University of Applied Sciences

HOCHSCHULE EMDEN·LEER

Kommunikationsphase

Prof. Dr. Elmar Wings

Yahya Fakhet:7012464
Faissal Hammouda:7012301

WS/2021

Inhalt

- ***Kommunikationsphase***
- ***Kommunikations-Test***
- ***Applikationsvorschlag***

Kommunikationsphase

❖ ***KUKAVARPROXY und OpenShowVar***

➢ ***KUKAVARPROXY***

- KUKAVARPROXY ist ein TCP/IP-Server, der auf Netzwerknachrichten am TCP-Port 7000 hört und Daten in die KRC-Systemvariablen liest und schreibt.
- Um eine Ethernet-Verbindung (TCP/IP) herzustellen, muss zuerst KUKAVARPROXY auf dem Betriebssystem der Steuerung ausgeführt und dann die Netzwerkverbindung vom KUKA ‚HMI' konfiguriert werden

❑ Kopieren KUKAVARPROXY in das Betriebssystem auf der KRC

✓ Erhalten den KUKAVARPROXY auf https://sourceforge.net
✓ Entpacken und kopieren den Ordner auf einen USB-Stick
✓ USB-Stick an die KRC anschließen
✓ Als Experte oder Administrator anmelden: KUKA Menü -> Konfiguration -> Benutzergruppe
✓ HMI minimieren: KUKA Menü -> Inbetriebnahme -> Service -> HMI minimieren

Kommunikationsphase

- ✓ Kopieren den Ordner KUKAVARPROXY auf den Desktop
- ✓ KUKAVARPROXY.exe starten
- ✓ Wenn es ein Problem mit der Datei cswsk32.ocx gibt, verwenden den folgenden Befehl in der Eingabeaufforderung des Administrators regsvr32.exe c:\asdf\cswsk32.ocx und ändern das asdf in den tatsächlichen Dateipfad.
- ✓ Starten das Programm automatisch beim Neustart, indem eine Verknüpfung zu KUKAVARPROXY.exe in Windows Start -> Alle Programme -> Rechtsklick auf Autostart -> Öffnen definieren

❑ HMI-Netzwerk-Konfiguration

- ✓ Verbinden den Roboter mit einem Netzwerk
- ✓ Konfigurieren die IP-Adresse der KRC: KUKA Menü -> Inbetriebnahme -> Netzwerkkonfiguration

✓Entsperren Sie Port 7000: KUKA Menü -> Inbetriebnahme -> Netzwerkkonfiguration -> Erweitert

Kommunikationsphase

✓ NAT -> Port hinzufügen -> Portnummer 7000

✓Einstellen der erlaubten Protokolle: TCP/UDP

❑ Kommunikations-Anschluss: KUKA Line Interface X66

- Der X66-Anschluss auf dem Anschlussfeld ist für den Anschluss eines externen Computers für Installations-, Programmier-, Debugging- und Diagnosezwecke vorgesehen

➢ ***OpenShowVar***

- es funktioniert als Client auf einem Rechner und ist mit dem KUKAVARPROXY -Server in der KRC4-Steuerung des Roboters verbunden.
- um das python osv auf dem Rechner zu installieren, muss zuerst die Python-Bibliothek "py_openschowvar" installiert werden (pip install py_openshowar)
- Nach der Installation von py_openshowvar wird mit Hilfe von Python-Befehlen ein Test der Kommunikation zwischen dem Server und dem Client durchgeführt.

Kommunikationsphase

❖ ***Python -OPCUA***

die Installation von Python-OPCUA erfolgt durch die Installation der Python-Bibliothek „OPCUA" (pip install opcua)

➢ ***Erstellung des Python-OPCUA-Servers***

- Der Python-Code zum Erstellen des Servers lautet wie folgt

```python
from opcua import Server
from kukavarproxy import *
#create instance server opcua
server = Server()
# set end point server adresse
server.set_endpoint("opc.tcp://127.0.0.1:8080")
# register namespace of server
addspace = server.register_namespace("OPC_SIMULATION_SERVER")
# getting objects nodes from server
node = server.get_objects_node()
# creating parameters of values for server nodes
Param = node.add_object(addspace, "Parameters")
# add variable with initialisation for server parameters node
M1 = Param.add_variable(addspace,"M1",0)
# adding permission writable to the variable
M1.set_writable()
# starting the server
server.start()
```

Kommunikationsphase

➢ ***Erstellung des Python-OPCUA-Clients***

- Der Python-Code zum Erstellen des Clients lautet wie folgt

```python
import sys
sys.path.insert(0, "..")
from opcua import Client,ua

url = "opc.tcp://127.0.0.1:8080" # OPCUA Python server adresse
url2 = "opc.tcp://172.31.1.149:4840" # OPCUA Twincat server adresse

client_kuka = Client(url) #Kuka client instantiation
client_xts = Client(url2) #Twincat client instantiation

client_kuka.connect() #Kuka client connection
client_xts.connect() #Twincat client connection
print("Client Connected")
kuka_pos = client_kuka.get_node("ns=2;i=2") # ns=2;i=2 nodeID
```

