Motion control solution for new PLC-based standard development platform for VLT instrument control systems

D. Popovic*[a], R. Brast[a], N. Di Lieto[a], M. Kiekebusch[a], J. Knudstrup[a], C. Lucuix[a]
[a]European Southern Observatory, Karl-Schwarzschild-Str. 2, 85748 Garching bei München, Germany

ABSTRACT

More than a decade ago, due to obsolescence issues, ESO initiated the design and implementation of a custom-made CANbus based motion controller (CAN-RMC) to provide, together with a tailor-made software library (motor library), the motion control capabilities for the VME platform needed for the second generation VLT/VLTI instruments. The CAN-RMC controller has been successfully used in a number of VLT instruments but it has high production costs compared to the commercial off-the-shelf (COTS) industrial solutions available on the market today.

In the scope of the selection of a new PLC-based platform for the VLT instrument control systems, ESO has evaluated motion control solutions from the company Beckhoff. This paper presents the investigation, implementation and testing of the PLC/TwinCAT/EtherCAT motion controllers for DC and stepper motors and their adaptation and integration into the VLT instrumentation framework. It reports functional and performance test results for the most typical use cases of astronomical instruments like initialization sequences, tracking, switch position detections, backslash compensation, brake handling, etc. In addition, it gives an overview of the main features of TwinCAT NC/PTP, PLCopen MC, EtherCAT motion control terminals and the engineering tools like TwinCAT Scope that are integrated into the development environment and simplify software development, testing and commissioning of motorized instrument functions.

Keywords: VLT, COTS, PLC, motion control, EtherCAT, PLCopen, OPC UA, TwinCAT

1. INTRODUCTION

The original motion control solution for the Very Large Telescope (VLT) instruments, developed in the early ‘90s, was based on the VMEbus standard and the Motorola 68k family of CPUs. The motion controllers, for both DC and stepper motors, came from the company MACCON. More than a decade ago MACCON announced discontinuation of this product line and ESO was looking for another solution in order to support the second generation of VLT/VLTI instruments. At the time there was no suitable COTS solution and ESO embarked on a lengthy process of developing its own motion control solution as a replacement for MACCON. The last VLT instrument to use MACCON controllers was KMOS.

The new motion controler, called CAN-RMC, was partially developed with the help of an external company. The abbreviation CAN-RMC comes from **CAN**bus **R**emote **M**otion **C**ontroller. The system is based on the CANbus. A CANbus controller resides in the VME crate of the Local Control Unit (LCU) and communicates with the motion controller. The new controller has solved the obsolescence problem and has relatively easily been integrated into the existing VLT Instrument Software Framework. Upgrades from the MACCON to the CAN-RMC solution turned to be simple. The first MACCON based VLT instrument to be upgraded (addition of new motorized function) was VIMOS. The second generation VLT instruments MUSE, SPHERE, GRAVITY and MATISSE, including the Adaptive Optics (AO) module GALACSI, all base their motion control on the CAN-RMC solution.

Although the main characteristic of the CAN-RMC controller is a very high performance, there are a number of important issues that have been analyzed and have influenced the decision to develop a new COTS based motion control solution. The most important ones are:

*dpopovic@eso.org; phone +49 89 3200 6533; fax +49 89 3202 362; www.eso.org

- The controller was an in-house development and is a customized solution.
- The development took close to ten years.
- The controller comes only in a 4-channel configuration. This means that even for a very simple and low demand control the most expensive solution has to be used. In addition, the 4-channel controller, as the only option, is to be used even in situations where a single motor has to be controlled.

2. PLC-BASED SOLUTION

The PLC Motion Control project is part of a much larger PLC project whose aim is to move from the VME/VxWorks based VLT control system to a more modern PLC/EtherCAT based system. This became possible with the introduction of the VLTSW Fieldbus Extension.[1] The initial evaluation of the EtherCAT technology[2] was done as part of the E-ELT Technology Demonstrator project.[3] Encouraged by the initial results and by the simplicity in developing PLC control application the suitability of the PLC-based motion control was officially evaluated.[4]

The new system has to meet the following top level requirements, as defined in [5]:

- The system shall be integrated into existing VLTSW framework.
- The system shall use user units.
- The system shall be PLCopen MC compliant.
- The control interface shall be compliant with [6].
- The interface with hardware devices shall be compliant with [7].
- The system shall implement a velocity control structure.
- The system shall implement a position control structure.
- The system shall support tracking in both position and velocity mode.
- The system shall be capable of performing a predefined initialization sequence.
- The system shall be capable of handling axis with brakes.
- The system shall support backlash compensation.
- The system shall be configurable.
- The system shall provide telemetry data.
- The system shall provide system identification capabilities.
- The system shall be local controllable.
- The system shall be capable to control DC, Stepper and BLDC.
- The system shall support linear and circular optimized axis.

