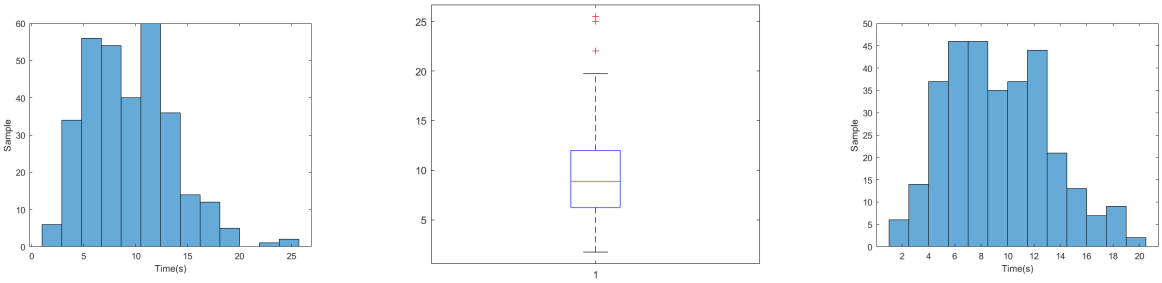


Figure 1.13: Distributions and outliers boxplot (Camera Inspection Station)

Back Cover Station

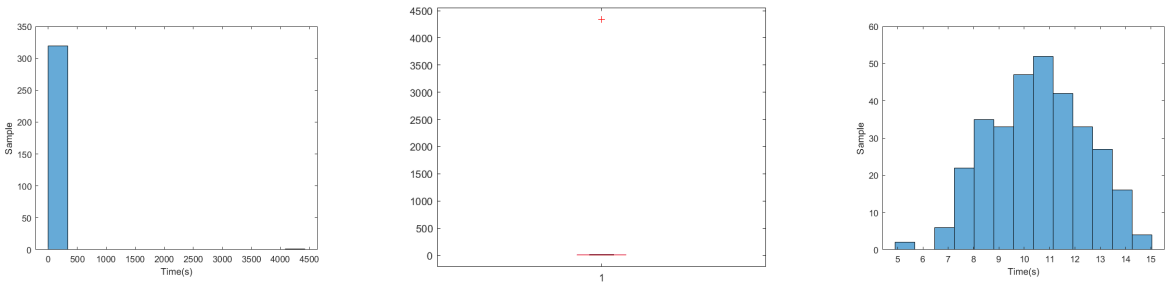


Figure 1.14: Distributions and outliers boxplot (Back Cover Station)

Pressing Station

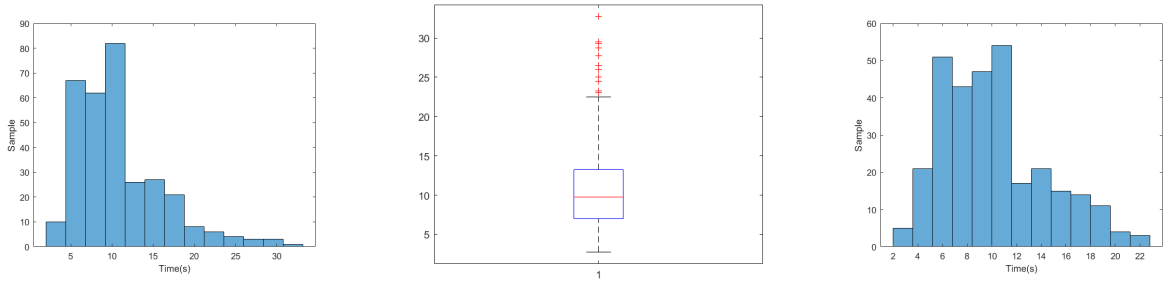


Figure 1.15: Distributions and outliers boxplot (Pressing Station)

Manual Station

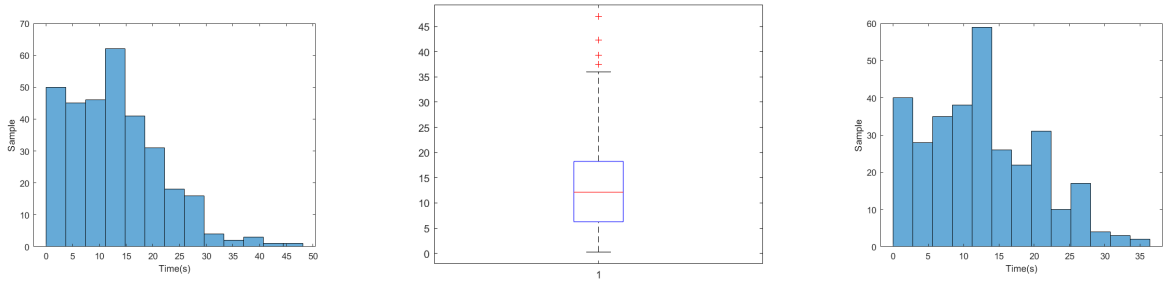


Figure 1.16: Distributions and outliers boxplot (Manual Station)

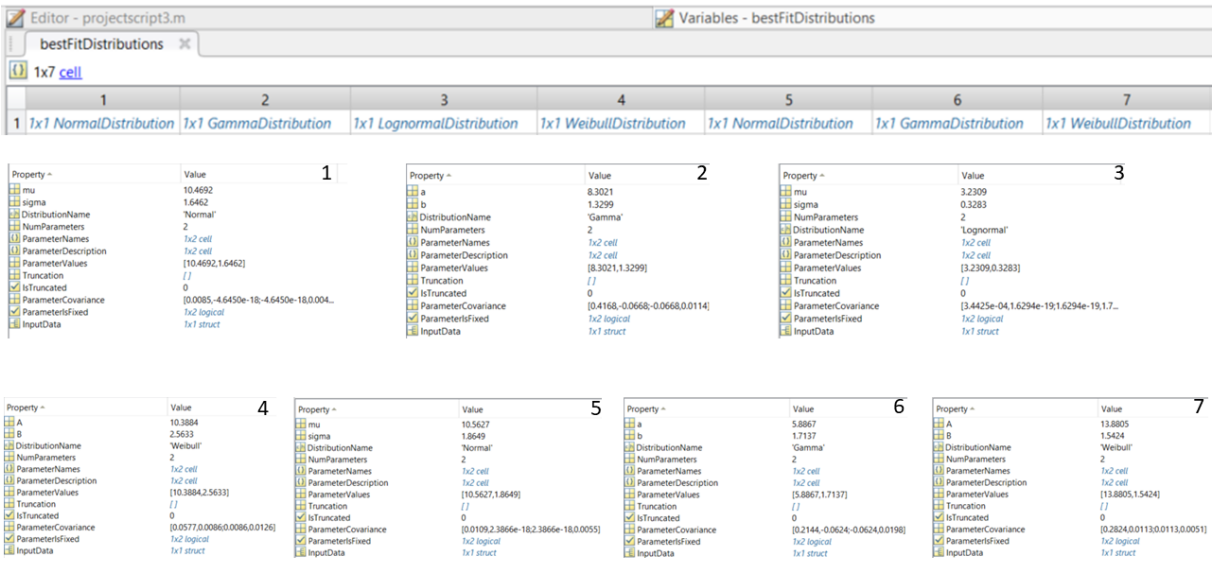


Figure 1.17: Service time distributions defined (Clean Data)

2 | Throughput analysis

2.1. Bottleneck identification

To find the bottleneck in a production line, we first need to understand where congestion occurs. This happens when the workload arrives faster than the production process can manage. To address this, data from the simulation, specifically about processing times, has been collected and analyzed.

