CMPE412-S01-2023.2 COMPUTER SIMULATION

Project 2

**Manufacturing System**



*EBRU BEYZA ÖZEN*

**Faculty of Engineering and Natural Sciences**

**Kadir Has University**

**June 2024**

Contents

[AIM 3](#_Toc169144866)

[METHODOLOGY 3](#_Toc169144867)

[SIMULATION CODE 3](#_Toc169144868)

[ANALYSIS OF SIMULATION CODE 6](#_Toc169144869)

[CONCLUSION 7](#_Toc169144870)

# AIM

This project aims to develop a discrete event simulation to model and optimize a high-volume automotive parts production line. The simulation models a production line that includes processing, assembly, quality control and packaging stages. The aim is to increase efficiency in a single product line, identify bottlenecks and analyze the effects of operational variables. The simulation developed using SimPy was run on various scenarios to evaluate the performance of the production process.

This report presents a project involving the simulation of a high-volume automotive parts production line. The project aims to model processes on the production line, analyze them through simulation and optimize performance. In this context, various stages in the production line and the resources used in these stages were examined in detail.

# METHODOLOGY

Discrete event simulation was used as the simulation method. The simulation was developed using the Python programming language and the SimPy library. The production line is divided into processing, assembly, quality control and packaging stages. Specific resource capacity and processing times are defined for each stage. In addition, failure rates and maintenance times of the machines are also included in the model.

# SIMULATION CODE

import simpy

import random

import pandas as pd

class ManufacturingLine:

def \_\_init\_\_(self, env):

self.env = env

self.machine = simpy.Resource(env, capacity=10)

self.assembly = simpy.Resource(env, capacity=5)

self.inspection = simpy.Resource(env, capacity=3)

self.packaging = simpy.Resource(env, capacity=2)

self.shift\_end = 8\*60\*60

self.results = []

def process(self, name, time):

start\_time = self.env.now

yield self.env.timeout(time)

end\_time = self.env.now

self.results.append((name, start\_time, end\_time))

print(f'{name} completed at {end\_time}')

def machine\_process(env, line, name):

with line.machine.request() as request:

yield request

if random.random() < 0.1: # %10 arıza oranı

yield env.timeout(30\*60) # 30 dakika bakım süresi

yield env.process(line.process(f'Machining {name}', 30\*60))

def assembly\_process(env, line, name):

with line.assembly.request() as request:

yield request

yield env.process(line.process(f'Assembling {name}', 20\*60))

def inspection\_process(env, line, name):

with line.inspection.request() as request:

yield request

yield env.process(line.process(f'Inspecting {name}', 10\*60))

def packaging\_process(env, line, name):

with line.packaging.request() as request:

yield request

yield env.process(line.process(f'Packaging {name}', 15\*60))

def product\_process(env, line, name):

yield env.process(machine\_process(env, line, name))

yield env.process(assembly\_process(env, line, name))

yield env.process(inspection\_process(env, line, name))

yield env.process(packaging\_process(env, line, name))

def production\_line(env, line):

i = 0

while env.now < line.shift\_end:

i += 1

env.process(product\_process(env, line, f'Product {i}'))

yield env.timeout(random.randint(20\*60, 40\*60)) # Ham madde girişi sıklığı

env = simpy.Environment()

line = ManufacturingLine(env)

env.process(production\_line(env, line))

env.run(until=8\*60\*60)

# Simülasyon sonuçlarını analiz etme

df = pd.DataFrame(line.results, columns=['Process', 'Start Time', 'End Time'])

print(df.describe())

# İşlemler arası geçiş süreleri ve performans değerlendirmesi

df['Duration'] = df['End Time'] - df['Start Time']

print(df)

# Performans metriklerini hesaplama

average\_time\_per\_process = df['Duration'].mean()

total\_products\_completed = len(df['Process'].unique())

print(f'Average time per process: {average\_time\_per\_process}')

print(f'Total products completed: {total\_products\_completed}')

# ANALYSIS OF SIMULATION CODE

* Libraries Used

simpy: A Python library used for simulation.

random: Python module used for random number generation.

pandas: Python library used for data analysis and processing.

* Classes

ManufacturingLine:

init(self, env): Constructor function of the class. The SimPy environment and resource capacities are defined here.

process(self, name, time): Function that calculates how long a particular process takes and saves the results.

* Functions

1-) machine\_process(env, line, name):

It simulates a product going through a machine process. A 10% failure probability is added, in which case 30 minutes of maintenance time is added.

Parameters: env (SimPy environment), line (ManufacturingLine object), name (Product name).

2-) assembly\_process(env, line, name):

It simulates a product going through the assembly process.

Parameters: env, line, name.

3-)inspection\_process(env, line, name):

It simulates a product going through the inspection process.

Parameters: env, line, name.

4-) packaging\_process(env, line, name):

It simulates a product going through the packaging process.

Parameters: env, line, name.

5-)product\_process(env, line, name):

It simulates a product going through the entire production process (machining, assembly, inspection, packaging).

Parameters: env, line, name.

6-)production\_line(env, line):

It simulates the operation of the production line. It ensures that new products enter the production line at regular intervals.

Parameters: env, line.

* Simulation Flow

1-) SimPy environment (env) and ManufacturingLine object (line) are created.

2-) The production line (production\_line) process is started.

3-) The simulation is run for the specified time (8\*60\*60 seconds, i.e. 8 hours).

4-) Simulation results are saved in a pandas DataFrame (df).

5-) The data in the DataFrame is analyzed to calculate the duration of the transactions and the total number of products completed.

* Analysis and Performance Evaluation

1-)DataFrame:

Columns: Process (Process name), Start Time, End Time.

2-)Duration:

The duration of each transaction is calculated (End Time - Start Time).

3-)Average Time Per Process:

The average time per transaction is calculated.

4-)Total Products Completed:

The total number of completed products is calculated.

* Performance Metrics

1-) Average Time Per Process: Average time per process.

2-) Total Products Completed: Total number of products completed.

# CONCLUSION

According to the simulation results, the processes on the production line generally work efficiently and a certain number of products are completed within the specified time.

This report will provide an assessment of the current state of the production line, forming the basis for future improvement efforts. The results obtained provide important information to increase the capacity of the production line and maximize efficiency.