



# **CMPE412: Computer Simulation**

## **Project 2 - Manufacturing System**

**Submitted by:**

Enise Nur Yılmaz – 20191791934 (CE)

**Project Mentor:**

Dr. Doğan Çörüş

**Faculty of Engineering and Natural Sciences**

**Kadir Has University**

**Spring 2024**

## TABLE OF CONTENTS

1 INTRODUCTION .....	3
2 METHODOLOGY .....	4
2.1 Library that is Used with Python & Defined Variables .....	4
2.2 Simulation Parts.....	6
2.2.1 Manufacturing Line Class .....	6
2.2.2 Operation Process Class .....	7
2.2.3 Machine Repairing Class.....	8
2.2.4 Manufacturer Process Class.....	8
2.2.5 Setup Class .....	9
2.3 Outputs of Simulation .....	10
3 CONCLUSION .....	11
REFERENCES.....	12

# 1 INTRODUCTION

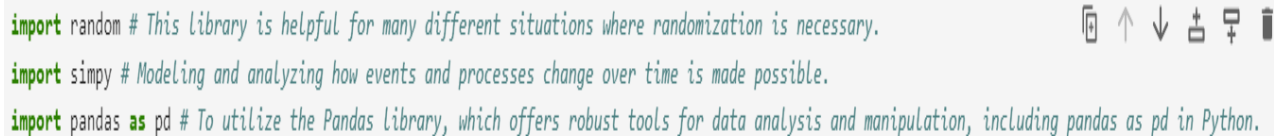
The second of our project for the Computer Simulation course is called production. This project aims to examine methods of developing a discrete event simulation of production. On an extensive automobile part production line, real-world examples will be made for this purpose. Modeling will be done with the samples created. Initially, work will be done on a single product line, then work will be done to make this project more efficient and simpler. As with every project, this project also has its own specific requirements, and within these requirements, a project that will cover more than one product type is created. This formation will be the basis of the simulation. It is created through many stages such as assembly, quality control, painting and packaging between production lines. Each of the production lines has special teams, and a shift system is applied within these teams. According to the shift system, production is continuous and there are teams that complete this continuity throughout the day. In this way, the need for machines and labor arises in the production line. In this context, in order to create the mentioned simulation, it is necessary to create an efficient optimization of the line process of a single product. While performing the optimization application, possible problems and possible expenses that may be experienced during production should also be evaluated. With the analyzes obtained in this direction, possible damages should be reduced. As a result, more than one product type will be considered in the simulation, in addition, potential harms should be analyzed in terms of complexity and resource allocation. There will also be specifications for the steps involved in discrete event simulation, including event preparation, time progression, and status updates. While performing all these analyses, Python language was used in this report. Additionally, SimPy [1] library is used to create this simulation production line. An event-based simulation package called SimPy [1] makes it simple to simulate how systems behave over an extended period.

## 2 METHODOLOGY

Raw materials, intermediate parts, stages of completed products as the basic steps of the project; Modeling will be done by identifying processes such as loading, processing, assembly, control and packaging, as well as machines and operators. Subsequently, the rates of raw material needs, machinery, and working hours will be decided based on the forecasts. In addition, factors like the likelihood of recurring production equipment problems and maintenance intervals will be considered. A component of the simulation will be these evaluations. Limits will be established using the basic input variables supplied, and potential scenarios will be experienced through a single product. This means that in addition to evaluating options, performance evaluation can be simulated. In the simulation, variable assignments will be made for more than one product and the results will be analyzed.

### 2.1 Library that is Used with Python & Defined Variables

Python programming language was utilized as the programming tool in this project. The project was created using simPy, pandas and random libraries. Also, as Figure 2.1 shows, these libraries' explanations are in the comments.



```
import random # This library is helpful for many different situations where randomization is necessary.
import simpy # Modeling and analyzing how events and processes change over time is made possible.
import pandas as pd # To utilize the Pandas library, which offers robust tools for data analysis and manipulation, including pandas as pd in Python.
```

**Figure 2.1:** Python Libraries that are used in Manufacturing Project

The project's production line is made up of several steps that a product goes through from raw materials to finished goods. The production line has five primary stages of this project.

1. Loading: Supplying the production line with necessary supplies is known as loading.
2. Machining: Creating precise components from unprocessed materials.
3. Assembling: Components to create semi-finished or finished goods are known as an assembly.
4. Inspecting: Items undergoing assurance of quality control procedures are inspected.