Throughput rate has been considered as the right parameter in order to to detect the bottleneck. The reason behind is that all the parts follow a serial process, thus they have the same routing. On the contrary, if products would have followed different routings we should have detected the bottleneck station by looking at the utilization rates.

To sum up, utilization rates and throughput reach the same conclusion when products has the same routing. Differently, utilization rates and throughput might provide different outcomes in terms of detecting the bottleneck.

In order to compute the throughput, the following formula has been exploited.

$$Throughput\ rate\ (TH) = \frac{1}{mean\ process\ time}$$

station	Front	Drill	Robot	Camera	Back	Press	Manual
Process time [sec/pallet]	10,470	10,040	26,700	9,220	10,560	10,090	12,490
TH [pallet/sec]	0,096	0,100	0,037	0,108	0,095	0,099	0,080

We calculated the throughput with the formula just mentioned since the machines process only one pallet at a time.

After having calculated all the throughputs we concluded that the Robot Cell is the bottleneck with a throughput value of 0.037 pcs/seconds which is the minimum among all stations.

2.2. Analysis of the behaviour of the line based on Little's law

After the detection of the bottleneck our next step was to measure the critical WIP level and to analyze the behavior of the line in accordance with the number of WIP.

For that we have computed the lead time (LT)of the system as well. The critical WIP is obtained thanks to the following formula: $WIP* = min(TH)xLT$.

Our analysis result with a lead time of 89,570 seconds and a critical work-in-process of 3.355.

LT [sec]	89,570
min(TH) [pcs/sec]	0,03745
Critical WIP [pcs]	3,355

Then we analyzed the Best-Case Performance, the Worst-Case Performance, the Practical Worst-Case Performance, and the Real-Case Performance computing the throughput rate and throughput time by applying the following formulas:

Best case performance	Worst case performance	Practical case performance
$TTP_{best} = \begin{cases} T_0 \text{ if } w \le W_0 \\ \frac{w}{r_b} \text{ otherwise} \end{cases}$	$TTP_{worst} = w * T_0$	$TTP_{PWC} = T_0 + \frac{w-1}{r_b}$
$TH_{best} = \begin{cases} \frac{w}{T_0} \text{ if } w \le W_0 \\ r_b \text{ otherwise} \end{cases}$	$TH_{worst} = \frac{1}{T_0}$	$TH_{PWC} = \frac{w}{W_0 + w - 1} r_b$

where:

- T_0 is the minimum Lead Time
- R_b is the throughput of the bottleneck
- W_0 is the critical WIP
- w is the number or pieces within the system

The values and the corresponding graphs are as following:

	WIP	1	2	3	3,36	4	5	6	7	8	9	10
Best case	TTP	90	90	90	90	107	134	160	187	214	240	267
	TH	0,011	0,022	0,033	0,038	0,038	0,038	0,038	0,038	0,038	0,038	0,038
Worst case	TTP	90	179	269	301	358	448	537	627	717	806	896
	TH	0,011	0,011	0,011	0,011	0,011	0,011	0,011	0,011	0,011	0,011	0,011
Practical worst case	TTP	90	116	143	153	170	196	223	250	276	303	330
	TH	0,011	0,017	0,021	0,022	0,024	0,025	0,027	0,028	0,029	0,030	0,030

Table 2.1: Theoretical behavior of the AS IS system

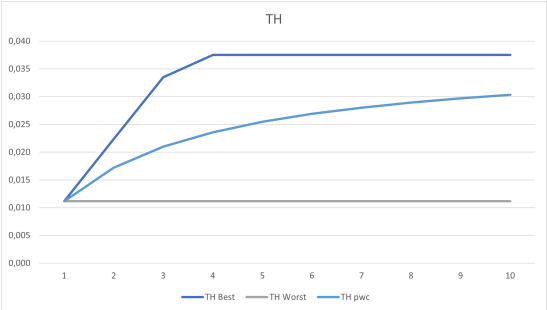


Figure 2.1: Throughput based on Little's law

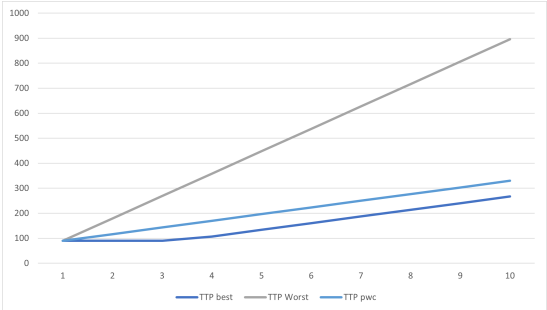


Figure 2.2: Throughput time based on Little's law

2.3. TH and UR of each station running on critical the WIP

Regarding the analysis of Throughput with the critical Work in Process (WIP), it is evident that our system remains consistent without variation.

This observation is particularly noteworthy considering that the critical WIP is lower than the WIP level used in previous model simulations. Despite this difference in WIP levels, the analysis of Throughput remains unchanged. This implies a certain robustness and stability in the system's performance, even when operating under varying levels of work in process.

The concept of 'critical WIP' refers to the optimal amount of work in process that maximizes throughput without causing delays or inefficiencies. In this context, the analysis demonstrates that our system maintains its efficiency and throughput levels effectively, even when the amount of work in process is adjusted to this lower, critical level. This finding is significant for operational planning and for optimizing production processes to achieve the best possible balance between workload and output.

For the analysis, we designated a critical Work in Progress (WIP) value of 3,355. In order to obtain the corresponding utilization rates for each station, the group has decided to use the static "utilization" available within the machines' settings.

'To Workspace' blocks has been connected directly to the Entity Servers, where we enabled the option for UR measurement.

As anticipated before, talking about what server can monitor, the figure below shows how the model has been arranged with the purpose of collecting data about utilization:

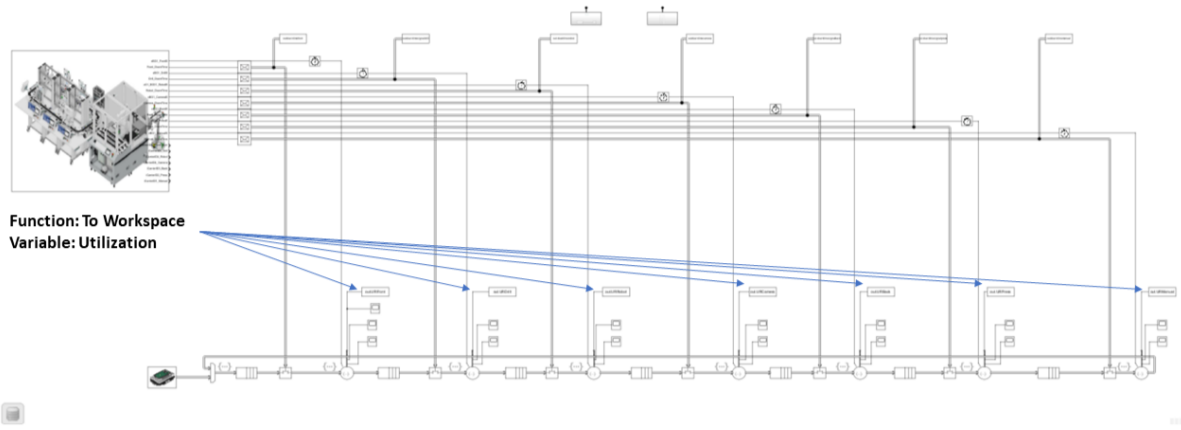


Figure 2.3: Utilization statistic connected

The result are seven time series on MATLAB from which it has been very easy to find the mean value, correspondent to the real utilization rates of the sations.

