## Lab Units 1 & 2 Introduction to Digital Twins

- Software Installation and Configuration
- Exploration of Digital Twin Environments
- Introduction to RobotStudio and Basic Programming
- Setting Up Digital Twin Communication with MQTT and CSV Integration
- Hands-On Test Environment and Validation

**Install and Configure the Following Software and Components:**

**1. Modeling Tools and Simulation Platforms:**

- **Online Exploration**: In Unit 1, students should explore online videos and documentation to learn about various digital twin environments and communication protocols. Some recommended sources include:
    - Digital Twin Environments: Siemens MindSphere, GE Predix, and Azure Digital Twins.
    - Communication Protocols: OPC UA, MQTT, and Modbus.

**2. Install Systems for Future Use:**

- **RobotStudio**: Required for later labs. https://new.abb.com/products/robotics/downloads Server License: 20.123.29.129
- **Xcos (part of Scilab)**: Useful for modeling real-world systems.

  https://www.scilab.org/software/xcos
- **Python**: You'll use it throughout the course. https://www.python.org/downloads/
- **Install necessary libraries** like NumPy, SciPy, SymPy, and matplotlib for modeling, visualization, and implementing analytical models and data analysis.

  ```
  pip install numpy
  pip install scipy
  pip install sympy
  pip install matplotlib
  ```

**3. Cloud Storage Solutions:**

- **Configure access to cloud services**, such as Google Cloud Platform or Amazon Web Services.
- **Install related Python libraries** such as boto3 for AWS or azure-storage-blob for Azure.

  ```
  pip install boto3
  ```

  boto3 is the Amazon Web Services (AWS) SDK for Python that allows you to interact with various AWS services, such as S3, EC2, and more.

  ```
  pip install azure-storage-blob
  ```

azure-storage-blob is the Azure SDK for Python that allows you to work with Azure Blob Storage, providing a convenient way to upload, download, and manage blobs.

**4. Communication Protocols:**

- **Install MQTT broker software** (such as Mosquitto) and related Python packages (like paho-mqtt).

  **On Windows**

  - Download Mosquitto from the official website: https://mosquitto.org/download/
  - Run the installer and follow the instructions.

  ```
  pip install paho-mqtt
  ```

  paho-mqtt is a popular Python library used for interfacing with MQTT brokers.

- **Familiarize students with nodeRED** for cloud connectivity.

  https://flowfuse.com/blog/2024/06/how-to-use-mqtt-in-node-red/

  https://nodered.org/

**5. Hands-on Test Environment:**

- **Set up a basic test case involving a tool like RobotStudio** to demonstrate the functioning of the digital twin environment and ensure all components are communicating correctly.

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

**Step 1: Install and Set Up RobotStudio**

**RobotStudio** is a software platform from ABB for creating, simulating, and testing robotic applications.

1. **Download RobotStudio**:
   - Visit the ABB RobotStudio download page and download the latest version.
   - You may need to create an ABB account to download.
2. **Install RobotStudio**:
   - Run the installer and follow the instructions.
   - Once installed, open RobotStudio. You might need a license, but there is usually a 30-day free trial available.

**Step 2: Create a New Station in RobotStudio**

1. **Open RobotStudio** and create a new station:
   - Click on **File > New** and select **New Station**.
   - You can select a template or start with a blank station. For this case, starting with a **blank station** is sufficient.
2. **Add a Robot**:

- From the **Home** tab, click on **Add Robot**.
- Choose a robot model from the library that fits your use case (e.g., ABB IRB 120).
- Place the robot in the station.

3. **Configure a Simple Task**:
   - Right-click on the robot and select **Create a New Program**.
   - Use **Rapid Programming Language** to create a simple program that moves the robot between a few points.
   - Define at least two points (e.g., **Point A** and **Point B**) and add a **MoveL** (linear movement) command between them.

