ITIA Target Programming (Robot) Lab Report

Stefan Adelmann (01633044) and Hannes Brantner (01614466)

May 17, 2020

1 SysML Review

Since the robot station was already designed in an singular SysML description, no two projects had to be merged in this stage. As a review two parts of the description where modified in this phase.

1.1 Parametric diagram

The parametric diagram described the calculation of the line distance between two points in 3D space. This calculation however is not part of the user programmable environment for the robot arm. The movement of the arm is controlled by internal algorithms that maximize the efficiency of the robot. Due to the building characteristics of an robot arm the most efficient way is not a straight line, rendering the distance calculation presented in phase 1 obsolete. To create a streamlined description the parametric diagram was removed in phase 2, leaving only relevant diagram types.

1.2 Internal Block Diagram

The internal block diagram should display the connectivity between components. For the programming part it would be required in reality to model this down to the individual components of the station, like for example the spring push out piston. This was omitted and abstracted in the description by only specifying the connectors that are present on the robot. If a developer needs to know the pin that is used for a specific part a chart or specification of the connector standard can be found in the datasheet of the robot controller. The internal block diagram is therefore regarded as the viewpoint of a system integrator that needs to connect modules but not individual components.

2 I/O Mapping

The Robot station primarily consists of the robot arm with a multi-functional gripper and sensors and actuators located in two modules. Since the robot arm is specifically connected and controlled by the robot controller it is separated in this report from the I/Os of the other two modules namely the assembly and handling stations.

2.1 Robot

The location of the robot arm in 3D space is controlled by 6 freedom axis. It is important to note that the coordinate system of the robot can be changed on the controller. For this report the XYZ mode was chosen. The basis of the coordinate system can be seen in 1. The first 3 parameters of the move command shown in 2.1 specify the X, Y, and Z coordinate of the arm. The last 3 A, B, C describe rotations around the coordinate system axis. In XYZ mode the basis of rotation is the same as the basis of the xyz coordinate system. In other

movement modes this however can be changed, a thorough list of coordinate system modes can be found in the datasheet of the robot controller.
Due to the inherent collision danger embossed by the remote connection and camera angle it was not safely possible to test all movement patterns of the robot. The move commands shown in 2.1 where however tested by entering precollected arm positions that where known to be collision free.

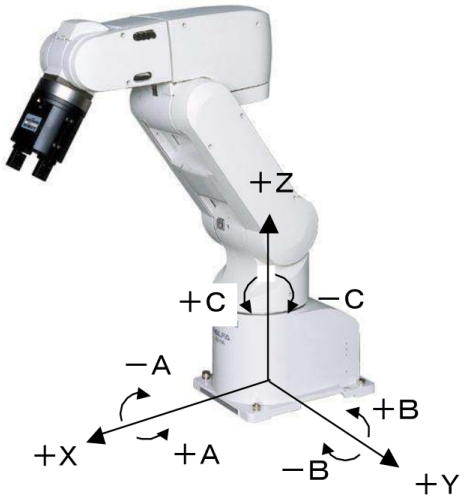

Figure 1: Robot coordinate system in XYZ mode

Additional to the movement commands in the robot programming language Melfa-V, methods running on an OPC UA server where implemented in this phase. A detailed description of these methods can be found in section 3 of this report.

Melfa-Command	**OPC UA**	**Beschreibung**
Mov(X,Y,Z,A,B,C)	move(X,Y,Z,A,B,C)	Modul Roboter (Hand) - Position, Rotation anfahren
HOpen 1	gripperOpen()	Modul Roboter (Hand) - Gripper öffnen
HClose 1	gripperClose()	Modul Roboter (Hand) - Gripper schließen

2.2 Robot Control

The robot itself as well as the modules present in the robot station are controlled by a robot controller. Programming is done via the programming language

Melfa-V. Connection with the controller can be established via the parameters shown in table 2.2.

Controller	
Device:	Robot Controller
ID:	CR750-D
MAC:	
IP:	192.168.162.82/25

To gain the possibility of integrating the robot station into the other station in this lab, an OPC UA server running on a Raspberry Pi was improved in this phase. Connection to this server can be established via the parameters shown in table 2.2.