station	Front	Drill	Robot	Camera	Back	Press	Manual
UR	30.40%	32.29%	78.00%	26.19%	30.36%	29.34	35.00%

Table 2.2: Utilization rates computed on critical WIP

The study subsequently uncovers significant disparities in efficiency and performance between the various stations, most notably the Robot cell. It was observed that the Robot cell exhibits the highest Utilization Rate (UR), thereby identifying it as the bottleneck within the linear production system.

2.4. Analysis of the behaviour of the line (Real case)

With this step, we chose to remove the digital twin from the line and replace the service time source with a MATLAB action with the values of the stations that were presumed to follow different distribution.

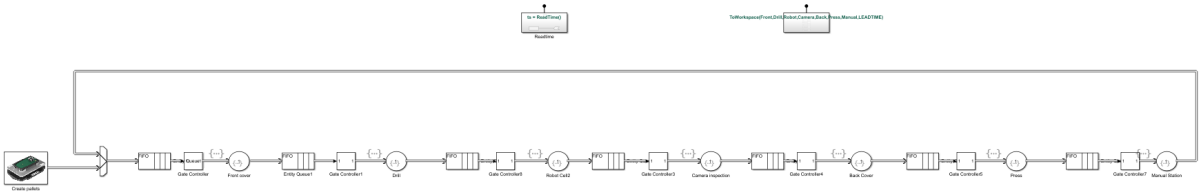


Figure 2.4: Disconnection of the line

The configuration of the servers requires to switch their settings from "signal port" to "MATLAB action" and code the distributions parameters as follows:

Front Cover station

```
mu_front=10.47
sigma_front=1.65
dt = random('Normal',mu_front,sigma_front);
dt=abs(dt);
```

Drill station

```
a = 8.302146469152879;
b = 1.329882644709480;
dt = random('Gamma',a,b);
dt = abs(dt);
```

Robot Cell station

```
mu_robot= 3.23;
sigma_robot= 0.33;
dt = random('Lognormal',mu_robot,sigma_robot);
dt=abs(dt);
```

Camera inspection station

```
A = 10.388380523364347;
B = 2.563316142282167;
```

```
dt = random('Weibull',A,B);
dt = abs(dt);
```

Back cover station

```
mu_back = 10.56;
sigma_back = 1.86;
dt = random('Normal',mu_back,sigma_back);
dt = abs(dt);
```

Press station

```
a = 5.886670011788683;
b = 1.713742281112883;
dt = random('Gamma',a,b);
dt = abs(dt);
```

Manual station

```
A = 13.880477339605644;
B = 1.542381617112172;
dt = random('Weibull',A,B);
dt = abs(dt);
```

This was the setting required in order to let the machine behave (in terms of service time) as they would be still connected to the I4.0 block.

The model has been run 15 times for one shift: each time increasing the number of pallets in the entity generator. We modified the code written inside the last server to extract the lead time of the process. We then computed the TH for each station to find the bottleneck.

In the simulation, it was decided to increase the number of WIP until a constant Throughput Rate (TH) was achieved, in this case rb=0.038 pcs/sec. The minimum throughput time for which this rate was obtained is given by a WIP level of 3.3, according to the formulas used for critical WIP considering minimum throughput time, given by the sum of the long-term average process times (working time) of each station in the line.

As can be seen from the table, after the WIP level 4, the throughput rate follows a linear pattern. Thus, it was concluded that the critical WIP in real case scenarios is 5. The critical WIP is the WIP required for a line with no variability to achieve maximum throughput with the shortest cycle time, implying that the previously described formula can only be employed in a line without any variability. This also explains why the calculated WIP is

different from the observed one.

WIP	LT	TH
1	90,35	0,011
2	95,38	0,021
3	100,58	0,030
4	112,48	0,036
5	132,73	0,038
6	158,12	0,038
7	184,49	0,038
8	210,79	0,038
9	237,19	0,038
10	263,46	0,038
11	289,71	0,038
12	311,85	0,038
13	342,03	0,038
14	368,15	0,038
15	394,23	0,038

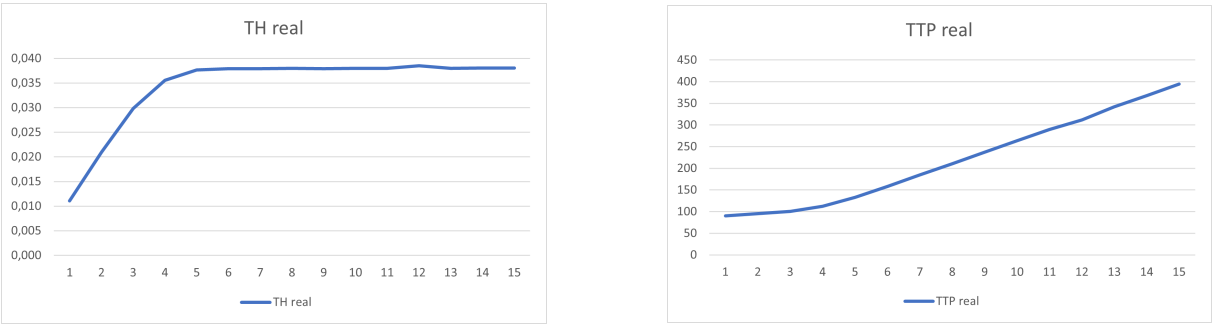


Figure 2.5: Real case performances: TH and TTP

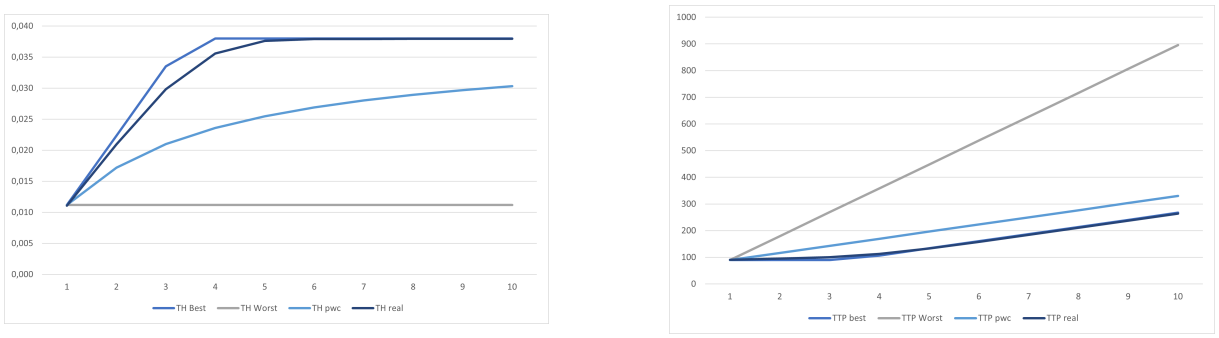


Figure 2.6: Performances comparison

The PWC curve can be understood as a midway to mimic the behavior of the real system, according to theory. This curve can be used to determine if real-world operation conditions are in the region between the best and the practical worst cases or between the practical worst and the worst cases. The real case curve is in the lean area, very close to the best-case performance, as expected because the production does not use batch moves and the coefficients of variation of each station are not too big for the distribution that represents them.

