K.H.U. Faculty of Engineering and Natural Sciences

**CMPE412 Computer Simulation**

**Project 2-Manufacturing System**

Project Advisor: Assistant Professor Doğan Çörüş(Instructor)

Project Team: Süleyman Çelik 20181704035

Abdurrahman Arif Demirel 20181706019

Submission Date: 13 June 2024

Table of Contents

[1 Project Overview 3](#_Toc169207899)

[1.1 System Description 3](#_Toc169207900)

[1.2 Objective: 3](#_Toc169207901)

[1.3 Tools and Programming Requirements 3](#_Toc169207902)

[2 Detailed Instructions 4](#_Toc169207903)

[2.1 System Modeling: 4](#_Toc169207904)

[2.2 Data Requirements: 4](#_Toc169207905)

[2.3 Simulation Implementation: 4](#_Toc169207906)

[2.4 Experimentation and Analysis: 4](#_Toc169207907)

# Project Overview

This project involves developing a discrete-event simulation to model a high-volume automotive parts manufacturing line. Initially, the focus is on a single product line to simplify development. A bonus challenge includes extending the simulation to handle multiple product types, each with unique manufacturing requirements.

## System Description

The production line is divided into several phases: raw material handling, machining, assembly, quality control, and packaging. Each stage includes specific machinery and labor requirements, operating across multiple shifts.

## Objective:

* Optimize throughput for a single product line
* Identify and mitigate production bottlenecks
* Analyze impacts of operational variables through scenario analysis
* **Bonus Objective:** Extend the simulation to handle multiple product types, analyzing the additional complexity and resource allocation challenges.

## Tools and Programming Requirements

For the project due to the useful libraries like pandas, simpy, matpotlib, numpy Python has chosen. Also, Python has very easy usage. This usage can be seen at the code. Our team also try the implement project on the C++ but the after the coding challenges team is settled on the Python.

# Detailed Instructions

## System Modeling:

import simpy

import pandas as pd

class ManufacturingLine:

    def \_\_init\_\_(self, env):

        self.env = env

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

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

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

        self.unloading = 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  # 8 hours in seconds

        # Store for raw materials

        self.raw\_materials = simpy.Store(env, capacity=50)  # Assuming a capacity of 50 units

        # Data collection

        self.data = []

At this code we define the processes which are loading, machining, assembly, unloading, inspection, packaging and shift end. Also, capacity of the processes can be seen.

  def process(self, name, time):

        yield self.env.timeout(time)

    def log\_process(self, name, process\_name, start\_time, end\_time):

        self.data.append({

            'Product': name,

            'Process': process\_name,

            'Start': start\_time,

            'End': end\_time,

            'Duration': end\_time - start\_time

        })

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

Product log processes defined.

## Data Requirements:

Process name, start and finish times, durations are defined. As can be inspected simply library give very easy usage of the codes.

def process\_with\_logging(env, line, name, process\_name, duration, resource):

    start\_time = env.now

    with resource.request() as request:

        yield request

        yield env.process(line.process(name, duration))

    end\_time = env.now

    line.log\_process(name, process\_name, start\_time, end\_time)

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

    # Wait for raw materials to be available

    yield line.raw\_materials.get()  # Remove one unit of raw material from the store

    yield env.process(process\_with\_logging(env, line, name, 'Loading', 5\*60, line.loading))

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

    yield env.process(process\_with\_logging(env, line, name, 'Machining', 30\*60, line.machining))

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

    yield env.process(process\_with\_logging(env, line, name, 'Assembly', 20\*60, line.assembly))

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

    yield env.process(process\_with\_logging(env, line, name, 'Unloading', 5\*60, line.unloading))

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

    yield env.process(process\_with\_logging(env, line, name, 'Inspection', 10\*60, line.inspection))

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

    yield env.process(process\_with\_logging(env, line, name, 'Packaging', 15\*60, line.packaging))

Processes and their durations are defined.

