SI4

TwinCAT 3 Source Code Documentation

By Luuk Aarts, Dimitar Dyulgerov

Date: 15 Jun 2023

Abstract

The purpose of this report is to go over the source code of the PLC application controlling an Igus Mover5 robot arm. It will control this robot based on input received from a digital twin which is connected to the BECKHOFF PLC via the BECKHOFF ADS protocol

Table of contents

Introduction

In order to control the robot arm in an industrial setting, a Beckhoff PLC is used, running TwinCAT 3. The PLC is responsible for translating user-defined movement to proper data for the robot arm and moving it to the required location.

At the end of reading this document you will understand the responsibilities of each class/FB that the source code uses to achieve its goal. References to other documents will explain the more exact details even further, so that the code can be reused and adjusted by the next group responsible for the Igus Robot Arm.

Procedure

General code description

The TwinCAT 3 project aims to be modular, debuggable and easily modifiable. It is OOP oriented, using Function Blocks (abbreviated FB from now on) as classes, with their respective methods and properties that enable them to be interacted with from wherever they are called.

The MAIN FB can be considered as the main program loop. There, the UI is updated, button presses are read and the different handlers are called.

Communication

This section is dedicated to the communication method used for the project. Please see the "CANopen vs Ethernet" document for the research behind our decision to use Ethernet as a communication method.

CANopen

The initial plan was to use CANopen as the communication method between the PLC and the robot arm. Due to the inexperience of the team with TwinCAT 3 at the time, the process started off slow and not much progress was made. That was partly because of the lack of proper documentation on how to use the EL6751 CANopen module. Most of the documentation provided by Beckhoff is related to the electrical or already implemented features of the module, and not as much as how to set it up and use it to communicate.

The thing that convinced the team to start looking in another direction for the communication protocol was that not even Igus-provided CANopen demo software worked with the arm. The arm itself was not responding as it should, according to its own documentation, therefore the issue was deemed too big to reliably solve in the time frame for the project and we moved on to another option.

TCP/IP

The robot arm can also be controlled via a TCP/IP connection, via a socket client. The robot arm hosts the socket server. That method of communication was tested and successfully moved the robot arm via provided code and the Igus IDE, therefore was deemed reliable and possible.

The TCP/IP communication logic is housed in the TCPClient Class. It is a wrapper class which encapsulates the socket client logic and exposes only the crucial parts of it. Under the hood, it uses an implementation of the TF6310 software package for the connecting, sending and receiving data. The state machines for the TCPClient class can be found in the accompanying design document, along with a class diagram.
Implementing the socket client logic was more cumbersome and difficult than expected, especially when compared to a language like C/C++. That is mostly due to the fact that a

PLC cannot afford to have blocking functions, therefore everything is non-blocking. That creates challenges of its own. In the end, however, we managed to establish a reasonably abstracted communication layer.

Message History

In order to make debugging easier, messages received from the robot arm are stored in an array and displayed in a table in the visualisation. It displays up to 9 messages at a time. When more than 9 messages have been read, every message is shifted up, making space for the new one. That way, we always have the 9 latest messages to observe. The design and interaction is not ideal, but that is due to the very limited visualisation tools provided by TwinCAT 3.

Message Handling

The COMMUNICATION FB will take care of creating and handling the string from or for the robot arm. This will implement the commands and data for the robot into a string the Robot can read. The COMMUNICATION FB will check whether the software is running on Robot or Monarco mode and send the data to either the Create or Handle FB with the correct parameters for the mode it is in. There is a whole document "Findings_IgusRobotCommunication.docs", about the protocol and form of the string that the message handler should provide.

Arm control

Controlling the arm is done with its own protocol, for the exact guideline for controlling the robot we made a separate Document, "Findings_IgusRobotCommunication.docs". Basically the robot needs to receive a heartbeat with a minimum of every 2 seconds. This heartbeat will consist of the current movements of the robot arm. We are controlling the robot in this project by moving the arm for a period of time that we calculate.

The robot arm is moved via the ArmMovementHandler FB, via a finite state machine, the diagram for which can be found in the "design" document. In summary, the robot arm waits in an idle state, in its true-zero position (all joints are set to 0 degrees of rotation). Once new position data is acquired, it moves to the required position, if possible, and waits there for 5 seconds, simulating an interaction with the product. Afterwards, it goes to the drop off point, stays there for 5 more seconds, again, simulating interaction with the product, and then goes back to its true-zero position in the idle state, waiting for new data once more.

The PLC also keeps track of the robot arm's position, as polling the arm for its current values was unsuccessful.

Encountered gotchas

1. **You (probably) need an isolated core to run TwinCAT 3 locally**
   In the beginning phase of the project, a lot of people had problems running TwinCAT 3. One of the main issues was that TwinCAT 3 does not work well with windows' Hyper V or virtualization technologies. The easiest way to solve this issue is going into the TwinCAT 3 project, navigating to System/Real Time, clicking "Scan target" and adding 1 isolated core via the prompts. (See instruction video playlist in the references for a TwinCAT 3 quickstart guide)
2. **FB methods are similar to functions**
   Explanation: In the TCPClient class, the reading and writing functionality to the socket server are split into methods. It was wrongly assumed that FB methods keep their state between calls, similar to how FBs behave. After some debugging, it was discovered that, in fact, they function more akin to functions.
3. **The Ethernet IP address of your laptop has to be static and in the same range**
   In order to connect to the BECKHOFF PLC, the laptop/device you are connecting to the PLC or the IGUS robot needs to be in the same IP range as both devices. The static IP should be '192.168.3.xxx', where 'x' can be any number. The PLC's IP is, as of testing, 192.168.3.2, so it is best to avoid that one when setting the laptops' IP.

Conclusion

In conclusion, in spite of the difficulties faced during the development process of the TwinCAT 3 application, we managed to deliver a working product which moves the robot arm to the required positions, with the added feature of some debugging tools, such as a display for received messages.

References

TwinCAT 3 Getting Started Playlist
Beckhoff TwinCAT 3 courses