MUĞLA SITKI KOÇMAN UNIVERSITY
2025-2026 FALL SEMESTER PROJECT

TRACTION CONTROL SYSTEM SIMULATION

Student Number / Student Name

230702032 / Ali ÇİFT

Name of Lecturers

Fatma TAŞCIKARAOĞLU

Date:24/12/2025

EEE 3506 Programmable Logic Controllers

2025

CONTENTS

1. PROJECT SCENARIO

This project is a **Digital** application where we bridge the gap between industrial control and physical simulation:

- **The Driver (HMI):** Inputs commands like Gas, Brake, or Road Surface (Dry/Icy).
- **The PLC (The Brain):** Acts as the Electronic Control Unit (ECU). It processes driver requests and monitors speed data from Python. If it detects wheel spin, it automatically reduces motor power (**Traction Control**).
- **Python (The Physical World):** Acts as the vehicle and environment. It receives "Torque" from the PLC and calculates "Speed" based on gravity, air drag, and road friction.
- **The Outcome:** Just like a real car, hitting the gas on ice causes wheels to spin without moving the car, until the TCS activates to restore grip.

---

2. ST CODE (PLC LOGIC)

This logic runs on the PLCnext controller in a real-time loop:

1. **System_Active Logic:** A **Latching** circuit. The system stays active after a Start pulse until a Stop pulse is received.
2. **temp_MaxTorque:** A **Power Gate**. If the system is inactive, max torque is 0; otherwise, it is 100.
3. **Kp := 10.0;**: The **Proportional Gain**. It determines how aggressively the system reacts to wheel slip.
4. **temp_TargetTorque**: The **Raw Demand**. This is the initial torque requested by the driver via the gas pedal.
5. **temp_SlipError**: The **Error Measurement**. It calculates the difference between Wheel Speed and Vehicle Speed.
6. **B := (temp_SlipError > 5.0);**: The **Threshold**. A slip higher than 5 km/h triggers a warning signal.
7. **A := HMI_TCS_Active AND B;**: The **Activation Logic**. TCS engages only if the driver has enabled it and slip is detected.
8. **temp_Reduction**: The **Intervention**. If slip is detected, the error is multiplied by Kp to calculate how much torque to "cut."
9. **CMD_MotorTorque**: The **Final Decision**. The requested torque minus the TCS reduction.
10. **CMD_MotorTorque1**: The **Safety Clamp**. Ensures motor torque never drops below zero (prevents reverse rotation).

```
System_Active := (HMI_Start OR System_Active) AND NOT HMI_Stop;

temp_MaxTorque := SEL(System_Active, (0.0), (100.0));

    Kp := 10.0;




temp_TargetTorque := (HMI_GasPedal / 100.0) * temp_MaxTorque;


temp_SlipError := Py_WheelSpeed - Py_VehicleSpeed;
Monitor_Slip := temp_SlipError;


B:= (temp_SlipError > 5.0);

A := HMI_TCS_Active AND B;

Monitor_Status := A;

temp_Reduction := SEL(A, (0.0), (temp_SlipError * Kp));

CMD_MotorTorque := temp_TargetTorque - temp_Reduction;

CMD_MotorTorque1 := MAX(0.0, CMD_MotorTorque);
```

3. OPC UA: The Industrial Bridge

The Scenario: OPC UA acts as the "Universal Translator" between the industrial world (PLC) and the software world (Python). It enables a seamless and secure exchange of data, effectively bridging the gap between OT (Operational Technology) and IT (Information Technology).