**Step 3: Set Up Digital Twin Communication**

1. **Create an IO Simulation**:
   - To simulate a digital twin, the robot's interaction with a factory floor (sensors, actuators) should be represented.
   - Add **Digital IO** from the **Controller tab**.
   - Define some input (e.g., a sensor status) and output signals (e.g., an actuator control signal).
2. **Add Simulated Equipment**:
   - Use the **Modeling tab** to add conveyor belts, stations, or other components.
   - Adjust their properties to simulate behaviors, such as start/stop or speed.
3. **Add Data Sources (CSV Integration)**:
   - Prepare a **CSV file** containing simulated machine performance metrics, production outputs, or equipment status (e.g., data.csv).
   - Include columns like Time, MachineStatus, ProductionCount, etc.
4. **Use Python to Integrate CSV Data**:
   - You can write a **Python script** to read the CSV file and use the data to control simulated components in RobotStudio.
   - An example Python script might read from the CSV file periodically and set the **Digital IO** signals accordingly.

**Step 4: Connect to an MQTT Broker (Optional, for Real-Time Integration)**

1. **Set Up MQTT**:
   - Use **Mosquitto** (as previously installed) to act as the MQTT broker.
   - Configure Mosquitto to listen on a specified port (e.g., 1883).
2. **Set Up MQTT in RobotStudio**:

- Write a Python script using the **paho-mqtt** library to publish and subscribe to topics representing sensor data.
- For instance, publish sensor data changes from RobotStudio to an MQTT topic (e.g., factory/sensor1), and subscribe to another topic for incoming control signals.

Example Python snippet:

```
import paho.mqtt.client as mqtt
import csv
import time

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("factory/control")

def on_message(client, userdata, msg):
    # Handle incoming messages, such as changing robot state in RobotStudio
    print(f"{msg.topic} {msg.payload}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("localhost", 1883, 60)

# Publish data from CSV periodically
with open('data.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        client.publish("factory/sensor1", row['MachineStatus'])
        time.sleep(1)  # Simulate real-time data publishing
client.loop_forever()
```

**Step 5: Test the Digital Twin Environment**

1. **Run the Simulation in RobotStudio**:
   - Click on **Simulation** and start the robot program.
   - Ensure the robot moves between the defined points.
2. **Monitor Data Communication**:
   - Verify that the CSV data influences robot actions or IO status as expected.
   - Monitor the communication between the Python script, MQTT broker, and RobotStudio. The robot should respond in real-time to MQTT control messages.
3. **Visual Feedback**:
   - Use RobotStudio's **3D Visualization** to observe how robot actions change based on external data.
   - You can also add **Panels and Dashboards** in RobotStudio to display data visually, such as current production count or equipment status.

**Step 6: Validate the Test Case**

1. **Check Synchronization**:
    - Ensure the robot's physical actions match the commands given by the CSV or MQTT broker.
    - All the components (robot, data from CSV, control signals) should function in sync.
2. **Introduce Failure Scenarios**:
    - Edit the CSV to simulate failures (e.g., machine stop, error signals) and ensure the digital twin correctly reflects these conditions.
    - Monitor the robot's response—e.g., stopping its motion or raising an alarm.
3. **Log and Analyze Data**:
    - Use logging in Python to record communication with the MQTT broker.
    - Analyze the logs to verify that the data exchange occurs as expected without delays.

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

To use Python to integrate CSV data into **RobotStudio**, you will need to use an approach where data is read from a CSV file and then used to simulate or influence the behavior of components within the RobotStudio environment. Here's a step-by-step guide to achieve this:

**Step 1: Prepare Your Environment**

1. **Install Required Python Libraries**:
    - You will need the pandas library to handle CSV data and pywin32 (for Windows COM interface) if you want to use external interaction between Python and RobotStudio.

```
pip install pandas pywin32
```

2. **Prepare the CSV File**:
    - Create a CSV file (data.csv) with columns representing the data you want to integrate. For instance:

```
Time, MachineStatus, ProductionCount
0, On, 100
1, Off, 105
2, On, 110
```