From the HW requirements point of view the system shall support motor power supplies of 12-24 V with the maximum current of 6 A. Both TTL/24 V and differential (RS422) encoder feedback signals shall be supported. Stepper motors shall be with 2-phases and shall be possible to use micro stepping.

Figure 1 shows examples of the existing and the new solution. The PLC based solution comprises an Embedded PC and a number of I/O terminals attached directly to the PLC or to remote couplers. The system is characterized by a relatively low heat dissipation that allows placing of the I/O modules close to the hardware. The EtherCAT topology is very flexible and supports true distributed systems where couplers can be up to 100 m apart from each other.

There are numerous advantages in using the PLC technology. The most important one is the use of open standards like EtherCAT, PLCopen (including the standardization of the Motion Control (MC) library), and the OPC UA communication protocol. We are trying to avoid proprietary protocols that eventually lead to vendor lock in. By using COTS components the effort of developing customized hardware and software is minimized and the variety of HW choices is much bigger making it easier to fight obsolescence. This results in cost-effective solutions with short development cycle and low development costs.

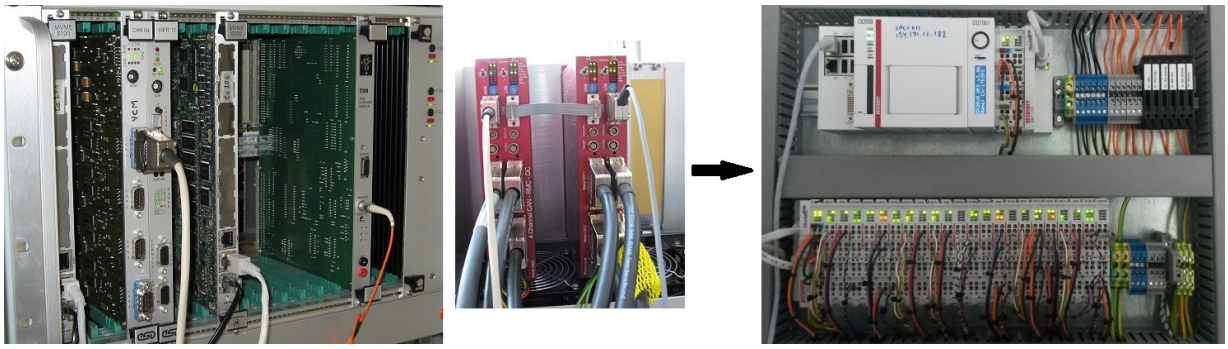

Figure 1. Moving from the VME/VxWorks based solution (left) to the PLC/EtherCAT based system (right). In the VME system the communication between the CPU and the CAN-RMC controller is done via CANbus. The PLC test setup comprises a PLC and a coupler hosting motor output stages, external encoders and I/O terminals.

From the hardware point of view, the PLC solution has lower costs, better compatibility and lower heat dissipation. Due to the lower cooling requirements the cooling water consumption and the vibrations introduced from the cooling fans can be reduced.

3. DEVELOPMENT AND IMPLEMENTATION DETAILS

3.1 Standards

The PLC motion control application presented in this paper is based on a number of open standards and technologies. The main ones are: *EtherCAT*[8], an Ethernet-based real-time high performance fieldbus system; *PLCopen*[9] technical specifications around the IEC 61131-3 standard as well as the Motion Control (MC); and the *OPC Unified Architecture*[10] (OPC UA) communication interface that was evaluated in [11].

3.2 Development environment

Both hardware and software used for the development are coming from the company Beckhoff. For the moment we use the CX2030 embedded PC as the proposed standard. We have already had experience with Beckhoff CANbus based I/O hardware since it is one of our current standards for VME based VLT control systems. Planning ahead of time, it would be possible to upgrade the existing I/O modules to EtherCAT by just replacing the CANbus coupler that hosts the terminals with the equivalent EtherCAT coupler without the need for any re-wiring.