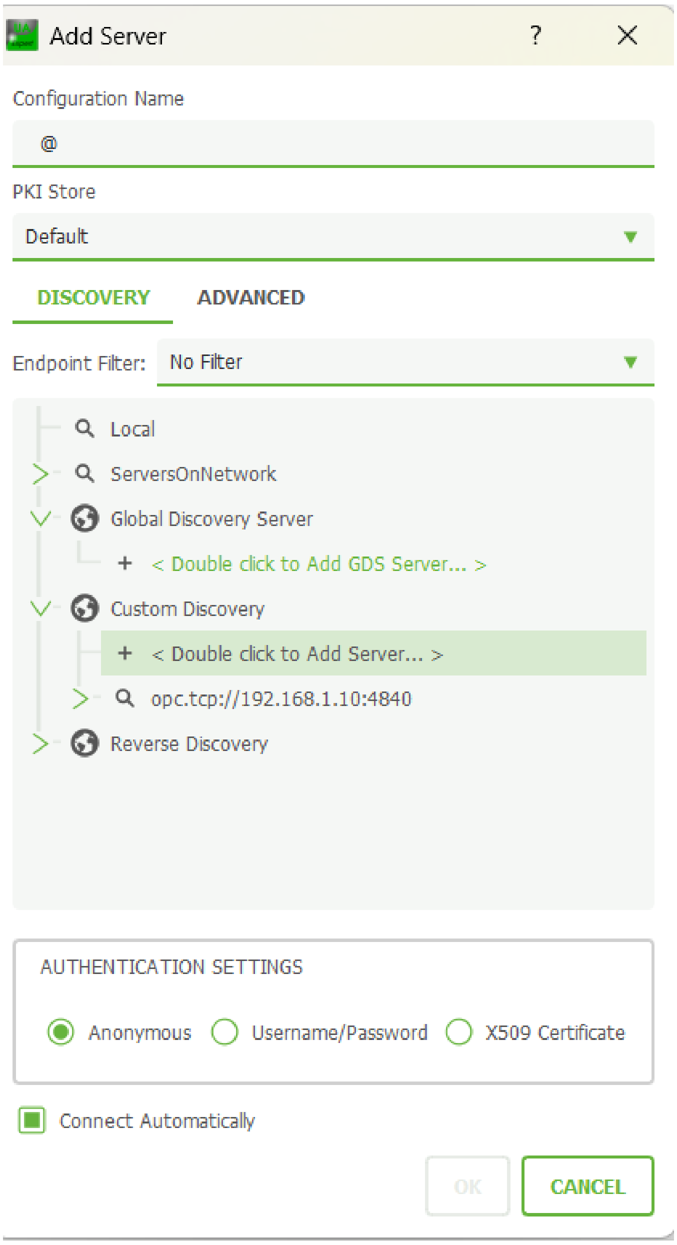
Key Technical Parts:

1. **PLC as the Server:** The PLCnext acts as the OPC UA Server, hosting all variables (Nodes) like Torque and Speed in its memory.
2. **Security & Authentication:** To ensure industrial cybersecurity, we utilized X.509 Certificates. Using OpenSSL, we generated a certificate that allows the PLC to verify and trust the Python Client.
3. **Encrypted Communication:** We implemented the **"Sign & Encrypt"** policy with Basic256Sha256. This ensures that the data traveling between Python and the PLC cannot be intercepted or manipulated**.**
4. **Node Addressing:** Every variable is accessed through a specific NodeID. This allows Python to pinpoint exactly which memory address to read from or write to.
5. **Real-Time Synchronization:** Python operates as a high-speed Client, reading torque commands and writing back speed telemetry every 0.1s, maintaining a tight feedback loop for the simulation.

opc.tcp://127.0.0.1:4840

VALUE	DATATYPE
0	Float
0	Float
false	Boolean
false	Boolean
0	Float
0	Float
0	Float

NODE ID
NS6|String|Arp.Plc.Eclr/Py_VehicleSpeed
NS6|String|Arp.Plc.Eclr/Py_WheelSpeed
NS6|String|Arp.Plc.Eclr/MainInstance.A
NS6|String|Arp.Plc.Eclr/MainInstance.B
NS6|String|Arp.Plc.Eclr/HMI_GasPedal



4. PYTHON CODE

The Python script connects to the PLC via OPC UA to simulate the "Real World":

PART 1: Security & Connection

- **client.set_security_string(...)**: Implements **SignAndEncrypt** using X.509 certificates. This ensures a secure, industrial-grade handshake with the PLC.

PART 2: Node Acquisition

- **client.get_node(...)**: Locates variables within the PLC memory using **NodeIDs**. These act as the digital entry points for reading and writing data.

PART 3: Data Acquisition (Reading)

- **gas_val = node_read_torque.get_value()**: Continuously reads the torque command calculated by the PLC.
- It also tracks environmental data like slippery_val and brake_val.

PART 4: Calculation

- **target_speed = gas_val * 2.2**: Defines the maximum reachable speed based on pedal position.
- **Acceleration & Drag**: Implements inertia and air resistance so the car doesn't speed up or slow down instantly.