In the graph it is also confirmed that around WIP 4-5 there is a constant trend implying the critical WIP.

3 | System Availability analysis

3.1. Computation of machines' availiability

In order to evaluate the availability of the seven machines constituting our production process, a systematic analysis was conducted based on data acquired through a simulation in Simulink. This analysis aimed to assess the machines' performance and operational effectiveness over a specified duration.

The process unfolded through several key steps:

Data Extraction: The initial step involved extracting pertinent time series data for each machine from the simulation output. This data encompassed information on cumulative downtime and the corresponding operating status for each machine, providing a comprehensive view of their performance. Subsequently, a list comprising the time series data for each machine was created, facilitating a consolidated approach to analysis.

```
% Extract the time series from the Output object
downtimecamera = out.downtimecamera;
downtimefront = out.downtimefront;
downtimemanual = out.downtimemanual;
downtimerobot = out.downtimerobot;
downtimesignalback = out.downtimesignalback;
downtimesignaldrill = out.downtimesignaldrill;
downtimesignalpress = out.downtimesignalpress;

% Set total time
total_time=28800;

% List of time series
time_series_list = {downtimefront, downtimesignaldrill,downtimerobot,...
                    downtimecamera, downtimesignalback,downtimesignalpress ,...
                    downtimemanual};
```

The heart of the analysis involved a systematic iteration through each machine's time series data.

The operating time for each machine was calculated by summing the product of its operating status and the corresponding time differences starting from the second row of every time series as at instant zero all machines are considered stationary.

The availability of each machine was then determined as the ratio of its operating time to the total time and they are collected in the following table.

station	Front	Drill	Robot	Camera	Back	Press	Manual
Availability	74.00%	84.99%	96.99%	100.00%	85.00%	92.98%	96.01%

Table 3.1: Availability of the machines

```
for i = 1:length(time_series_list)
    % Iterate through all time series in the data cell array time_series_list. The variable
    % i represents the index of the current time series.

    % Extract data and time from the current time series
    cumulative_time = time_series_list{i}.Time;
    operating_status = time_series_list{i}.Data;

    % Calculate operating time as the sum of the product
    % of the operating state and the difference between cumulative time values
    operating_time = sum(operating_status(2:end) .* diff(cumulative_time));

    % This line calculates the operating time for the current machine. It does so by
    % summing the product of the operating state and the difference between consecutive
    % cumulative time values. The operating_status(2:end) excludes the first row to align
    % with the time differences

    % Calculate availability for the current machine
    availability = operating_time / total_time;

    % Display reliability for the current machine
    fprintf('Availability of machine %.f: %.2f%%\n', i, availability * 100);

    % The code then displays the availability (reliability) of the current machine in a
    % formatted output using fprintf. The machine index (i) and availability percentage are
    % shown for each iteration of the loop
end
```

3.2. Space table of the system and correspondent table of probabilities

ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7	Probability
1	1	1	1	1	1	1	46,291%
0	1	1	1	1	1	1	16,264%
1	0	1	1	1	1	1	8,175%
1	1	0	1	1	1	1	1,432%
1	1	1	0	1	1	1	0,000%
1	1	1	1	0	1	1	8,169%
1	1	1	1	1	0	1	3,495%
1	1	1	1	1	1	0	1,924%
0	0	1	1	1	1	1	2,872%
0	1	0	1	1	1	1	0,503%
0	1	1	0	1	1	1	0,000%
0	1	1	1	0	1	1	2,870%
0	1	1	1	1	0	1	1,228%
0	1	1	1	1	1	0	0,676%
1	0	0	1	1	1	1	0,253%
1	0	1	0	1	1	1	0,000%
1	0	1	1	0	1	1	1,443%
1	0	1	1	1	0	1	0,617%
1	0	1	1	1	1	0	0,340%
1	1	0	0	1	1	1	0,000%
1	1	0	1	0	1	1	0,253%
1	1	0	1	1	0	1	0,108%
1	1	0	1	1	1	0	0,059%
1	1	1	0	0	1	1	0,000%
1	1	1	0	1	0	1	0,000%
1	1	1	0	1	1	0	0,000%
1	1	1	1	0	0	1	0,617%
1	1	1	1	0	1	0	0,339%
1	1	1	1	1	0	0	0,145%

ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7	Probability
0	**0**	**0**	**1**	**1**	**1**	**1**	**0,089%**
0	0	1	0	1	1	1	0,000%
0	0	1	1	0	1	1	0,507%
0	0	1	1	1	0	1	0,217%
0	0	1	1	1	1	0	0,119%
0	1	0	0	1	1	1	0,000%
0	1	0	1	0	1	1	0,089%
0	1	0	1	1	0	1	0,038%
0	1	0	1	1	1	0	0,021%
0	1	1	0	0	1	1	0,000%
0	1	1	0	1	0	1	0,000%
0	1	1	0	1	1	0	0,000%
0	1	1	1	0	0	1	0,217%
0	1	1	1	0	1	0	0,119%
0	1	1	1	1	0	0	0,051%
1	0	0	0	1	1	1	0,000%
1	0	0	1	0	1	1	0,045%
1	0	0	1	1	0	1	0,019%
1	0	0	1	1	1	0	0,011%
1	0	1	0	0	1	1	0,000%
1	0	1	0	1	0	1	0,000%
1	0	1	0	1	1	0	0,000%
1	0	1	1	0	0	1	0,109%
1	0	1	1	0	1	0	0,060%
1	0	1	1	1	0	0	0,026%
1	1	0	0	0	1	1	0,000%
1	1	0	0	1	0	1	0,000%
1	1	0	0	1	1	0	0,000%
1	1	0	1	0	0	1	0,019%
1	1	0	1	0	1	0	0,010%
1	1	0	1	1	0	0	0,004%
1	1	1	0	0	0	1	0,000%
1	1	1	0	0	1	0	0,000%
1	1	1	0	1	0	0	0,000%
1	1	1	1	0	0	0	0,026%

The process of constructing the state space table and the corresponding table of probabilities involves several key steps. After calculating the availability of each machine, the following procedure is employed: The probability of operation for each of the seven machines is defined, representing the likelihood of each machine being operational. Subsequently, all combinations of operational and failed states for the seven machines are generated using a binary representation. An additional case is added to cover the scenario where all machines are operational.

A cell array is initialized to store the state matrix with probability. The probability of each state is then calculated based on the product of individual machine probabilities for the given state. The cell array representing the state matrix is populated with information about each state and its corresponding probability. This state matrix provides a comprehensive overview of the system's state space, detailing whether each machine is operational or has failed.

In the final step, the constructed state space table, detailing the state (0 = failure, 1 = operational) and its associated probability, is displayed. Additionally, the total probability of the system being in any state is presented for a comprehensive understanding of overall system availability.