Kommunikationsphase

❖ ***TwinCat 3 OPCUA Server***

➢***Bereitstellen des OPCUA TwinCAT-Servers***

- Importieren die Daten in den OPCUA-Server
- Aktivieren die TF6100-Lizenz für den Server

➢***Konfigurieren die verschiedenen Facetten des OPCUA-Servers***

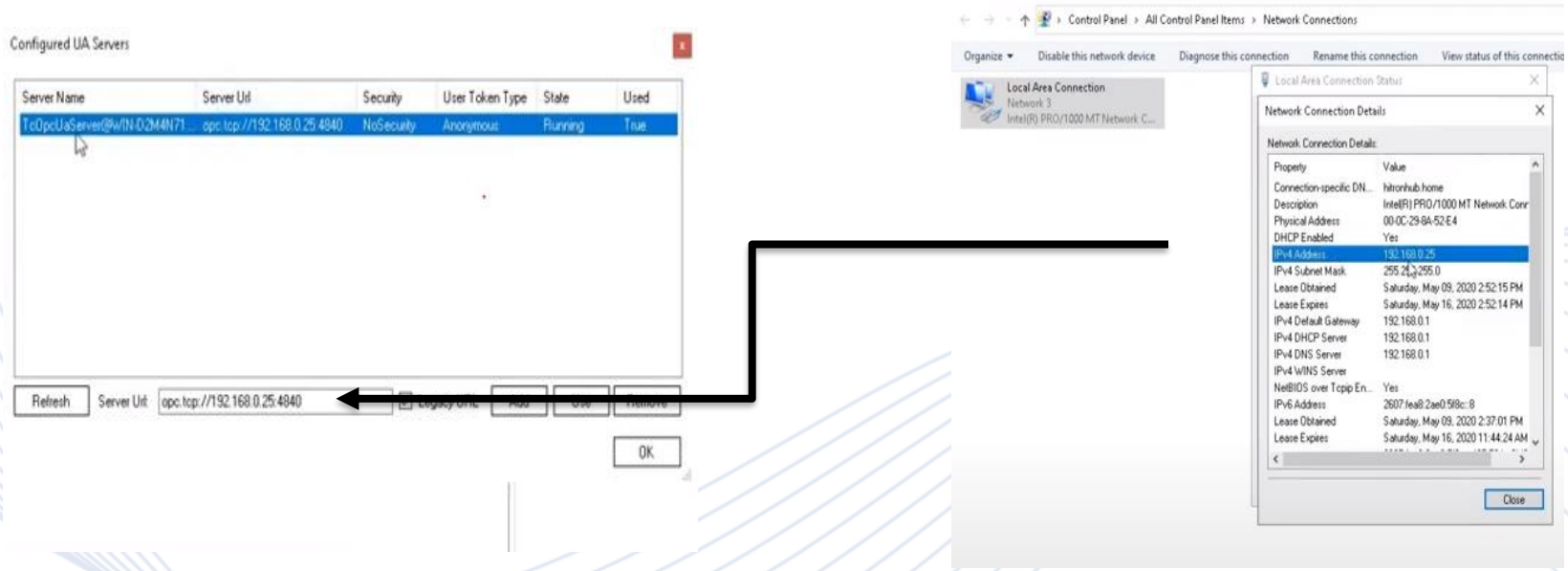
- Nachdem ein Projekt abgeschlossen ist. Es wird eine Verbindung zwischen dem OPC-UA-Server und den ADS-Geräten hergestellt.
- Es öffnet sich ein Dialogfenster, in dem die Verbindungsparameter konfiguriert werden können, z. B. AMS Net ID, ADS-Port,...

➢***Konfigurieren von Endpunkten und Zertifikatsvertrauenseinstellungen***

- Die Endpunkte des OPCUA Servers legen fest, welche Sicherheitsmechanismen beim Verbindungsaufbau eines Clients verwendet werden sollen.

Kommunikationsphase

- Aufbau einer Verbindung zum OPCUA-Server : Um eine Verbindung von einem OPCUA Client aufzubauen, muss der Client eine Verbindung mit der URL des OPCUA Servers aufbauen, z.B. opc.tcp://192.168.1.1:4840



Kommunikations-Test

❖ ***Kommunikation zwischen OpenShowVar und KUKAVARPROXY***

- um den Roboter im Automatikmodus zu steuern, sind die Variablen, die im KRL-Programm des Roboters verwendet werden, zunächst in der Datei Config.dat festzulegen.
- Nach der Installation des KUKAVARPROXY-Servers in der KUKA Robotersteuerung KRC4 kompakt wird eine Verbindung mit einem Ethernet-Kabel zwischen der Steuerung und dem Rechner hergestellt.
- Die Kodierung des Kommunikationstests zwischen dem KUKAVARPROXY-Server und dem OpenShowVar-Client ist wie folgt:

```
>>> from py_openshowvar import openshowvar
>>> client = openshowvar('172.31.1.147', 7000)
>>> client.can_connect
True
```