The TwinCAT 3 (TC3) IDE is used for the development of control software. TC3 is integrated into Microsoft Visual Studio and supports plug-ins for SVN. It includes the compiler for all IEC 61131-3 languages. Our development is done in Structured Text (ST), a language similar to C and Pascal. Among other features, the most important ones are the support for C++ and modules generated by Matlab/Simulink, and the integrated debugger for all IEC 61131-3 languages as well as for C++. With the C++ support it is possible to compile and run existing C/C++ code on the PLC. We have already reused some parts of the LCU code implementing the mathematical algorithms to calculate the compensation for field rotation needed for tracking devices.

The TC3 IDE includes an analysis tool called Scope View. With Scope View variables from the TwinCAT system can be recorded and displayed graphically. This tool is essential for system debugging and in particular for motor tuning.

3.3 Licenses

PLC applications for motor control require runtime licenses. There are three options: up to 10 axis, up to 25 axis and the unlimited number of axis controlled by a single PLC. The other licenses are for the PLC runtime, OPC UA Server, and, if needed, for C/C++ and Mathlab/Simulink. All PLC runtime licenses are paid for only once, i.e. there is no annual fees.

3.4 Data Model

The motor data model is optimized for interfacing via OPC UA. The data are organized in four groups: configuration, control, information and status. The grouping simplifies the view from OPC UA clients (see Figure 2). In addition there are a number of variables that have to be mapped (linked) to the corresponding physical I/O signals. They include limit switch signals, brake control I/O and the motor axis.

```
FUNCTION_BLOCK FB_MOTOR_CONTROL
VAR
    (* OPC UA Interface *)
    cfg:     T_MOTOR_CFG;    (*~ (OPC : 1 : Config  params) *)
    ctrl:    T_MOTOR_CTRL;   (*~ (OPC : 1 : Control params) *)
    info:    T_MOTOR_INFO;   (*~ (OPC : 1 : Info    params) (OPC_PROP[0005] : 1  : OPC_PROP_RIGHTS, RO) *)
    stat:    T_MOTOR_STAT;   (*~ (OPC : 1 : Status  params) (OPC_PROP[0005] : 1  : OPC_PROP_RIGHTS, RO) *)

    (* Variables that need mapping *)
    switches         AT %I*:  T_MOTOR_SWITCHES;  (* Limit switches *)
    i_bBrake_Stat    AT %I*:  BOOL;              (* Brake feedback signal - DIGITAL IN *)
    q_bBrake_Ctrl    AT %Q*:  BOOL;              (* Brake control  signal - DIGITAL OUT *)

    Axis:    AXIS_REF;
END_VAR
```

The above code shows an extract from the variable declaration of the main Function Block (FB) called `FB_MOTOR_CONTROL`. In the Beckhoff implementation OPC UA variables are declared with the special comments that are understood only by the Beckhoff OPC UA Server. This way of variable declaration is specific to Beckhoff but is still compatible with implementations from other vendors since the declarations are seen by them as ordinary comments. Structures *info* and *stat* are read only.

Figure 2 shows the view of the OPTI1 motor namespace as seen from an OPC UA client. It is easy to navigate through the namespace and inspect the content of any of the four data structures, as well as to modify the values of variables that are not read only, i.e. *cfg* and *ctrl*.