The system consists of seven machines, each of which can be in an operational state (1) or a failure state (0). Therefore, the total number of possible cases is given by 27, corresponding to 128 distinct cases. Each case represents a possible configuration of the seven machines. Analyzing the state table, we notice that in instances where there are more than 3 stations operational, the likelihood of such occurrences is nearly zero. To provide a clearer representation of the system states, we have opted to construct a table displaying only those cases where fewer than 4 stations are non-operational.

```
% Probability of operation for the seven machines: Defines a vector
% probability_of_operation containing the probabilities of operation for seven machines.
% The values in the vector represent the probability that each machine operates correctly
probability_of_operation = [0.74, 0.8499, 0.9700, 1.0000, 0.8500, 0.9298, 0.9601];

% Number of machines:Calculates the total number of machines based on the length of the
% probability_of_operation vector.
number_of_machines = length(probability_of_operation);

% Generate all combinations of operation and failure for the seven machines
combinations = dec2bin(0:(2^number_of_machines - 1), number_of_machines) - '0';

% 2^number_of_machines calculates the total number of possible binary combinations of
% operation and failure for the seven machines.
% 0:(2^number_of_machines - 1) generates a sequence from 0 to 2^number_of_machines - 1.
% dec2bin(..., number_of_machines) converts each number in the sequence into a binary
% representation of length number_of_machines.
% (2^number_of_machines - 1), number_of_machines) - '0' converts the binary representation
% into an array of integers (0 or 1), thus creating all possible combinations of operation
% and failure for the seven machines.

% Initialize a cell for the state matrix with probability
state_matrix = cell(size(combinations, 1), 2);

% This line initializes a cell array called state_matrix to store information about each
% state. The cell array has two columns: the first column for the state representation (0
% or 1), and the second column for the probability associated with each state.

% Calculate the probability of each state
probability_states = prod(combinations .* probability_of_operation + (1 - ...
combinations) .* (1 - probability_of_operation), 2);

% This line calculates the probability of each state. It uses the prod function to
% calculate the product of each element along the second dimension of the matrix resulting
% from element-wise multiplication. The multiplication involves combining the probability
% of operation and failure for each machine, based on the provided probabilities.

% Total probability calculation
total_probability = sum(probability_states)

% Populate the cell with information about the states and probabilities
state_matrix(:, 1) = num2cell(combinations, 2);
state_matrix(:, 2) = num2cell(probability_states);

% These lines populate the state_matrix cell array with information about each state and
% its associated probability. The num2cell function is used to convert numerical arrays
% into cell arrays for better organization

% Display the state matrix
disp('State Space:');
disp(['State (0 = failure, 1 = operational) | Probability']);
disp(state_matrix);

% These lines display the state matrix in a readable format, showing the representation of
% each state (0 for failure, 1 for operational) along with its calculated probability

% Display the total probability
fprintf('\n Total Probability: %.4f\n', total_probability);
```

3.3. Production capacity

The initial phase involves calculating the probability of each configuration, considering the operational probabilities of individual machines. The cumulative probability of the system being operational is then computed by summing these individual probabilities. To streamline the analysis, configurations with probabilities below 0.5% are excluded, narrowing down the focus to the most relevant cases. Following this, throughput rates for each machine station are introduced to enhance the accuracy of production capacity calculations. The throughput rate (TR) represents the number of units produced per unit of time, typically measured in items per hour. The next step is to calculate the lead time for each station in the production line. The lead time is the total time it takes for a unit to pass through a specific station, providing insights into the time contribution of each station to the overall production process. Once the lead times are established, the cycle time (CT) for each state is usually determined by selecting the maximum lead time among the functioning stations within that state.

In this case, in the production process operates as a series of successive actions, forming a cycle. Notably, if any one of the seven machines fails, the hourly production does not merely decrease but comes to a complete halt. This phenomenon arises due to the nature of the system, where the product must pass through each station sequentially, and if it fails to reach a subsequent station, the entire production line ceases operation.

To calculate the theoretical hourly production, it suffices to focus on scenarios where all machines are operational, corresponding to the cycle time of the robotic station, the maximum cycle time among all stations. Accordingly, dividing 3600 sec/h by this cycle time, which is 26.7 seconds, we obtain the theoretical hourly production.

Multiplying this by the probability of the scenario where all machines are properly working, the actual (or expected) production capacity is the outcome.

It becomes apparent that the system's availability, defined as the fraction of expected production to theoretical production, aligns with the probability of the state in which all stations are operational, 46.29%. This underlines the crucial relationship between system availability and the probability of a fully operational state in serial productions.

```
% Probability of Operation for seven machines
probability_of_operation = [0.74, 0.8499, 0.9700, 1.0000, 0.8500, 0.9298, 0.9601];

% Number of Machines
number_of_machines = length(probability_of_operation);

% Generate all combinations of operation and failure for the seven machines
combinations = dec2bin(0:(2^number_of_machines - 1), number_of_machines) - '0';

% Threshold for the number of operational machines
operational_threshold = 4;

% Find indices of configurations with at least four operational machines
relevant_configurations = find(sum(combinations, 2) >= operational_threshold);

% Filter relevant information based on the identified configurations
relevant_states = combinations(relevant_configurations, :);

% Sort relevant table based on the number of non-operational machines
[~, sort_idx] = sort(sum(relevant_states, 2), 'descend');
relevant_table = table(relevant_states(sort_idx, :), probability_states(sort_idx), ...
    'VariableNames', {'Configuration', 'Probability'});

% Display the sorted table
disp('Sorted Table:');
disp(relevant_table);

% Display the total probability
fprintf('\nTotal Probability: %.4f\n', total_probability);

% Define the throughput rates of stations from 1 to 7 (pieces/second)
throughput_rates = [0.096, 0.1, 0.037, 0.108, 0.095, 0.099, 0.080];

% Calculate the production capacity for each remaining case (pieces/second)
production_capacity_seconds = probability_states(sort_idx) .* (relevant_states(sort_idx, :) * throughput_rates');

% Convert production capacity from pieces/second to pieces/hour
production_capacity_hours = production_capacity_seconds * 3600; % 3600 seconds in an hour

% Add production capacity to the table
relevant_table.ProductionCapacityHours = production_capacity_hours;
```

4 | Redesigning the FML (TO-BE)

Considering the disconnected system and analyzing the throughput of the individual stations, we observed that the robotic station, initially identified as the bottleneck of the AS-IS system, required strategic enhancement.

To address this, an additional robotic machine was integrated, strategically positioned parallel to the existing one, thereby creating a dual-line operation aimed at boosting efficiency.

This strategic positioning was designed to optimize the flow of operations.

Subsequently, a comprehensive throughput analysis was reconducted on the revised system layout, taking into account the impact of this addition.

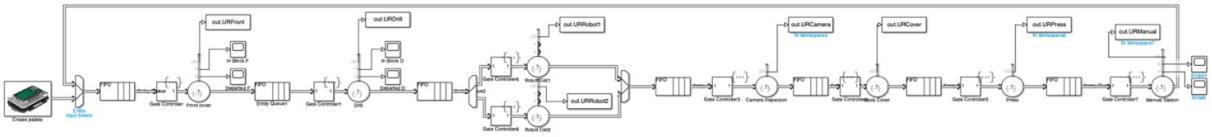


Figure 4.1: Doubled robot stations to improve the throughput

4.1. Bottleneck identification

From the meticulous analysis that ensued, it was evident that the throughput of the robotic station had effectively doubled.