PART 5: Friction & Slip Simulation (The Core)

- **grip_percent = 1.0 - (slippery_val / 110.0)**: Calculates "Grip" based on road conditions.
- **ideal_vehicle_speed = wheel_speed * grip_percent**: This is where **Slip** happens. On ice, the wheel speed can be high while the vehicle speed remains low.

PART 6: Feedback & Logging

- **node_write_veh_speed.set_value(...)**: Sends calculated speeds back to the PLC to complete the feedback loop.
- **print(...)**: Logs real-time data to the console for monitoring the simulation's health.

```
PLC_URL = "opc.tcp://127.0.0.1:4840"
PLC_USER = "admin"
PLC_PASS = "plcnext"
```

```
node_read_torque = client.get_node("ns=6;s=Arp.Plc.Eclr/MainInstance.CMD_MotorTorque")
```

```
node_write_veh_speed = client.get_node("ns=6;s=Arp.Plc.Eclr/Py_VehicleSpeed")
node_write_wheel_speed = client.get_node("ns=6;s=Arp.Plc.Eclr/Py_WheelSpeed")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

ACCELERATING | Gas:%70 | Brake:%10 | DRY | Wheel Speed:100 | Vehicle Speed:8
ACCELERATING | Gas:%70 | Brake:%15 | DRY | Wheel Speed:101 | Vehicle Speed:8
ACCELERATING | Gas:%70 | Brake:%20 | DRY | Wheel Speed:102 | Vehicle Speed:8
SLOWING DOWN | Gas:%70 | Brake:%25 | DRY | Wheel Speed:101 | Vehicle Speed:89
SLOWING DOWN | Gas:%70 | Brake:%30 | DRY | Wheel Speed:100 | Vehicle Speed:90
SLOWING DOWN | Gas:%70 | Brake:%35 | DRY | Wheel Speed:98 | Vehicle Speed:91
SLOWING DOWN | Gas:%70 | Brake:%40 | DRY | Wheel Speed:96 | Vehicle Speed:91
```