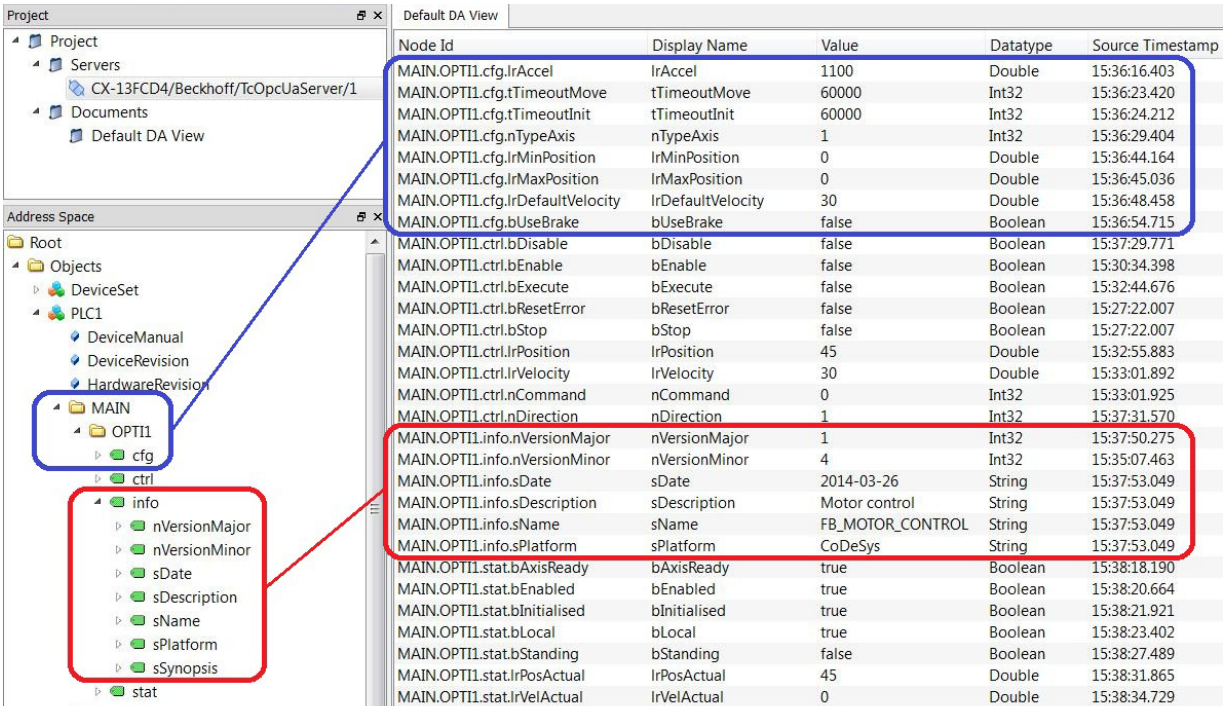

Figure 2. OPC UA client view of the OPTI1 motor namespace.

3.5 Control Interface

The control and monitoring of motors in operation is normally done from the Linux motor control application via the OPC UA communication interface by simply reading and writing of the variables in the OPC UA namespace. The same control can be achieved manually in a number of ways: from an OPC UA client, from the TC3 debugger or from a dedicated GUI running in the TC3 environment. The control interface is defined in [6].

For example, to move the motor to position 180 deg at the speed of 20 deg/sec the following variables will have to be written in given order:

```
ctrl.lrPosition = 180      # Set new position
ctrl.lrVelocity = 20       # Set new velocity
ctrl.nCommand   = 3        # Set command to 'move absolute'
ctrl.bExecute   = TRUE     # Trigger motion
```

3.6 ESO Motor Control Library

The ESO Motor Control Library contains the control code and the templates for GUIs. The control code is provided in a single FB called FB_MOTOR_CONTROL.

4. PLC MOTOR CONTROL APPLICATION DEVELOPMENT

This section briefly describes the most important steps in the development of PLC applications for motor control based on the ESO Motor Control Library with the aim to demonstrate how simple this process can be.

4.1 Adding References/Libraries

After the PLC project is created and before starting any development all required libraries have to be added to the PLC program References using the Library Manager. There are two ESO libraries: Common.library and Motor.library that have to be included. Common.library contains routines that are common to all ESO PLC projects. In addition the OPC UA library is required as well. The complete motion control code is provided in ESO TC3 Motor.library that contains a single FB called FB_MOTOR_CONTROL. This FB also encapsulates the OPC UA interface. Figure 3 shows the included libraries in the project.

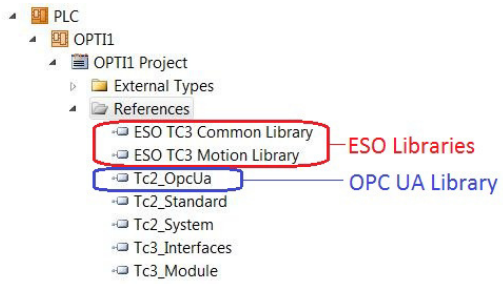

Figure 3. Additional libraries for motor control PLC applications. The four libraries: Standard, System, Interfaces and Module are the standards libraries that are automatically included in the project.

4.2 Writing PLC program

Writing PLC motor control applications is very simple. An instance of FB_MOTOR_CONTROL has to be instantiated in the declaration section and then called in the execution section of the PLC program. Thanks to the encapsulation of all the functionality in the motor library, a motion control application can be implemented with just a few lines of code. Figure 4 shows the complete PLC program for the control of a single motor called OPTI1.

```
PROGRAM MAIN
VAR
    (* Function block declaration *)
    OPTI1:  FB_MOTOR_CONTROL;
END_VAR
```

```
(* Function block execution *)
OPTI1();
```

Figure 4. Complete PLC program for control of a single motor called OPTI1.

For more motors it is just needed to add one instance of FB_MOTOR_CONTROL per motor and to execute them. Figure 5 shows two possible ways of doing it. One way is to declare each individual motor while the other approach is to declare all motors in a single array. The advantage of the former method is that each motor can be named individually, e.g. OPTI1, OPTI2, FILTER1, FILTER2, etc, while the latter method is more compact.