5. Packaging: Preparing the finished goods for transportation involves packaging them.

```
durationOfProcess = {  
  'loading': 5,  
  'machining': 10,  
  'assembling': 8,  
  'inspecting': 6,  
  'packaging': 4  
}
```

**Figure 2.2:** Five Primary Stages for Production Line

As in Figure 2.2, the basic components of the production line process are defined with 'loading', 'machining', 'assembling', 'inspecting', 'packaging'.

```
periodOfMaintaining = 3  
rateOfDamage = 0.1
```

**Figure 2.3:** Maintaining Period and Damage Rate

As shown in Figure 2.3, 'rateOfDamage' and 'periodOfMaintaining' variables were specified to take into account the possibility that the machines malfunction during production. This period was determined as 3 minutes and the probability of failure was determined as 0.1, that is, 10%.

```
quantityOfEmployees = {  
  'loading': 2,  
  'machining': 3,  
  'assembling': 4,  
  'inspecting': 2,  
  'packaging': 3  
}
```

**Figure 2.4:** Quantity of Employees

Figure 2.4 shows us that it is a class created to indicate how many employees are involved in each production stage. So it shows the quantity of employees for per implementation area.

```
durationOfShift = 8
timeOfSimulation = 100
productQuantity = 2
```

**Figure 2.5:** Defining of Variables

In Figure 2.5, we assign the shift times of each employee, here it is determined as 8. The determination of this hour in the simulation is due to the fact that production factories continue to produce in 8-hour shifts. Additionally, the duration of the simulation and the number of products produced are also stated.

## 2.2 Simulation Parts

### 2.2.1 Manufacturing Line Class

```
# It defines the line of manufacture period.
class routeofManufacturing:
    def __init__(self, env):
        self.env = env
        self.periods = {
            period: simpy.Resource(env, capacity=quantityOfEmployees[period]) for period in quantityOfEmployees
        }
        self.repair_crew = simpy.Resource(env, capacity=2)
        self.data = []
```

**Figure 2.6:** Manufacturing Line Class

In line with the purpose of the simulation, we can write functions that enable the simulation to occur after assigning variables that cover the requirements of the project. First, we created the manufacturing line class. In this class, the production route is prioritized with the `__init__` method, as shown in Figure 2.6, and thus simpy resources are defined for each period.

## 2.2.2 Operation Process Class

```
# It defines the operations of processing period.
def operationOfProcess(self, part, product_category):
    periods = ['loading', 'machining', 'assembling', 'inspecting', 'packaging']
    for period in periods:
        duration_process = durationOfProcess[period] * (1 if product_category == 1 else 1.2)
        with self.periods[period].request() as request:
            start = self.env.now
            yield request
            try:
                yield self.env.timeout(duration_process)
                if random.random() < rateOfDamage:
                    yield self.env.process(self.machineUpkeep(period))
            finally:
                finish = self.env.now
                self.data.append({
                    'Part': part,
                    'Period': period,
                    'Start': start,
                    'Finish': finish,
                    'Duration': finish - start,
                    'Product Category': product_category
                })
    except simpy.Interrupt:
        yield self.env.process(self.machineUpkeep(period))
```

**Figure 2.7:** Operation Process Class

In Figure 2.7, a class that covers the processing process of the operation is defined. In this class, each production period is considered and the required process steps, start and finish times for each product quantity are calculated together with the variables. The duration is extracted from the durationOfProcess class for every period in periods. Product category is used to compute duration once a condition is applied. Next, for every period, a resource request is made (self.periods[period].request()). Upon approval of the resource request, the procedure (at self.env.now time) begins. Following the completion of the procedure (self.env.timeout(duration\_process)), random damage is examined. The self.machineUpkeep(period) function is invoked if damage occurs (random.random() < rateOfDamage). The process end time is noted once the process (self.env.now) is finished.

### 2.2.3 Machine Repairing Class

```
def machineUpkeep(self, period):  
    with self.repair_crew.request() as repair:  
        start_repair = self.env.now  
        yield repair  
        yield self.env.timeout(periodOfMaintaining)
```

**Figure 2.8:** Machine Repairing Class

Figure 2.8 contains a set of functions depicting the machine repair process. As mentioned in the previous function, if a repair request comes to the repair team, this process starts and then how long this process takes is calculated.

### 2.2.4 Manufacturer Process Class

```
def manufacturerProcess(env, line, part_id, product_category):  
    yield env.process(line.operationOfProcess(part_id, product_category))
```

**Figure 2.9:** Manufacturer Process Class

In this part, the pipeline is created and processed according to product number and category as shown in Figure 2.9. It is then added to the transaction operation environment.



## 2.2.5 Setup Class

```
def setup(env, num_parts_per_type):
    # Manufacturing line is created.
    line = routeofManufacturing(env)

    # For product categories, this is the transaction part.
    for product_category in range(1, productQuantity + 1):
        # It is the part production process initiation part for each product category.
        for i in range(num_parts_per_type):
            part_id = f"Part_{i + 1}"
            env.process(manufacturerProcess(env, line, part_id, product_category))

    # Waiting for the duration of the simulation.
    yield env.timeout(timeOfSimulation)

    # This is the part of extracting data from the production line and creating a DataFrame.
    df = pd.DataFrame(line.data)

    # This is the part of rotating the DataFrame.
    return df

# This is the part of creating the SimPy environment.
env = simpy.Environment()
# This is the part of starting the setup function as a process.
process = env.process(setup(env, 5))
# It runs the simulation
env.run()
# We take the DataFrame obtained as a result of the process and print it.
df = process.value
print(df)
```

**Figure 2.10:** Setup Class for Simulation

As stated in Figure 2.10, first the setup function is defined and the production line is created. After this line is created, the category of each product is processed. Here, the production category can be 1 or more than 1. Then, the production category is prioritized for each production process. Afterwards, the downtime is determined for each production stage. Moreover, after the operations, information about the production line is obtained with the DataFrame function. This DataFrame is then returned inside the function. The process is created by creating an environment thanks to the Simpy library. Afterwards, for each process, the environment is sorted according to 5 different steps. As a result, each resulting value is printed.

