Lab Units 3 & 4 Creating a Digital Twin Shop-Floor

- Developing a Digital Twin of a Shop-Floor Using RobotStudio
- Cyber-Physical Fusion Techniques for Digital Twin Applications
- Real-World Data Integration into Digital Twins
- Validation of Digital Twin Models with Real-World Data

Overview:

In this unit, you will create a digital twin of a factory shop-floor using RobotStudio. The objective is to enable a virtual representation of the factory's operations, which can be used for monitoring, analyzing, and optimizing production. You will also explore the integration of real-world data into the digital environment to reflect actual factory dynamics.

Learning Objectives:

1. Develop a digital twin of a simple factory shop-floor using RobotStudio.
2. Understand and apply cyber-physical fusion techniques for digital twin shop-floors.
3. Integrate real-world data from the factory shop-floor, including machine performance metrics, production outputs, etc., into the digital twin.
4. Validate the effectiveness of the digital twin by comparing its predictive insights against real-world factory data.

Work Breakdown:

1. Class Work (2 Hours) - Introduction and Setup:

- **Introduction to Digital Twins:**
  - Overview of RobotStudio, including basic features and initial setup.
- **Initial Model Setup:**
  - Guided hands-on session to create a simple factory layout in RobotStudio. Replicate machines and material flow on the shop-floor.

2. Individual Work (10 Hours) - Digital Twin Development and Validation:

Task 1: Develop the Digital Twin (3 Hours)

- Set up your digital twin using RobotStudio.
- Recreate a simple shop-floor layout with key components such as robots, conveyors, and assembly stations. For beginners, consider one robot and one part positioner that can be imported from the RobotStudio library.

Integrating Simulated Data in RobotStudio Using a CSV File

Step 1: Prepare the CSV File

- Create a CSV file containing the relevant simulated data in tabular format.
- Example format:

Timestamp	Machine ID	Cycle Time (s)	Production Count	Status	Energy Consumption (kWh)
2024-10-03 08:00	Machine_1	15	100	Operational	1.5
2024-10-03 08:10	Machine_1	16	150	Maintenance	1.3

  - **Include:**
    - **Timestamp:** Time when data is recorded.
    - **Machine ID:** Identifies different machines.
    - **Cycle Time:** Simulated operational cycle time.
    - **Production Count:** Number of units produced.
    - **Status:** Machine status (e.g., Operational, Idle, Maintenance).
    - **Energy Consumption:** Power usage during operations.

See the illustrative csv file.

Step 2: Import CSV File into RobotStudio

1. **Save CSV File in an Accessible Location:**
   - Ensure the CSV file is saved in a location accessible by RobotStudio (e.g., on your local drive).
2. **Use a Client-Server Setup for Data Integration:**
   - In RobotStudio, use a client-server approach where the client is implemented in Python to read the CSV data, and the server is implemented in RAPID to receive and apply the data.
   - Write a Python script to read the CSV file and establish a connection to RobotStudio.
   - Implement a RAPID server program in RobotStudio to handle the incoming data and update signals accordingly.
3. **Define Signal Mapping in RobotStudio:**
   - **Create Signals in RobotStudio:** Define signals to represent each data point, such as cycle time or production count.

- **Map CSV Columns to Signals:** Using the client-server script, ensure that each column in the CSV is mapped to a corresponding signal in RobotStudio (e.g., map **Cycle Time** to CycleTimeSignal).

Step 3: Use Imported Data in Simulation

- **Simulate Real-World Behavior:**
  - Use RAPID routines that utilize the imported signals to adjust robot behavior in real-time.
  - For example, create a routine that changes the robot's speed based on the **Cycle Time** data received from the client.
- **Visualization (Optional):**
  - **Charts and Graphs:** Use RobotStudio's built-in visualization tools to display production counts, energy consumption, and machine status over time.
  - **Monitor Performance:** Monitor how the system performs and adjust the model to match expected outcomes using the imported data.

Step 4: Validate the Model

- Use the imported data to validate the digital twin's performance.
- Compare the simulation output (e.g., number of units produced, time taken) with the data in your CSV file to ensure that the twin's behavior aligns with expectations.

Tips for Better Integration:

- **Consistent Data Updates:** If changes are made to the CSV file, ensure the client re-reads and sends the updated data to RobotStudio for real-time simulation effects.
- **Scripting Enhancements:** Use Python scripts for data preprocessing and RAPID programs to control how and when data affects the simulation model.
- **Testing Scenarios:** Use different CSV datasets to simulate various production scenarios, such as increased demand or machine breakdowns, to explore optimization strategies.

Task 3: Data Integration and Validation (4 Hours)

- **Data Integration:**
  - Integrate machine parameters, operational times, and production counts into your digital twin using the client-server application.
- **Validation:**
  - Validate the digital twin by comparing its simulated outputs with the collected real-world data to assess its predictive capabilities.

- Identify discrepancies between the simulation and real data and analyze potential causes.

Factory Data: Machine Performance Metrics and Production Outputs

1. Machine Performance Metrics:

- **Cycle Time:** Time taken by each machine to complete one operational cycle (e.g., 30 seconds per item).
- **Downtime Duration:** Duration machines are non-operational due to maintenance or faults (e.g., 15 minutes per shift).
- **Utilization Rate:** Percentage of time machines are actively producing versus idle (e.g., 85%).

2. Production Outputs:

- **Daily Production Rate:** Number of units produced per day by the factory (e.g., 500 units/day).
- **Defect Rate:** Percentage of items produced that are defective or require rework (e.g., 3%).
- **Production Target:** Expected production output over a specific period (e.g., 2,500 units per week).