MAIN_10 × | OPTI1 | Scope NC Project | GUI_CFG

```
VAR
    (* Function block declaration *)
    OPTI1:  FB_MOTOR_CONTROL;
    OPTI2:  FB_MOTOR_CONTROL;
    OPTI3:  FB_MOTOR_CONTROL;
    OPTI4:  FB_MOTOR_CONTROL;
    OPTI5:  FB_MOTOR_CONTROL;
    OPTI6:  FB_MOTOR_CONTROL;
    OPTI7:  FB_MOTOR_CONTROL;
    OPTI8:  FB_MOTOR_CONTROL;
    OPTI9:  FB_MOTOR_CONTROL;
    OPTI10: FB_MOTOR_CONTROL;
END_VAR
```

```
(* Function block execution *)
OPTI1();
OPTI2();
OPTI3();
OPTI4();
OPTI5();
OPTI6();
OPTI7();
OPTI8();
OPTI9();
OPTI10();
```

MAIN_10_ARRAY × | MAIN_10 | OPTI1 | Scope NC Project

```
PROGRAM MAIN_10_ARRAY
VAR
    (* Function block declaration *)
    OPTI:   ARRAY [1..10] OF FB_MOTOR_CONTROL;
    i:      INT;
END_VAR
```

```
(* Function block execution *)
FOR i := 1 TO 10 DO
    OPTI[i]();
END_FOR
```

Figure 5. Two possible approaches to writing PLC programs for control of a number of motors. Motor instances can be declared individually (left) or in an array (right).

4.3 Motor configuration

The motor configuration (see Figure 6) is divided into three groups that define parameters for the hardware configuration, motor tuning (axis configuration) as well as the ones specific to the motor as an instrument function.

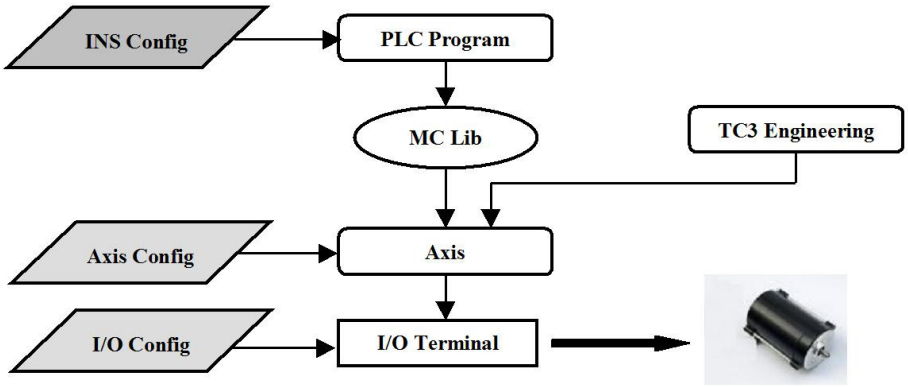

Figure 6. Distribution of motor configuration.

The first two groups are part of the TC3 project and are downloaded to the PLC together with the PLC application. The third one is included in the standard VLTSW instrument configuration and the parameters are written to the PLC during the PLC device, i.e. motor, initialization. Through this it is possible to write a generic PLC FB that can control any type of motor with any kind of configuration.

I/O Terminal Configuration is done through the CoE (**C**AN **o**ver **E**therCAT) configuration interface of the motor control output terminal. Parameters to be configured here are, for example, maximum current, holding current, feedback type and encoder resolution. These parameters should be carefully set prior to any attempt to move the motor in order to prevent harm to people or the hardware. Changes of the value take effect only after the INIT-OP transition of the terminal. Figure 7 shows an example of the way these parameters are set in TC3 for the Beckhoff stepper motor output stage EL7041.