OPC UA Gateway	
Device:	RaspberryPi 3
ID:	BCM2835 (a02082)
MAC:	b8:27:eb:09:db:ca
IP:	192.168.162.84/25
Port:	4840

2.3 Sensors/Actuators

The sensors and actuators are connected to the robot controller via the connectors shown in the internal block diagram of the SysML description. On the software side of the controller inputs and outputs are addressed by an index. As it can be seen in the tables 2.3 and 2.3, there is not distinction between input and output purely by looking at the index. This is handled in Melfa-V by two different function calls, `M_out(0)` will address output at index 0 while `M_In(0)` will address the input.

Sensoren	
Index	**Beschreibung**
1	Modul Roboterhandling - Werkstück ausgerichtet
2	Modul Roboterhandling - Werkstück in Abholposition
3	Bedienfeld - Start (Schließer)
4	Bedienfeld - Stopp (Öffner)
5	Bedienfeld - Reset (Schließer)
7	Bedienfeld - COM Brücke (I7)
8	Modul Robotermontage (Federmagazin) - Schieber eingefahren
9	Modul Robotermontage (Federmagazin) - Schieber ausgefahren
10	Modul Robotermontage (Federmagazin) - Feder vorhanden
12	Modul Robotermontage (Deckelmagazin) - Schieber eingefahren
13	Modul Robotermontage (Deckelmagazin) - Schieber ausgefahren
15	Modul Robotermontage (Deckelmagazin) - Deckel auf Ablage
900	Modul Roboter (Hand) - Teil nicht schwarz

The color sensor shown in the last line of table 2.3 is in a different index range as the other inputs, this is caused by its location on the robot arm itself. These inputs and subsequently the outputs of the arm are already predefined in the controller software, while the function of the general purpose pins of the other sensors and actuators can be chosen by the developer.
It can be seen in both tables that indices are missing. These missing pins are connected to boards on the robot station modules but not assigned any function. If the robot station is improved, this in and outputs can be used.

Aktoren	
Index	**Beschreibung**
0	Bedienfeld - Start (LED)
1	Bedienfeld - Reset (LED)
2	Bedienfeld - Q1 (LED)
3	Bedienfeld - Q2 (LED)
4	Bedienfeld - COM Brücke (Q4)
8	Modul Robotermontage (Federmagazin) - Schieber ausfahren
12	Modul Robotermontage (Deckelmagazin) - Schieber ausfahren

3 Handover Protocol

In the robot station the whole operation is carried out by one controller, a handover protocol is therefore not needed to implement the assembly process. The robot could however be integrated into the other stations in another role. For this use case, a previous bachelor thesis implemented a basic interface for controlling the robot via OPC UA. In its original form it was for example possible to start and stop the operation of the robot. In this phase this server was updated and new methods where added.
The methods that were implemented are shown in table 3. By calling these methods in combination with the tables provided in the previous section a different controller like one of the PLCs in this lab can control the robot via the OPC UA interface.

Methods	
Method	**Function**
OpenGripper()	Opens the multi function gripper
CloseGripper()	Closes the multi function gripper
Move(X,Y,Z,A,B,C)	Moves the robot arm to the given coordinates
GetErrorLog(NUMLOGS)	Returns the last NUMLOGS errors that occurred
ReadInput(INDEX)	Returns the state of the input at the given index
WriteOutput(INDEX, STATE)	Sets the output at the given INDEX to the given STATE
ResetError()	Resets the robot controller from the error state

While it is possible with these methods to implement the assembly behavior of the demo robot implementation certain settings like the coordinate system have to be done manually, otherwise the move command could crash.
In addition to the active control methods, two important error methods were implemented. During use of the robot arm, internal safeguards of the robot arm or syntax errors will switch the robot into error mode preventing its use. `ResetError()` can be used to revert the robot into its operational state. A list of the last errors can be retrieved with the `GetErrorLog(NUMLOGS)`. A call will return the last NUMLOGS errors, with additional information like timestamp, error-code and error text and in which program/line the error occurred.