3. Operational Data:

- **Power Consumption:** Energy used by each machine during operation (e.g., 1.5 kWh per cycle).
- **Machine Status:** Real-time status logs showing whether a machine is operational, idle, or under maintenance.

++++++++++++

Below is an example of a Python client script and a RAPID server program that can be used for integrating CSV data into RobotStudio using a client-server approach. This implementation aims to read data from a CSV file using Python and transmit it to RobotStudio, where the data will be processed using RAPID.

Python Client Script

This script reads data from a CSV file and sends it to RobotStudio using a TCP/IP connection.

```
import csv
import socket
import time
# Configuration parameters for server connection
SERVER_IP = '127.0.0.1'  # IP of the machine running RobotStudio
SERVER_PORT = 1025  # Port number on which the RAPID server is listening

# CSV file path
CSV_FILE_PATH = 'factory_data.csv'
```

```
# Function to send data to the server
def send_data_to_server(data):
    try:
        # Create a socket connection
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((SERVER_IP, SERVER_PORT))
            # Send data to the server
            s.sendall(data.encode('utf-8'))
            # Wait for server acknowledgment
            response = s.recv(1024)
            print(f"Server response: {response.decode('utf-8')}")
    except Exception as e:
        print(f"Error: {e}")

# Function to read CSV file and send each row to the server
def read_and_send_csv():
    with open(CSV_FILE_PATH, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            # Convert row to a formatted string to send to the server
            data = f"{row['Timestamp']},{row['Machine ID']},{row['Cycle Time (s)']},{row['Production
Count']},{row['Status']},{row['Energy Consumption (kWh)']}"
            send_data_to_server(data)
            time.sleep(1)  # Send data with a delay to simulate real-time behavior

if __name__ == '__main__':
    read_and_send_csv()
```

RAPID Server Program

This RAPID server script runs on the robot controller, listens for incoming data from the Python client, and updates the corresponding signals in RobotStudio.

```
MODULE ServerModule
    VAR socketdev clientSocket;  ! Socket variable for communication
    PERS string socketName:="clientSocket"; ! Socket device name
    CONST int portNum:=1025;      ! Port number to listen on

    ! Signals for data mapping
    VAR num CycleTimeSignal;
    VAR num ProductionCountSignal;
    VAR string MachineStatusSignal;
    VAR num EnergyConsumptionSignal;

    ! Procedure to handle incoming data
    PROC HandleData(string inputData)
        VAR string timestamp;
        VAR string machineID;
        VAR num cycleTime;
        VAR num productionCount;
        VAR string status;
        VAR num energyConsumption;

        ! Parse incoming data (comma-separated)
        ReadStr(inputData, timestamp, machineID, cycleTime, productionCount, status,
energyConsumption);

        ! Update corresponding signals
        CycleTimeSignal := cycleTime;
        ProductionCountSignal := productionCount;
        MachineStatusSignal := status;
        EnergyConsumptionSignal := energyConsumption;

        ! Print received data for debugging
        TPWrite "Received Data:";
        TPWrite "Timestamp: " + timestamp;
        TPWrite "Machine ID: " + machineID;
        TPWrite "Cycle Time: " + NumToStr(cycleTime, 0);
        TPWrite "Production Count: " + NumToStr(productionCount, 0);
        TPWrite "Status: " + status;
        TPWrite "Energy Consumption: " + NumToStr(energyConsumption, 0);
    ENDPROC
```

```
    ! Main loop to listen for incoming client connections
    PROC Main()
        ! Setup socket server
        OpenSocket clientSocket, socketName, portNum;

        WHILE TRUE DO
            VAR string receivedData;
            VAR string clientAddress;

            ! Wait for incoming data
            SocketAccept clientSocket\Wait:=10, clientAddress;

            ! Read data from client
            SocketReceive clientSocket, receivedData;

            ! Handle the received data
            HandleData(receivedData);

            ! Send acknowledgment to the client
            SocketSend clientSocket, "Data Received Successfully";

            ! Close connection after receiving
            SocketClose clientSocket;
        ENDWHILE

        ! Close socket server when done
        CloseSocket clientSocket;
    ENDPROC
ENDMODULE
```

Explanation:

1. **Python Client Script:**
    - **Socket Connection:** The Python client connects to the server via a TCP socket to send data.
    - **CSV File Read:** The client reads each row from the CSV file and sends it to the server.
    - **Data Encoding:** Data is formatted as a comma-separated string for easy parsing in RAPID.
2. **RAPID Server Program:**
    - **Socket Setup:** A socket is opened on a specific port to listen for incoming data.
    - **Data Handling:** Data received from the client is parsed and mapped to specific signals in RobotStudio (e.g., CycleTimeSignal, ProductionCountSignal).
    - **Acknowledgment:** The server sends a response to the client indicating successful reception.

Notes:

- **IP Address and Port Number:** Update the SERVER_IP in the Python script to the IP address of the machine where RobotStudio is running. Make sure the SERVER_PORT matches the port number used in the RAPID server (portNum).
- **Signal Mapping:** The RAPID server maps incoming data to specific signals representing different machine metrics. You can adjust the signal variables (CycleTimeSignal, etc.) as needed.
- **Real-Time Data Integration:** A time.sleep(1) delay in the Python client simulates real-time data transfer, which you can adjust as needed.
- **Debugging:** The TPWrite statements in the RAPID program help to debug by showing the received data on the teach pendant.