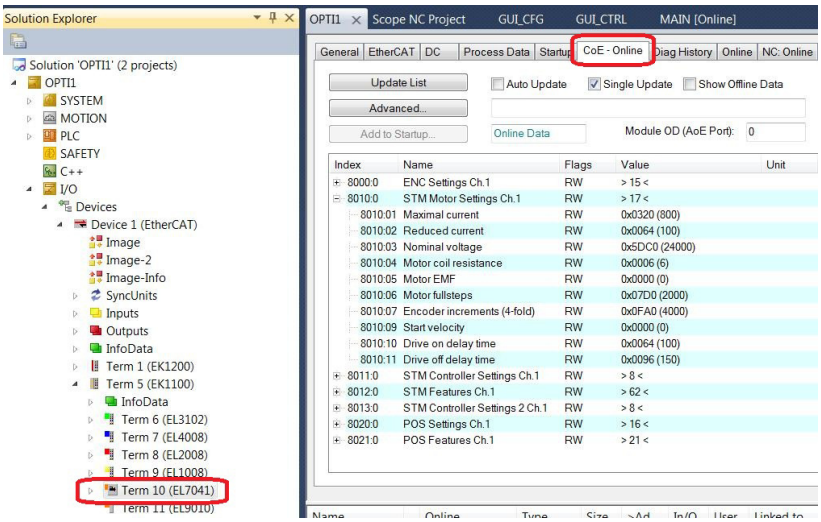

Figure 7. CoE configuration of motor output stage.

It is important to be able to replace an I/O module without having to re-configure it. In TC3 projects the I/O terminal configuration parameters that have to be persistent can be added to a so-called Startup List. Figure 8 shows a number of stepper motor parameters in such a list. The configuration of an I/O module can be exported as well. This way new projects can simply import previously tested configurations and therefore ensure continuous and safe operation.

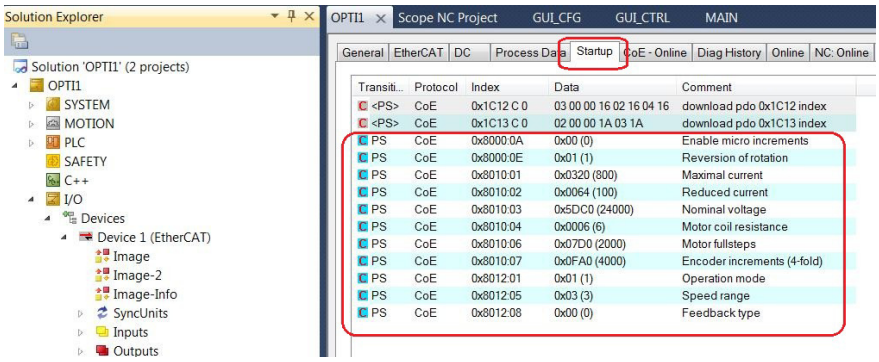

Figure 8. Crucial CoE configuration parameters can be saved in a startup list in order to ensure simple and safe replacement of the motor output stage.

Axis Configuration defines motor control and feedback parameters like scale factor, acceleration, deceleration, jerk, PID parameters, following error limit, in-position window, etc. The TC3 IDE provides a comprehensive tool for tuning

motors and setting of axis configuration parameters. Screenshots of this interface are given in Figure 9. Under the *Online* tab, it is possible to move the motor and monitor its status, including the following error and the control output. The *Dynamics* tab is used for defining the dynamic characteristics of the motor and it will calculate acceleration, deceleration and jerk based on the given time to reach the maximum velocity and the required response (stiffness) of the system. Under the *Functions* tab, it is possible to execute a number of pre-define repetitive motions in order to test the response of the motor. The axis configuration can be exported and re-used for other instances of the same type of motor in the very same project or in some other projects that control the same type of motor.

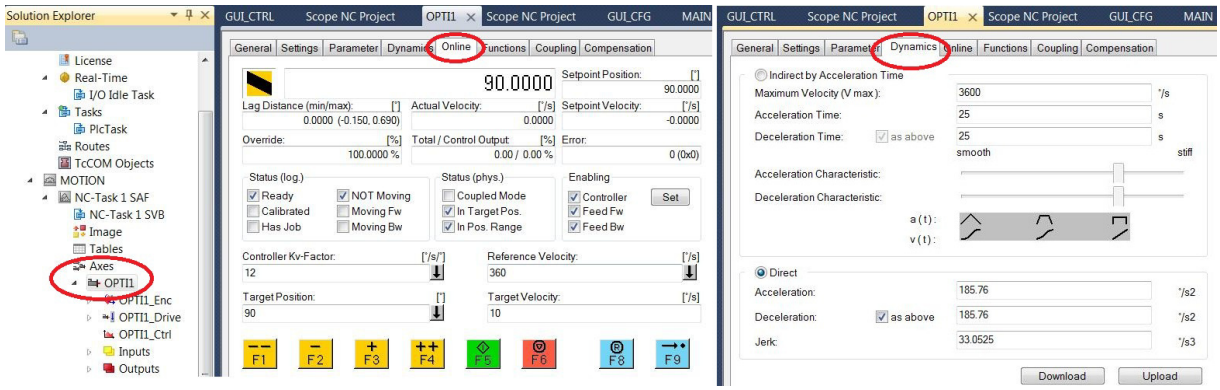

Figure 9. TC3 interface for initial motor axis configuration and tuning. The interface can be used without any PLC code. It is only necessary to link the axis to the motor output stage. Features like automatic dynamics configuration are provided (right).