## 2.3 Outputs of Simulation

	Part	Period	Start	Finish	Duration	Product	Category
0	Part_1	loading	0.0	5.0	5.0		1
1	Part_2	loading	0.0	5.0	5.0		1
2	Part_3	loading	0.0	10.0	10.0		1
3	Part_4	loading	0.0	10.0	10.0		1
4	Part_1	machining	5.0	15.0	10.0		1
5	Part_2	machining	5.0	15.0	10.0		1
6	Part_5	loading	0.0	15.0	15.0		1
7	Part_1	loading	0.0	16.0	16.0		2
8	Part_2	loading	0.0	21.0	21.0		2
9	Part_3	loading	0.0	22.0	22.0		2
10	Part_1	assembling	15.0	23.0	8.0		1
11	Part_2	assembling	15.0	23.0	8.0		1
12	Part_3	machining	10.0	23.0	13.0		1
13	Part_4	machining	10.0	25.0	15.0		1
14	Part_5	machining	15.0	25.0	10.0		1
15	Part_5	loading	0.0	28.0	28.0		2
16	Part_1	inspecting	23.0	29.0	6.0		1
17	Part_2	inspecting	23.0	29.0	6.0		1
18	Part_4	loading	0.0	30.0	30.0		2
19	Part_3	assembling	23.0	31.0	8.0		1
20	Part_4	assembling	25.0	33.0	8.0		1
21	Part_5	assembling	25.0	33.0	8.0		1
22	Part_1	packaging	29.0	33.0	4.0		1
23	Part_2	packaging	29.0	33.0	4.0		1
24	Part_1	machining	16.0	35.0	19.0		2
25	Part_2	machining	21.0	37.0	16.0		2
26	Part_3	machining	22.0	37.0	15.0		2
27	Part_3	inspecting	31.0	37.0	6.0		1
28	Part_4	inspecting	33.0	39.0	6.0		1
29	Part_3	packaging	37.0	41.0	4.0		1
30	Part_5	inspecting	33.0	43.0	10.0		1
31	Part_4	packaging	39.0	43.0	4.0		1
32	Part_1	assembling	35.0	44.6	9.6		2
33	Part_2	assembling	37.0	46.6	9.6		2
34	Part_5	machining	28.0	47.0	19.0		2
35	Part_5	packaging	43.0	47.0	4.0		1
36	Part_4	machining	30.0	49.0	19.0		2
37	Part_3	assembling	37.0	49.6	12.6		2
38	Part_1	inspecting	44.6	51.8	7.2		2
39	Part_2	inspecting	46.6	53.8	7.2		2
40	Part_1	packaging	51.8	56.6	4.8		2
41	Part_5	assembling	47.0	56.6	9.6		2
42	Part_2	packaging	53.8	58.6	4.8		2
43	Part_4	assembling	49.0	58.6	9.6		2
44	Part_3	inspecting	49.6	59.0	9.4		2
45	Part_5	inspecting	56.6	63.8	7.2		2
46	Part_3	packaging	59.0	63.8	4.8		2
47	Part_4	inspecting	58.6	66.2	7.6		2
48	Part_5	packaging	63.8	68.6	4.8		2
49	Part_4	packaging	66.2	71.0	4.8		2

**Figure 2.11:** Outputs of Simulation

The outputs of the manufacturing simulation are shown in Figure 2.11.

### **3 CONCLUSION**

According to the outputs of the simulation, situations such as the duration of each operation performed within the process of the part production line and possible dangers were taken into consideration. Additionally, these data were analyzed using Python libraries and were useful for evaluation. Thanks to the evaluations made, we can conclude that it is an open source for balancing the relationship of each production stage with each other and detecting possible damages in order to strengthen the efficiency and sustainability of the production line we simulated. As a result, thanks to manufacturing simulation, we can better understand production line processes and identify improvements that can be made in this direction. In this way, simulations provide optimization opportunities for factories that provide real production. In this way, existing basic resources are used in the most efficient way [2].

## REFERENCES

- [1] Matloff, N. (2011). A discrete-event simulation course based on the SimPy language. Retrieved from <http://heather.cs.ucdavis.edu/~matloff/simcourse.html>
- [2] Azadeh, N., Ali, A., & Sheikhalishahi, M. (2014). An efficient computer simulation-based approach for optimization of complex polling systems with general arrival distributions. *Simulation*, 90, 1346-1359. <https://doi.org/10.1177/0037549714556018>