- ‚True' zeigt an, dass die Verbindung bereits aufgebaut ist

Kommunikations-Test

❖ ***Python-Kommunikationscodierung***

➢ OpenShowVar und Python OPCUA Server Kommunikationscode

```
1  from opcua import Server
2  from kukavarproxy import *
   #create instance server opcua
3  server = Server()
   # set end point server adresse
4  server.set_endpoint("opc.tcp://127.0.0.1:8080")
   # register namespace of server
5  addspace = server.register_namespace("OPC_SIMULATION_SERVER")
   # getting objects nodes from server
6  node = server.get_objects_node()
   # creating parameters of values for server nodes
7  Param = node.add_object(addspace, "Parameters")
   # add variable with initialisation for server parameters node
8  M1 = Param.add_variable(addspace,"M1",0)
   # adding permission writable to the variable
9  M1.set_writable()
   # starting the server
10 server.start()

   # creating kuka object and connect (with adresse ip connection )
11 k = KUKA('172.31.1.147')
```

Kommunikations-Test

1. Server importieren aus der OPCUA-Python-Bibliothek
2. importieren der KUKAVARPROXY-Bibliothek
3. Erstellen eines OPCUA-Instanzservers
4. Server-Adresse des Endpunkts einstellen
5. Namensbereich des Servers registrieren
6. Objektknoten vom Server holen
7. Werteinstellungen für Serverknoten erstellen
8. Initialisierungsvariable für den Knoten "Server-Einstellungen" hinzufügen
9. Hinzufügen der Berechtigung schreibbar für die Variable
10. Starten des Servers
11. KUKA Objekt erstellen und verbinden (mit Adresse IP Verbindung)

Kommunikations-Test

- Python OPCUA Client Kommunikationscode

```
1  import sys
   sys.path.insert(0, "..")
   from opcua import Client,ua

2  url = "opc.tcp://127.0.0.1:8080" # OPCUA Python server adresse
3  url2 = "opc.tcp://172.31.1.149:4840" # OPCUA Twincat server adresse

4  client_kuka = Client(url) #Kuka client instantiation
5  client_xts = Client(url2) #Twincat client instantiation

6  client_kuka.connect() #Kuka client connection
7  client_xts.connect() #Twincat client connection
   print("Client Connected")
   kuka_pos = client_kuka.get_node("ns=2;i=2") # ns=2;i=2 nodeID
   M = client_xts.get_node("ns=4;s=ua.M") # ns=4;s=ua.M nodeID ua.M value to be treated
```

Kommunikations-Test

1. Client UA importieren aus der OPCUA-Python-Bibliothek
2. OPCUA Python-Server-Adresse aufrufen
3. OPCUA TwinCat 3 Server-Adresse aufrufen.
4. Kuka-Client-Instanziierung
5. TwinCat3-Client-Instanziierung
6. Kuka-Client-Verbindung
7. TwinCat3-Client-Verbindung

Kommunikations-Test

- ***Beispiel-Test***

```
# Test example
# i init pos for M1 in server
1  i = 0
 # standby for client M1 action
while True:
    if i != M1.get_value():
2       i=M1.get_value()
        #sending M action to Kuka (changing M value)
3       k.write('M',i)
        # Reinit for server values to make a standby
4       i = 0
        M1.set_value(0)
```

Python-Server-Beispiel

```
while True:
        print(M.get_value())
        pos = input('set kuka pos 1/2')
        # sending kuka action value
5       kuka_pos.set_value(pos)
        #sending M twincat opcua value
6       M.set_attribute(ua.AttributeIds.Value, ua.DataValue(45))
```

Python-Client-Beispiel

Kommunikations-Test

1. Initialisierung der Position für M1 im Server
2. Warten auf eine M1-Aktion des Clients.
3. Senden der M-Aktion an Kuka (Ändern des M-Werts)
4. Serverwerte zurücksetzen, um eine Ersetzung vorzunehmen
5. KUKA Aktionswert senden
6. Senden M twinCat3 OPCUA Wert

Applikationsvorschlag

- Die Applikation benötigt die Konstruktion einer Fünf-Platz-Box und fünf kleine Würfel, ggf. auch 5 Klemmen dieser Würfel, die an den Movers befestigt werden.

Die Applikation ist wie folgt:

1/ die Box wird mit den 5 Würfeln gefüllt.

2/ die 5 Movers bewegen sich in der gleichen Gruppe. Festlegen eines Punktes auf dem Weg des Movers, an dem der Roboter beginnt, den ersten Würfel aus der Box zu nehmen und ihn an den ersten Mover zu übergeben. Danach kommt Mover 2 an die Füllposition und so weiter, bis Mover 5 gefüllt ist.

3/ In der Phase 2 der Applikation wird eine Tabelle der Movers erstellt, die es erlaubt, die Reihenfolge der Movers zu kennen und auch, ob der Mover voll oder leer ist.

4/ Nach dem Befüllen der 5 Movers erfolgt eine Entladung der Würfel, die sich in den 5 Movers befinden, an den 5 Stellen der gebauten Box.

University of Applied Sciences

HOCHSCHULE
EMDEN·LEER

Vielen Dank für Ihre Aufmerksamkeit