Instrument Function Configuration includes the configuration parameters like axis type (e.g. linear or circular-optimized), backlash compensation, INIT sequence, active level of limit switches, etc. These parameters are not related to the way the motor is controlled but rather to the way the motor is used as an instrument function. The best example is the definition of the initialization sequences for a number of motors of the same type but installed at different instrument locations. Although the motors will have identical I/O and Axis configurations (dynamics) they will most likely be initialized differently due to, for example, different position of limit switches or home positions. Some of these parameters are often set during calibration. Parameters from this configuration group are defined on the Instrument Workstation and are downloaded to the PLC for each motor on every INIT of that motor. Some examples of configuration parameters, including the definition of INIT sequence, are given below.

```
#*****************************************************************************
# Device: OPTI1
#*****************************************************************************
INS.OPTI1.INITTOUT    120000;          # Initialization timeout
INS.OPTI1.MOVETOUT    120000;          # Move timeout
INS.OPTI1.BACKLASH    5.0;             # [UU] Backlash compensation
INS.OPTI1.MINPOS      44.0;            # [UU] Min position
INS.OPTI1.MAXPOS      308.0;           # [UU] Max position

#
# Initialization Sequence.
#
# - Find Upper HW Limit: coarse vel = 20 deg/sec, fine vel = 1.5 deg/sec
INS.OPTI1.INITSTEP1     "FIND_UHW,20.0,1.5"
# - Move relative -220 deg with vel = 30 deg/sec:
INS.OPTI1.INITSTEP2     "MOVE_REL,30.0,-200.0"
# - Find Z (index) pulse: coarse vel = -5 deg/sec, fine vel = 2 deg/sec
INS.OPTI1.INITSTEP3     "FIND_INDEX,-5,2"
# - Set detected Z-pulse position to 180 deg:
INS.OPTI1.INITSTEP4     "CALIB_SWITCH,180.0,0.0"
# - Make to absolute position of 250 deg with vel = 30 deg/sec:
INS.OPTI1.INITSTEP5     "MOVE_ABS,30.0,250.0"
```

Figure 10 shows the above defined initialization sequence recorded with the Scope View. In this particular case it is possible to analyze each initialization step as well as the behavior of the motor.

Figure 10. Scope View recording of the initialization sequence.

4.4 Variable mapping

The remaining task is to link the motor Axis to the physical I/O terminal and the PLC program Axis variable. In addition, if used, the switches and brake control I/O signals have to be linked (mapped) to the corresponding PLC program variables. Figure 11 shows how the mapping is done from the TC3 IDE. It requires just a few mouse clicks. For each link the system provides a list of available selection choices. In the case of a single motor there is only one choice per link.

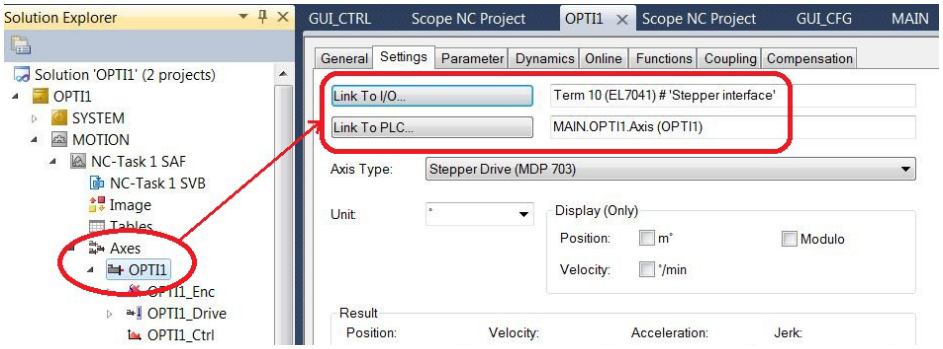

Figure 11. Mapping of motor Axis to motor output module and PLC program.

4.5 Controlling the motor

The basic configuration and control of the motor can be done from the GUIs running in the TwinCAT environment or on dedicated touch panels. The GUIs, shown in Figure 12, are instances of GUI templates provided in the ESO Motor Control Library.