This marked improvement was in alignment with the objectives of the new model, which necessitated a 35%-45% increase in the system's overall throughput. The updated system now operates at a throughput of 0.075 pallets per second.

Despite this enhancement, the robotic station, interestingly, continues to represent the system's bottleneck.

This ongoing challenge is substantiated not only by the theoretical utilization rates of the stations but also by the practical rates derived from detailed simulation models. In

a significant stride towards operational excellence, the system now achieves a much more favorable balance of workloads. This enhanced balance translates into a markedly more efficient operational state than what was observed in the initial AS-IS model.

A noteworthy observation is the system's approach towards saturation capacity, which now extends beyond the previously identified bottleneck stations to include even those stations that were not initially considered as bottlenecks.

Furthering our analysis, the throughput of the identified bottleneck station prompted a focused study on the critical Work in Process (WIP). This in-depth analysis, considering the new operational dynamics, revealed a substantial increase in the critical WIP, which now approximates 6.709. This figure nearly doubles the initial measurements, highlighting the significant impact of the system modifications on the overall processing capability and throughput efficiency.

station	Front	Drill	Robot x 2	Camera	Back	Press	Manual
Process time [sec/pcs]	10,470	10,040	26,700	9,220	10,560	10,090	12,490
TH [pcs/sec]	0,096	0,100	0,075	0,108	0,095	0,099	0,080
UR	76.37%	80.90%	96.40%	65.78%	75.24%	72.55%	87.81%

LT [sec]	89,570
min TH [pcs/sec]	0,07491
Critical WIP [pcs]	6,709

4.2. Throughput analysis TO-BE

From this point, we conducted a comprehensive analysis of the system, considering scenarios of best-case, worst-case, and practical worst-case.

To achieve this, we utilized the data gathered thus far and considered a range of Work in Process (WIP) from one to fourteen pieces. This thorough approach enabled us to determine the system's performance under various hypothetical conditions.

By exploring these scenarios, we aimed to understand the system's resilience and capacity to handle different levels of workload and stress. The analysis was meticulously structured to provide a holistic view of the system's capabilities and limitations under a spectrum of operational circumstances. The findings from this investigation yielded the following results:

from 1-7	WIP	1	2	3	4	5	6	6,7	7
Best case	TTP best	90	90	90	90	90	90	90	93
	TH Best	0,011	0,022	0,033	0,045	0,056	0,067	0,075	0,075
Worst case	TTP Worst	90	179	269	358	448	537	600	627,0
	TH Worst	0,011	0,011	0,011	0,011	0,011	0,011	0,011	0,011
Practical worst case	TTP pwc	90	103	116	130	143	156	166	170
	TH pwc	0,011	0,019	0,026	0,031	0,035	0,038	0,040	0,041
from 7-14	WIP	7	8	9	10	11	12	13	14
Best case	TTP best	93	107	120	134	147	160	174	187
	TH Best	0,075	0,075	0,075	0,075	0,075	0,075	0,075	0,075
Worst case	TTP Worst	627	717	806	896	985	1075	1164	1254
	TH Worst	0,011	0,011	0,011	0,011	0,011	0,011	0,011	0,011
Practical worst case	TTP pwc	170	183	196	210	223	236	250	263
	TH pwc	0,041	0,044	0,046	0,048	0,049	0,051	0,052	0,053

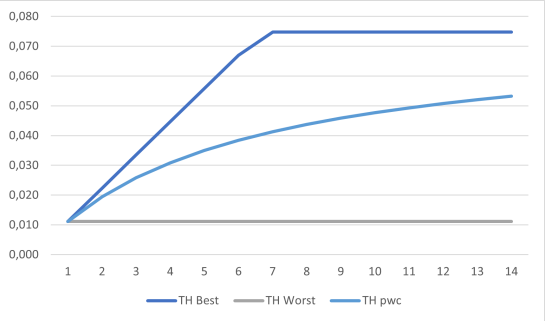


Figure 4.2: Throughput based on Little's law (TO-BE)

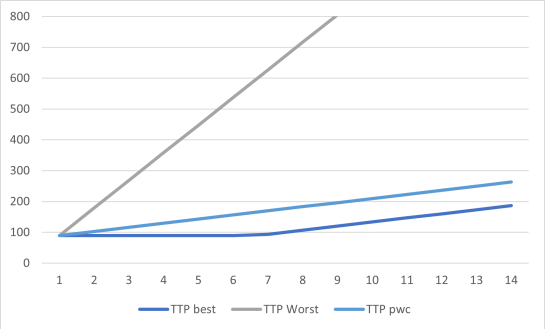


Figure 4.3: Throughput time based on Little's law (TO-BE)

4.3. Analysis of the behavior of the line TO-BE (real-case)

To conduct a real-case analysis of the system, we initiated a series of simulations, running the system a total of 17 times.

During these runs, we varied the number of pallets in the entity generator, mirroring the conditions of the AS-IS model. This methodical approach allowed us to observe the system's behavior under a range of operational conditions, providing a comprehensive understanding of its real-world performance.

As illustrated in the table, our system in the real-case scenario stabilizes with a throughput equivalent to that of the bottleneck station from 11 WIP (Work in Process) onwards. This finding is consistent with the trends observed in the best-case and practical worst-case scenarios.

However, it's noteworthy that in the real-case scenario, the system's lead time increases at a slower rate compared to the AS-IS model. This improvement in lead time can be attributed to the distribution of workload across the two machines at the robotic station, which has effectively mitigated the saturation imbalance prevalent in the initial system setup.

The successful implementation of this dual-machine strategy at the robotic station not only enhances throughput but also contributes to a more equitable distribution of workload. This results in a more streamlined process flow, reducing bottlenecks and improving overall system efficiency. The data gathered from these simulations provide valuable insights into the system's capacity for handling varying levels of demand, enabling further optimization and strategic planning for future operational scalability.

WIP	LT	TH
1	90,35	0,011
2	92,81	0,022
3	95,45	0,031
4	97,63	0,041
5	100,86	0,050
6	105,81	0,057
7	110,69	0,063
8	117,8	0,068
9	125,65	0,072
10	136,64	0,073
11	**147,24**	**0,075**
12	160,38	0,075
13	173,45	0,075
14	186,55	0,075
15	199,47	0,075
16	212,7	0,075
17	225,74	0,075