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

    """ Run the complete production process for a single product. """

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

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

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

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

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

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

## Simulation Implementation:

Run the complete production process for a single product.

def production\_line(env, line):

    """ Generate products and process them continuously. """

    i = 0

    while env.now < line.shift\_end:

        i += 1

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

        # Wait a small time before starting the next product to avoid overlap

        yield env.timeout(1)  # 1 second delay between starting new products

def raw\_material\_arrival(env, line):

    """ Generate raw materials at a regular interval. """

    while True:

        yield env.timeout(30\*60)  # Raw materials arrive every 30 minutes

        for \_ in range(10):  # Assume each arrival brings 10 units of raw materials

            yield line.raw\_materials.put(1)

        print(f'Raw materials arrived at {env.now}')

env = simpy.Environment()

line = ManufacturingLine(env)

# Start raw material arrivals and production processes

env.process(raw\_material\_arrival(env, line))

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

# Run the simulation for 24 hours

env.run(until=24\*60\*60)

# Convert collected data to a DataFrame for analysis

df = pd.DataFrame(line.data)

# Save the data to a CSV file (optional)

df.to\_csv('manufacturing\_line\_data\_with\_raw\_materials.csv', index=False)

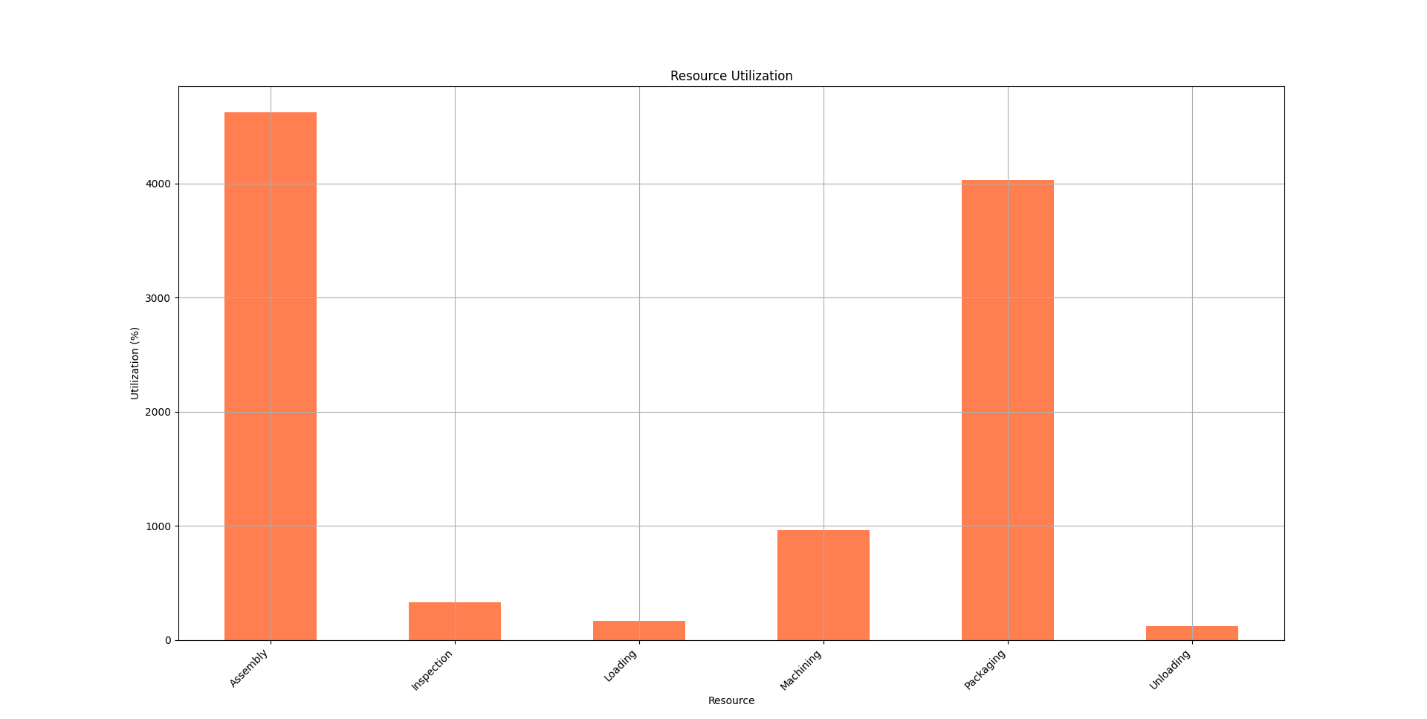
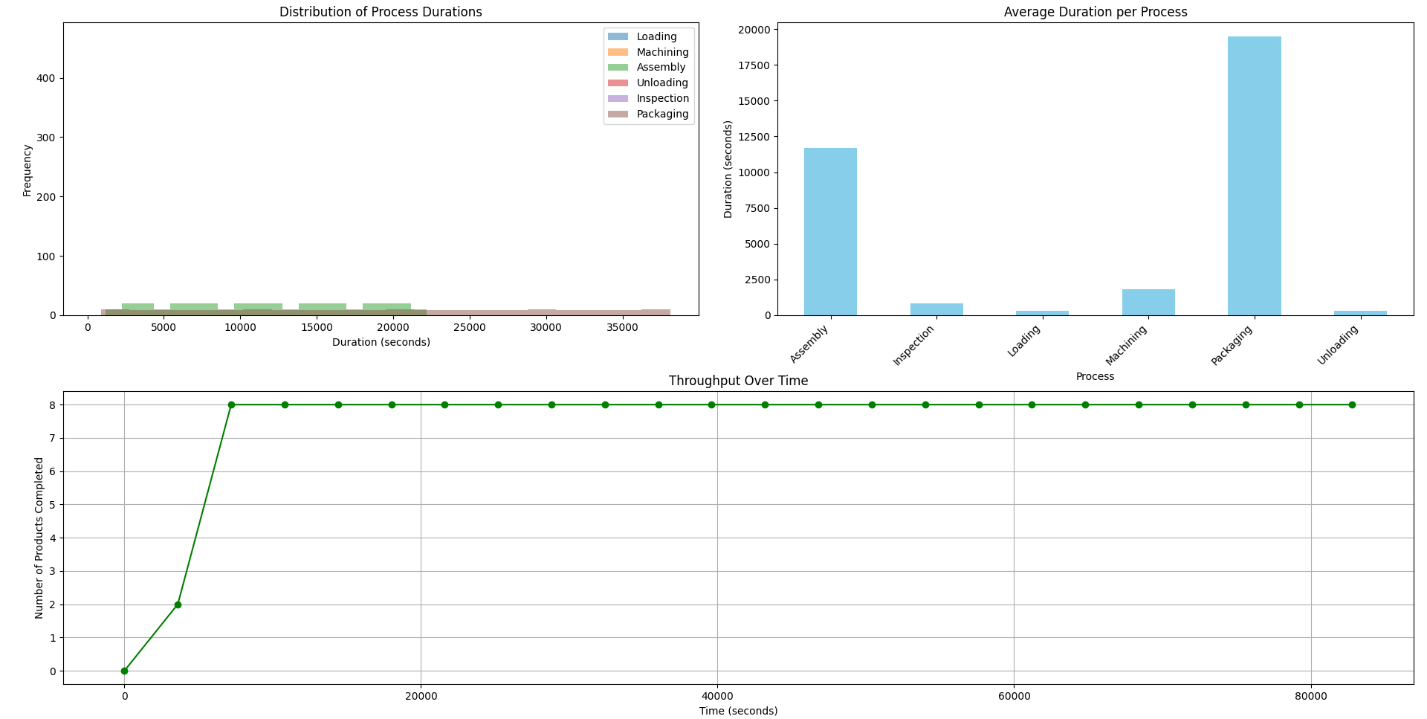
# Display the DataFrame

print(df.head())

Last portion of the code arranges the simulation time, raw material arrivals and write the data to csv file.

## Experimentation and Analysis:

As we can see at the plots Assembly and the Packaging process’ are taking much more time than other processes.



Taking much more time at the assembling and packaging cause bottleneck at the packaging.