**Step 2: Configure RobotStudio for External Control**

1. **Add Digital IO in RobotStudio**:
    - In RobotStudio, create a simple digital IO setup.
    - Go to the **Controller** tab and add **Digital IO** signals to represent sensors or actuators.
    - You may name them MachineStatusSignal, ProductionCountSignal, etc.
2. **Set Up the Controller Communication**:
    - If you want to control the digital IO from an external Python script, you'll need to use a communication channel through a TCP/IP protocol or OPC server that RobotStudio can understand.

**Step 3: Writing the Python Script**

Here, we will write a Python script that reads the CSV file and then updates the digital IO signals accordingly in RobotStudio.

**Example Python Script to Read CSV Data**

This script will read from the CSV file and simulate the control over the digital IO using RobotStudio's COM interface.

1. **Import Required Libraries**:

```
import pandas as pd
import time
import win32com.client  # pywin32 for COM interaction with RobotStudio
```

2. **Load the CSV Data**:

```
# Load CSV data using pandas
csv_file = 'data.csv'
data = pd.read_csv(csv_file)
```

3. **Establish Communication with RobotStudio**:
    - RobotStudio exposes an **OLE Automation API** which you can access via Python.
    - We will create a COM client to connect to RobotStudio.

```
# Establish COM interface with RobotStudio
try:
    robot_studio = win32com.client.Dispatch("RobotStudio.Robot")
    print("Connected to RobotStudio successfully.")
except Exception as e:
```

```
    print(f"Failed to connect to RobotStudio: {e}")
    exit()
```

4. **Read Data and Control IO**:
    - Iterate over the rows in the CSV file, and set digital IO signals accordingly in RobotStudio.

```
for index, row in data.iterrows():
    # Assuming digital IO points correspond to MachineStatus and ProductionCount
    machine_status = row['MachineStatus']
    production_count = row['ProductionCount']
    # Simulate setting digital IO in RobotStudio
    # Assuming you have set IO named "MachineStatusSignal" and "ProductionCountSignal"
    try:
        # Set IO for MachineStatusSignal
        machine_status_signal = 1 if machine_status == "On" else 0
        robot_studio.SetIO("MachineStatusSignal", machine_status_signal)
        print(f"Machine Status set to: {'On' if machine_status_signal else 'Off'}")
        # Set IO for ProductionCountSignal (just a demonstration)
        robot_studio.SetIO("ProductionCountSignal", production_count)
        print(f"Production Count set to: {production_count}")
        # Wait for 1 second before moving to the next row to simulate real-time data
        time.sleep(1)
    except Exception as e:
        print(f"Failed to set IO: {e}")
print("Completed setting all IOs based on CSV data.")
```

**Step 4: Test the Integration**

1. **Run RobotStudio** and make sure the **controller** is started.
2. **Load the Python script** and let it interact with RobotStudio in real-time.
3. **Monitor the digital IO signals**:
    - You can use the **I/O Simulator** in RobotStudio to see if the signals change based on the CSV data.

**Explanation**

- The Python script:
    - Reads the CSV file line by line.
    - Converts the data to a format that can control the digital IO (MachineStatusSignal and ProductionCountSignal) in RobotStudio.
    - Uses **COM (Component Object Model)** to communicate with RobotStudio and set the IO values.
- This approach simulates the integration of real-time sensor data into a digital twin environment.

**Alternative Approach Using OPC or MQTT**

If COM interaction doesn't fit well for your needs, you can also consider using **OPC servers** or **MQTT** to establish communication between Python and RobotStudio, especially for more scalable industrial use cases.