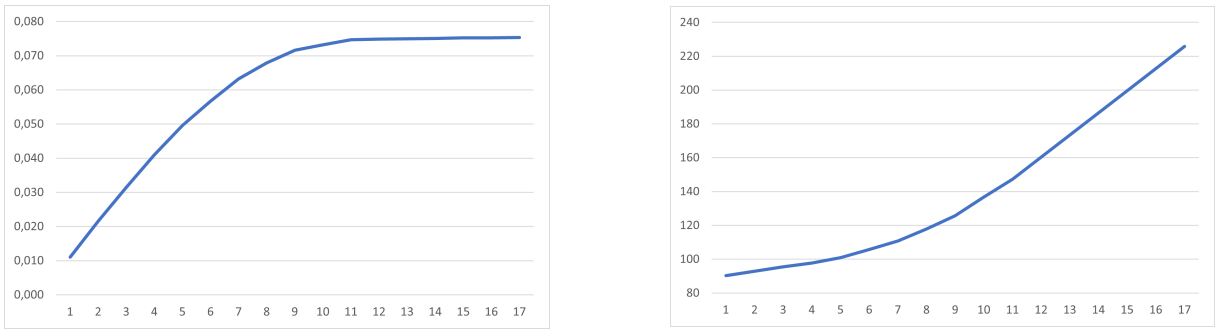


Figure 4.4: TO-BE Real case performances: TH and TTP

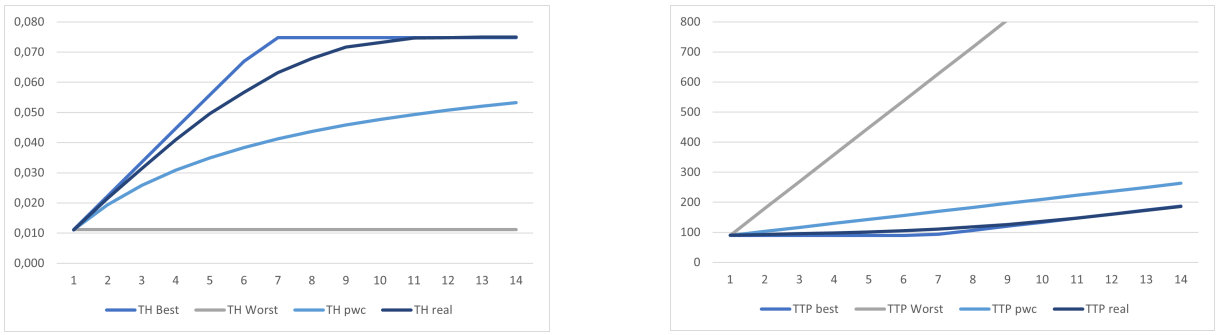


Figure 4.5: Performances comparison (TO-BE)

In the AS-IS model, the real-case scenario curves are positioned between the best case and the practical worst-case, highlighting how the new system closely approximates, and almost reaches, the best-case performance. This observation suggests a significant improvement in the system's efficiency in the real-case scenario compared to the initial AS-IS model.

The proximity of the real-case performance to the best-case scenario indicates that the optimizations implemented in the new system design have been highly effective.

These improvements are likely due to strategic modifications such as the redistribution of workload, enhanced automation, or the introduction of more efficient processes. As a result, the new system demonstrates a level of operational efficiency and effectiveness that approaches the ideal conditions represented by the best-case scenario.

Moreover, the fact that the real-case curves are notably distanced from the practical worst-case scenario underscores the resilience of the new system against potential operational challenges. It reflects the system's robustness and its ability to maintain high efficiency

levels even under less-than-ideal conditions.

This improved performance in the real-case scenario is a testament to the success of the system redesign and offers valuable insights for further enhancements and future system development strategies.

4.4. Redesign the FML based on the initial System Availability Analysis

To enhance the line's availability, we identified the station with the lowest availability, namely the front cover station I (74%). The team decided to duplicate Station 1 and set the availability of the duplicate to 88.8%, according to the constraints given by the project guidelines.

Since the bottleneck wouldn't change and keep remaining the Robot cell, we consciunsly decided to redesign the "To-Be model" based on a standalone analysis of the availabilities.

Afterwards, in the next section we will be able to easily combine the two redesigned model in one, aware that the theoretical production capacity will be given by the doubled Robot cells, so from the output of the latter perfomance analysis. The purpose will be to show the final actual production capacity and the final availability, but for this section just focus on the doubled Front cover station model shown below:

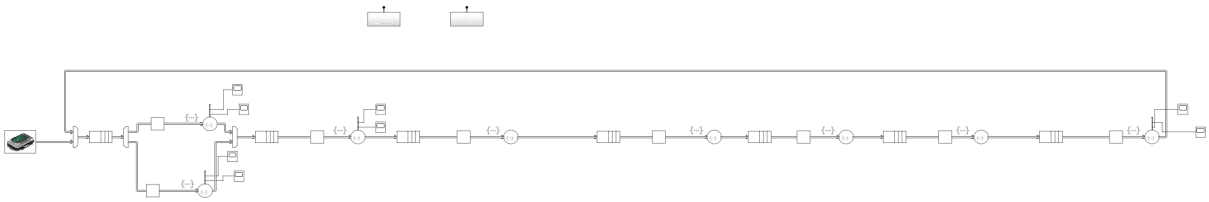


Figure 4.6: Doubled Front Cover station to improve the availability

This redesign resulted in a new state table with eight states (0 or 1) and, consequently, 2^8 combinations.

The MATLAB code for calculating the new state table remains identical, except for the fact that when creating the vector for the probability of operation of the stations, we will add the value associated with the new front cover station (0.888).

This process aimed to boost the line's availability, as opposed to the previous scenario where only the state with all stations functioning contributed to production. Now, even

states where one of the two front cover stations is not operational contribute to production.

Again for the calculation of the actual production capacity the processing time of bottleneck, so the cycle time, has been exploited. We have taken the same theoretical production capacity since doubling up just the Front Cover station involves that the Robot cell keep being the station responsible for congestion.

St_1	St_1BIS	St_2	St_3	St_4	St_5	St_6	St_7	Probability	TPC [pcs/h]	APC [pcs/h]
1	1	1	1	1	1	1	1	44,90%	135.175	60,69
0	1	1	1	1	1	1	1	15,78%	135.175	21,33
1	0	1	1	1	1	1	1	7,93%	135.175	10,72

The new actual hourly production capacity of the line is 92.74 pieces per hour.

Dividing this by the maximum production capacity of the bottleneck (135.175 pcs/hour), the line's availability is determined to be 68.61%.

Such increase in availability confirms the need of Front Cover station parallelization. For this reason, in the following section we will try to combine the benifits of adding a new Front Cover station and a new Robot cell redesigning the final model TO-BE.

4.5. Final model TO-BE

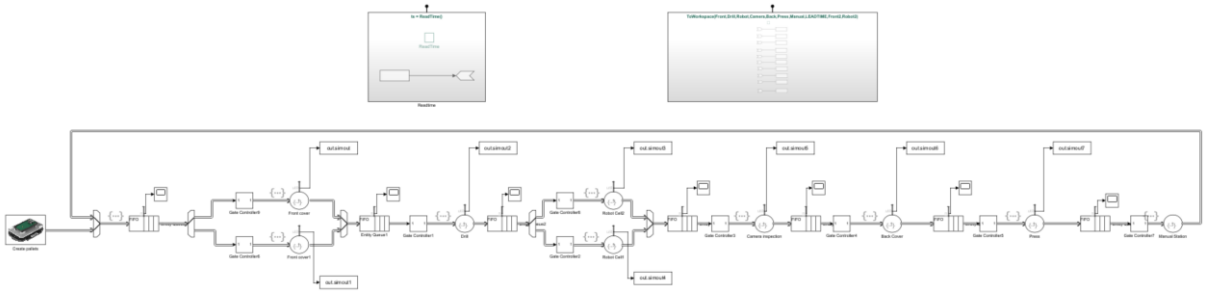


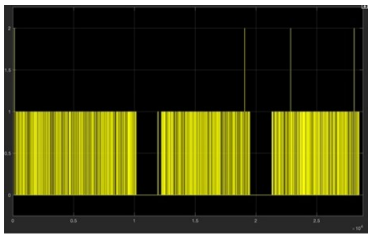
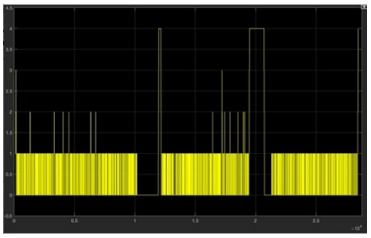
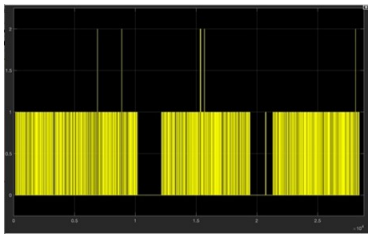
Figure 4.7: TO-BE final version

