SI4

Testing report

By Rik van Heesewijk, Luuk Aarts, Dimitar Dyulgerov, Jorn Kersten

Date: 12 May 2023

Abstract

The purpose of this report is to go over the testing process of the robot arm project, an Industry project for Smart Industry, Semester 4.

Table of contents

Introduction

In order for the end product to meet the proper criteria, it needs to be tested in the expected conditions, provided by the client. This document describes the tests performed in that direction.

PLC/TwinCAT 3

Test Case Table

Nr	Test Case	Result	Solution
1	Control the robot via CANopen	Could not establish CAN/CANopen communication	Move on to another communication protocol
2	Communicate with the robot arm via TCP/IP	The client and server communicate as expected	none
3	Control the movement of the robot	The by sources provided move commands did not work	Moving to position using timers.
4	Calculate joint angles for a target position	The robot moves to the required position	none
5	Move the robot to a pickup location, then to a dropoff location, and finally to its default location	The robotic arm went through the pick-and-place sequence as expected	none* *see details*

Detailed test descriptions

1. CANopen

Test case: Control the robot via CANopen

CANopen was tested with the provided P-CAN adapter and a C# application, written by the developers of the robotic arm. No proper communication could be made - the application would get stuck waiting for a response that never came.

Afterwards, CAN commands were manually sent via the P-CAN software that came with the usb-to-CAN adapter. It was determined that the robotic arm does not respond in accordance to its CAN protocol, therefore the C# application could not start up. False responses meant that the robotic arm could not be sensibly controlled via CANopen in its current state.

Furthermore, the CANopen and CAN integration in TwinCAT 3 were poorly documented, with few to no examples.

Result: Could not establish CAN/CANopen communication

2. TCP/IP socket client

Test case: Communicate with the robot arm via TCP/IP

Since CANopen was dropped as a viable communication method, the team moved to TCP/IP communication, via a socket client/server. The robotic arm hosts the socket server, the PLC needs to connect to it as a client and send commands.

The socket client was tested by developing a windows–based socket server in C++. The C++ server would simply echo back the commands that it received. Both the TwinCAt 3 client and the dummy server worked as expected and were deemed viable. Communication was tested both locally, both the client and server running on one machine, and remotely - the client running on the beckhoff PLC and the server running on a laptop, connected via an ethernet cable.

Result: the client and server communicate as expected

3. Movement

Test case: Control the movement of the robot

According to the developers of the Igus robot, the robot arm should move by sending the given commands. While testing these commands, the robot returned us errors which stated they did not recognize some of the variables.

Since there was no good documentation or any community help for our problem, we decided to go for the end goal to move the robot arm and started moving the robot on the basis of time and distance. While testing and trying to debug the problem we had found that the solution was to use the static movement we could set per joint by changing the alive message.

By timing the movement we could calculate the time it should move to move to a given position. This was the solution that we went for since it had no problems while testing at all.

Result: The robot can move each joint to the given position.

4. Inverse Kinematics

Test case: Calculate joint angles for a target position

A simple inverse kinematics algorithm was developed to calculate the relative joint positions to get to a target cartesian position. The calculator was first developed in C++, as it was the fastest way the developer could get a working prototype working. The output values for each joint were then plugged into the PLC code responsible for the robot arm movement, making the arm move by the required degrees per joint. The algorithm proved to work accurately enough, with an estimated error of +-1 to 3 degrees.

Afterwards, the algorithm was translated to Structured Text. The output was compared to the C++ output. It was slightly different, and after increasing the PI accuracy (by using a user-defined constant instead of the built-in one) and fixing minor errors, the output became the same.

Finally, the ST algorithm was integrated into the PLC code and tested again, with the same position.

Result: the robot moves to the required position.

5. Pick-and-Place action sequence

Test case: move the robot to a pickup location, then to a dropoff location, and finally to its default location

The pick-and-place sequence was initially tested by directly running it on the robotic arm. That was unsuccessful, however, as the robot arm would not stop and honour the interaction time with teh object (staying still for 5 seconds at the end position). To determine the issue, the state machine was tested locally, disconnected from the robot, by putting breakpoints throughout its code. After fixing the not waiting for 5 seconds issue, we encountered another one.

The robotic arm would not move past the pick up location, on to the drop off point. The reason for that was the fact that the arm would be considered not busy the first program cycle after it was given a setpoint (and assumed that it would start returning busy). That was

fixed by calling the robotic arm movement loop in the on-entry state of each state of the pick-and-place state machine.

Result: the robotic arm went through the pick-and-place sequence as expected.

Unity

Test case table

Nr	Test Case	Result	Solution
1.	The conveyor belt runs smoothly.	It runs smoothly.	None
2.	Objects spawn at a random position.	Objects spawn at a random position, but the depth is hard coded.	Change the min and max z value in the parameters.
3.	Objects get deleted when they reach the end of the conveyor belt.	When the objects fall off of the conveyor belt the objects get deleted and they respawn again.	None
4.	The objects get detected in a certain area.	An area above the conveyor belt detects the objects on the conveyor belt and adds them to a list.	The communication via ADS from Unity to TwinCAT doesn't work anymore
5.	When the objects get detected the conveyor belt stops.	If the robot arm is ready for the next pickup, the conveyor belt stops.	When the communication via ADS from Unity to TwinCAT works, the conveyor belt stops.
6.	The location of the objects is sent to the TwinCAT code.	When an object is detected the conveyor belt stops.	None
7.	Unity can communicate with TwinCAT.	It can communicate via ADS but it can't read a certain variable anymore	Haven't found a solution to fix this problem.

8.	The conveyor belt changes size accordingly from the settings of the main menu.	When using different sizes than the default ones, it shapes accordingly.	None
9.	The conveyor belt speed can be set from the settings of the main menu.	When using a different speed, the speed changes to the correct one.	None
10.	When the visualisation is running the speed of the conveyor belt can be changed.	The speed can be changed by pressing the escape button on the keyboard. But when the conveyor belt is stopped the belt won't go to the correct speed anymore.	Create a check over the update function in the conveyor belt script. Then change speed to faster speed if the belt is running again.

Integration

Test case table

Nr	Test Case	Result	Solution
1.	Robot arm receives position	Unity sends the correct positions to TwinCAT using ADS	None
2.	Robot arm moves to given position	The arm moves to the correct position	None
3.	Conveyor belt stops if robot arm is ready for new cube	It stops when the robot arm sends the ready command	None
4.	The conveyor belt waits until the robot arm is done	The conveyor belt continues after giving the arm the positions	Add a variable to TwinCAT to compare where the arm is so Unity Will know what is happening
5.	The conveyor belt starts again when robot arm is done	The conveyor belt continues after giving the arm the positions	None

Conclusion

In conclusion, most of the testing was done manually, by either running the program to be tested and observing the results. Due to the limited time and the fact that the whole team was new to TwinCAT 3, we could not make use of automated testing methods (such as unit tests)