PYTHON CODE IN TEXT

```
import sys
import time
import os

try:
    from opcua import Client, ua
    import cryptography
except ImportError:
    print("  EKSİK: 'pip install opcua cryptography' yap.")
    sys.exit()

# --- AYARLAR ---
PLC_URL = "opc.tcp://127.0.0.1:4840"
PLC_USER = "admin"
PLC_PASS = "plcnext"
CERT_FILE = "my_cert.der"
KEY_FILE = "my_private_key.pem"

def main():
    if not os.path.exists(CERT_FILE) or not os.path.exists(KEY_FILE):
        print(f"  HATA: Sertifika dosyaları eksik!")
        sys.exit()

    print(f"     Bağlanılıyor... (V8.1 - Akıllı Adres Bulucu)")

    client = Client(PLC_URL)
    client.set_user(PLC_USER)
    client.set_password(PLC_PASS)
    client.application_uri = "urn:python:client"
    client.set_security_string(f"Basic256Sha256,SignAndEncrypt,{CERT_FILE},{KEY_FILE}")
```

```
try:
    client.connect()
    print(f"   BAĞLANDI! Sürüş Başladı.")

    # --- ADRESLERİ BULMA KISMI (Burayı Güçlendirdik) ---

    # 1. TORK (Gaz Pedalı)
    node_read_torque = client.get_node("ns=6;s=Arp.Plc.Eclr/MainInstance.CMD_MotorTorque")

    # 2. SYSTEM ACTIVE (Start/Stop) - Hem Global hem Main içinde arar
    node_read_active = None
    try:
        # Önce MainInstance içine bak
        node_read_active = client.get_node("ns=6;s=Arp.Plc.Eclr/MainInstance.System_Active")
        print("INFO: System_Active 'MainInstance' içinde bulundu.")
    except:
        try:
            # Bulamazsa Global'e bak
            node_read_active = client.get_node("ns=6;s=Arp.Plc.Eclr/System_Active")
            print("INFO: System_Active 'Global' içinde bulundu.")
        except:
            print("    UYARI: System_Active HİÇBİR YERDE BULUNAMADI! (OPC tiki açık mı?)")
            print("   -> Sistem varsayılan olarak AÇIK kabul edilecek.")

    # 3. DİĞERLERİ
    try:
        node_read_friction = client.get_node("ns=6;s=Arp.Plc.Eclr/HMI_RoadFriction")
    except:
        node_read_friction = None

    try:
```

```
    node_read_brake = client.get_node("ns=6;s=Arp.Plc.Eclr/HMI_BrakePedal")
except:
    node_read_brake = None

# ÇIKTILAR
node_write_veh_speed = client.get_node("ns=6;s=Arp.Plc.Eclr/Py_VehicleSpeed")
node_write_wheel_speed = client.get_node("ns=6;s=Arp.Plc.Eclr/Py_WheelSpeed")

wheel_speed = 0.0
vehicle_speed = 0.0

print("\n--- SÜRÜŞ KURALLARI ---")
print("1. Önce HMI'dan START ver (System_Active).")
print("2. Gaz %100 ise Hız Max 220 km/h olur.")
print("-" * 50)

while True:
    try:
        # 1. OKUMA
        gas_val = float(node_read_torque.get_value())   # 0-100
        gas_val = max(0.0, min(100.0, gas_val))

        # START/STOP DURUMU
        is_active = True
        if node_read_active:
            try:
                val = node_read_active.get_value()
                if isinstance(val, bool): is_active = val
                else: is_active = (val > 0)
            except:
                pass # Okuyamazsa açık varsay
```

```
# Kayganlık
slippery_val = 0.0
if node_read_friction:
    slippery_val = float(node_read_friction.get_value())
slippery_val = max(0.0, min(100.0, slippery_val))

# Fren
brake_val = 0.0
if node_read_brake:
    brake_val = float(node_read_brake.get_value())
brake_val = max(0.0, min(100.0, brake_val))

# --- 2. MOTOR MANTIĞI ---
target_speed = gas_val * 2.2
acceleration_power = (gas_val / 100.0) * 2.2

# --- 3. DURUM KONTROLÜ ---
status = ""

# SİSTEM KAPALIYSA
if not is_active:
    status = "  SYSTEM CLOSED (STOP)"
    target_speed = 0
    if wheel_speed > 0: wheel_speed -= 0.5

# FREN BASILIYSA
elif brake_val > 20.0:
    target_speed = 0
    if brake_val > 40.0:
        wheel_speed -= (brake_val / 10.0)
        status = "   STOPPING"
    else:
```

```
        wheel_speed -= (brake_val / 20.0)
        status = "    SLOWING DOWN"

    # GAZ BASILIYSA
    elif wheel_speed < target_speed:
        diff = target_speed - wheel_speed
        wheel_speed += diff * 0.02 + acceleration_power * 0.2
        status = "  ACCELERATING"

    # SÜZÜLME
    elif wheel_speed > target_speed:
        wheel_speed -= 0.5
        status = "  "

    if wheel_speed < 0: wheel_speed = 0

    # --- 4. ZEMİN ETKİSİ ---
    grip_percent = 1.0 - (slippery_val / 110.0)
    ideal_vehicle_speed = wheel_speed * grip_percent

    if vehicle_speed < ideal_vehicle_speed:
        vehicle_speed += (ideal_vehicle_speed - vehicle_speed) * 0.1
    else:
        vehicle_speed = ideal_vehicle_speed

    if vehicle_speed < 0: vehicle_speed = 0

    # --- 5. YAZDIR ---
    dv_veh = ua.DataValue(ua.Variant(float(vehicle_speed), ua.VariantType.Float))
    dv_wheel = ua.DataValue(ua.Variant(float(wheel_speed), ua.VariantType.Float))
    node_write_veh_speed.set_value(dv_veh)
    node_write_wheel_speed.set_value(dv_wheel)
```

```
                zemin_txt = "DRY"
                if slippery_val > 25: zemin_txt = "SLIPPERY"
                if slippery_val > 70: zemin_txt = "ICY"

                print(f"{status} | Gas:%{gas_val:.0f} | Brake:%{brake_val:.0f} |         {zemin_txt} | Wheel
Speed:{wheel_speed:.0f} |          Vehicle Speed:{vehicle_speed:.0f}")

            except Exception as e:
                print(f"    Hata: {e}")

            time.sleep(0.1)

    except Exception as e:
        print(f"\n  BAĞLANTI HATASI: {e}")

    finally:
        try:
            client.disconnect()
        except:
            pass

if __name__ == "__main__":
    main()
```

5. HMI PAGE

---

6. SUMMARY

In this architecture, the **PLC** serves as the **Decision Maker** (The Brain), while **Python** serves as the **Physical Simulator** (The Body), interacting in real time over the **OPC UA** industrial protocol.