St_1	St_1.2	St_2	St_3	St_3.2	St_4	St_5	St_6	St_7	Probability	PC [pcs/h]
1	1	1	1	1	1	1	1	1	39,873%	270,3515
0	1	1	1	1	1	1	1	1	14,009%	270,3515
1	0	1	1	1	1	1	1	1	5,029%	270,3515
1	1	1	0	1	1	1	1	1	1,233%	135,1757
1	1	1	1	0	1	1	1	1	1,233%	135,1757
0	1	1	0	1	1	1	1	1	0,433%	135,1757
0	1	1	1	0	1	1	1	1	0,433%	135,1757
1	0	1	0	1	1	1	1	1	0,156%	135,1757
1	0	1	1	0	1	1	1	1	0,156%	135,1757

Regarding the final performance, we can conclude that the theoretical production capacity (per hour) is equal to 270.35 pcs/h. By using the usual method for calculating the actual production capacity, we get 164.19 pcs/h.

4.6. Buffer sizing

Upon conducting a detailed queue analysis on the rebalanced model, we assessed the critical throughput. This was executed by applying scopes to the queues, which enabled us to monitor the Work-In-Progress (WIP) levels at every point during the simulation. The data obtained from the simulation illustrated distinct trends in the workflow, which are represented in the accompanying charts, organized according to the parts' routing sequence.

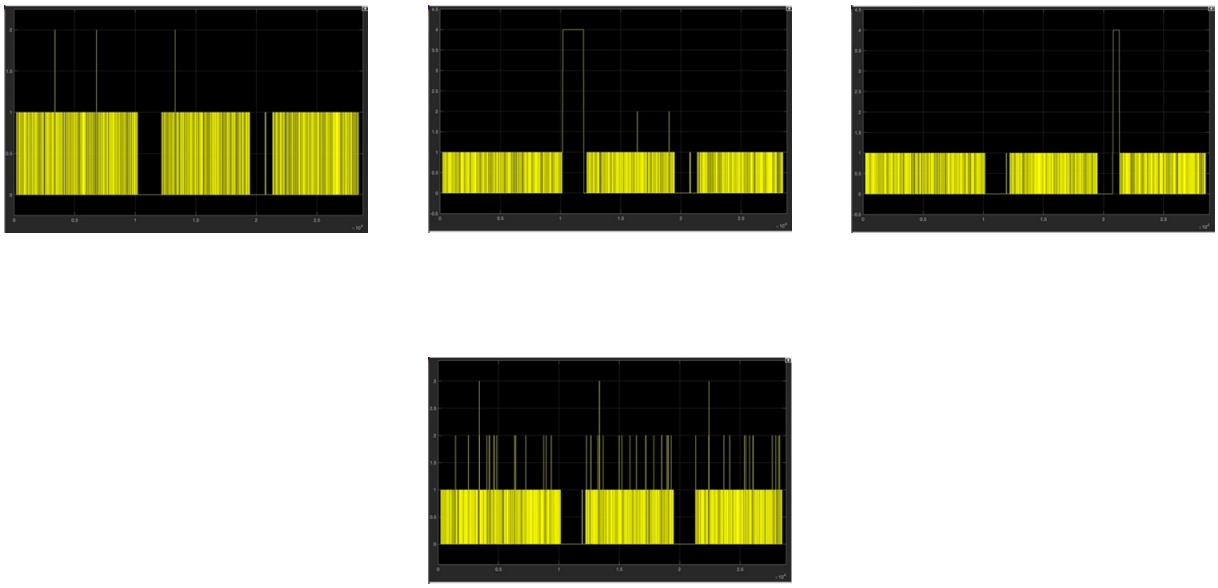


Figure 4.8: Required buffers capacities

Our observations from the scope analysis highlighted that the occupancy of the buffers varied significantly, which was directly influenced by the processing times at the preceding station. In an effort to streamline our production line, we were tasked with the challenge of allocating no more than 24 spaces across all the buffers in the line. The simulation outcomes were promising, showing that we succeeded in configuring the buffers to operate below the maximum capacity threshold. The distribution of spaces was carried out as depicted in the table included below, with an aggregate of 21 spaces being effectively used.

Front	Drill	Robot	Camera	Back	Press	Manual	TOTAL
2	4	2	2	4	4	3	21

Table 4.1: Queueing capacity

This strategic buffer sizing approach serves multiple purposes. It not only optimizes the physical spaces within our production line but also ensures a lean operational model. However, a limitation of this approach is its reduced capacity to handle potential fluctuations in processing times across different stations. This could impact our ability to respond to sudden changes in production volume or unexpected downtimes. The simulation charts and the space allocation table are crucial for visualizing the efficiency of our current model and will guide future adjustments to our production process. They will be included in the formal report to provide a comprehensive view of our operational enhancements and the potential areas for further improvement.

5 | Conclusions

The transition from the AS-IS to the TO-BE model represents a transformative journey in the realm of advanced manufacturing simulation. The AS-IS model laid the groundwork, showcasing a system capable of adapting to varied product configurations but limited by the bottleneck of slower processing times at the robotic station. Building upon these insights, the TO-BE model was conceived with a focus on amplifying throughput, availability, and utilization rates. The strategic doubling of the robotic station, informed by meticulous statistical analysis, allowed for a remarkable enhancement in system productivity. This was a calculated decision that ensured the continued status of the robotic cell as the bottleneck, thus maximizing the return on investment by fully leveraging the performance benefits without shifting the system's critical point of congestion.

Further analyses within the TO-BE model revealed the Front Cover station as having the lowest availability, prompting a duplication of this station as well. The resulting increase in actual productive capacity by a significant margin justified this investment. This careful, stepwise approach to system redesign—first by addressing the throughput via the robotic station and then by enhancing availability with the duplication of the Front Cover station—allowed for an incremental yet impactful progression towards a highly efficient production line.

The combined effect of these strategic enhancements in the TO-BE model is a testament to the power of data-driven decision-making in manufacturing simulations. By paralleling critical stations, the system not only witnessed a surge in performance but also achieved a more balanced utilization across all stations, addressing the previously noted imbalances.

In summary, the research encapsulates a narrative of progress, from a solid foundation in the AS-IS model to a significantly improved TO-BE model. The latter not only promises a notable increase in productivity but also points to the vast potential of the line when both critical stations are duplicated.