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

**Step-by-Step Guide for Integrating CSV Data with RobotStudio Using Python**

**Step 1: Set Up RobotStudio**

1. **Create a New Project**:
    - Open **RobotStudio**, create a new station, and add a **robot model**.
    - Add **Digital IO signals** that will represent the states you want to control (e.g., signals for controlling machine status).
2. **Set Up Digital IO in RobotStudio**:
    - Go to the **Controller** tab and add digital input and output signals to represent the actions or statuses.
    - Name these signals (e.g., MachineStatus, ProductionCount).
3. **Prepare the CSV File**:
    - Create a CSV file named data.csv containing the relevant data for controlling the simulation.

Time, MachineStatus, ProductionCount

0, On, 100

1, Off, 150

2, On, 200

    - This CSV file will be used as a data source for controlling the robot's actions.

**Step 2: Write the Python Script to Read CSV Data**

The Python script will read the CSV data and simulate an action by changing IO signals in RobotStudio. Since RobotStudio does not directly execute Python scripts, we need to rely on communication mechanisms to interact with the digital twin.

**Option A: TCP/IP Communication Using RAPID**

1. **Set Up TCP Server in RobotStudio (RAPID Code)**:
    - Write a **RAPID** program in RobotStudio that will open a socket and listen for incoming data from Python.
    - Example RAPID code:

```
VAR socketdev serverSocket;
VAR string inData;
VAR num result;
PROC startServer()
    Connect serverSocket \Server, "127.0.0.1", 1025;
    WHILE TRUE DO
        result := SocketReceive(serverSocket, inData);
        IF result = 0 THEN
            ! Parse the incoming data
            SetDO MachineStatusSignal, StrFind(inData, "On") > 0;
        ENDIF
        WaitTime 0.5;
    ENDWHILE
    Disconnect serverSocket;
ENDPROC
```

2. **Write a Python Script to Send Data**:
    - Use Python's socket library to send CSV data to RobotStudio.
    - Example Python code:

```
import socket
import pandas as pd
import time
# Load CSV data using pandas
data = pd.read_csv('data.csv')
# Create a TCP client socket
```

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

client_socket.connect(("127.0.0.1", 1025))

# Send each row of data periodically to RobotStudio

for _, row in data.iterrows():

    message = f"{row['MachineStatus']},{row['ProductionCount']}\n"

    client_socket.sendall(message.encode('utf-8'))

    time.sleep(1)  # Wait 1 second between each data point

client_socket.close()
```

**Option B: OPC-UA for Communication**

1. **Set Up OPC-UA Server in RobotStudio**:
    - RobotStudio can work with **OPC-UA** as a protocol for communication with external systems.
2. **Install Python OPC-UA Library**:
    - Install the opcua library for Python.

```
pip install opcua
```

3. **Write Python Code to Control the Robot**:

```
from opcua import Client

import pandas as pd

import time

# Load CSV data using pandas

csv_file = 'data.csv'

data = pd.read_csv(csv_file)

# Set up an OPC-UA client

client = Client("opc.tcp://127.0.0.1:4840")  # Adjust URL as per your
configuration

client.connect()

# Iterate through the CSV data

for _, row in data.iterrows():

    machine_status_node = client.get_node("ns=2;i=2")  # Example node ID for
MachineStatus

    production_count_node = client.get_node("ns=2;i=3")  # Example node ID for
ProductionCount

    # Update the values on the OPC-UA server
```

```
    machine_status_node.set_value(True if row['MachineStatus'] == 'On' else
False)
    production_count_node.set_value(row['ProductionCount'])
    time.sleep(1)  # Delay to simulate time intervals
# Disconnect from OPC-UA server
client.disconnect()
```

**Step 3: Verify and Test**

1. **Run RobotStudio**:
    - Start your station, load the RAPID code, and execute the startServer() procedure.
    - Ensure the digital IO setup is properly configured.
2. **Run the Python Script**:
    - Run the Python script to send the CSV data to RobotStudio.
    - Observe that the robot actions and IO signals are updated based on the incoming data.
3. **Verify Communication**:
    - Check if the **Digital IO** signals in RobotStudio reflect the status changes coming from the CSV data.