Once the motor has been tested at the PLC level, the next step is to test the OPC UA interface prior to integrating it into the VLT instrument. The interface can be tested, as described in Section 3.5, using any OPC UA client.

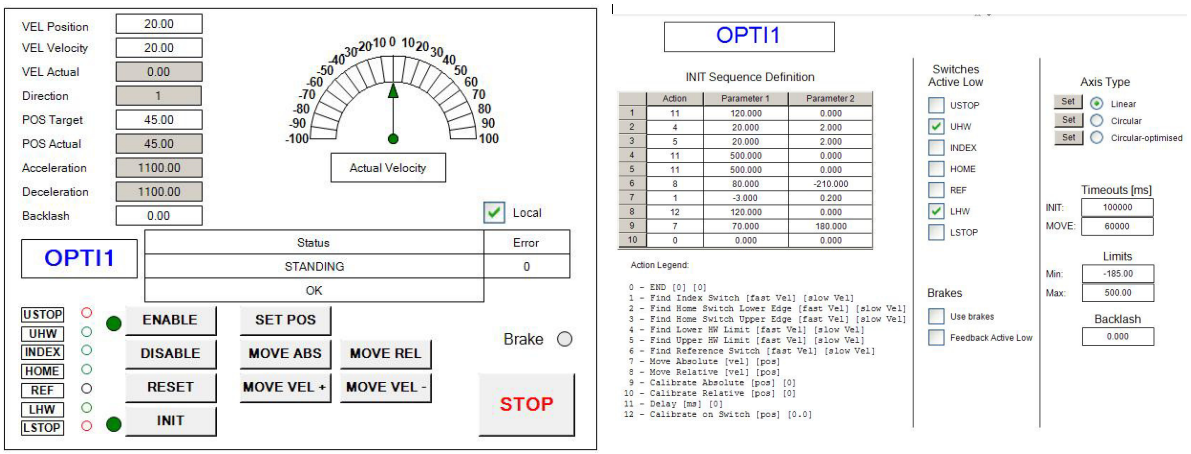

Figure 12. GUIs for control and configuration of motors are provided as templates. Each instance is created by setting the motor reference and giving the motor name.

5. CURRENT PROJECT STATUS

The goal of the PLC Motion Control project is to fulfill all motor control requirements for VLT Instruments. So, the absolute minimum is to match the existing functionality of the VxWorks based control system. The following sections describe the current status of the most important features of the system.

ESO Motor.library: The complete motor control code is provided in a single FB (FB_MOTOR_CONTROL). The FB supports all types of motors and is hardware independent. The FB has been tested with a number of stepper and DC motor output terminals, also in combination with both TTL and differential encoders. The differences in hardware are handled internally by the MC_* function blocks or in the configuration. Switches and brake control I/O signals are fully configurable and can be mapped to any digital I/O signal that can reside either on the motor output terminal or on dedicated digital I/O terminals. In addition, the active level of each digital I/O signal is individually configurable. This feature is normally used for hardware limit switches where it is desirable from the safety point of view that the active level is reversed so failures like broken wires can be detected.

SLOC: The total number of source lines of code written in ST (including comments and spaces) is currently around 3000 lines. This is to be compared with the SLOC contained in the VxWorks based solution that has around 90000 lines. This will significantly reduce the maintenance effort in the future.

Axis Types: Linear, circular and circular-optimized axis types are supported. Circular axes are treated as linear with the only difference that the software limits are not mandatory. For circular-optimized axes the movement is always done the shorter way taking into account the backlash compensation when applicable.

Operational modes: The system supports a number of operational modes: position, velocity and tracking (i.e. following the motion of objects in sky coordinates, including field rotation) in position & velocity. In tracking mode position and velocity can be updated at each PLC cycle. It is also possible to switch from one mode of operation to another without stopping the motor.

Backlash compensation: The system supports backlash compensation. There is a single configuration parameter that can have the value zero (no compensation), < 0 (negative backlash) or > 0 (positive backlash). For calibration purposes this value can be changed at runtime.

Brakes handling: The system supports motors with brakes and feedback. In cases where feedback is not possible our solution is to wire the control output signal directly to the feedback input.

User-configurable initialization sequence: The system supports a comprehensive set of initialization steps. Up to ten steps can be defined per motor. This includes detection of limit, index and home switches. Switches can be any digital input signal with either normal or reversed logic. During initialization motors can be moved to an absolute position or relative to the current one. The position can be calibrated so that the current position is set or that the position is set for

the location where the switch was detected. The latter method has significantly increased the positioning accuracy and repeatability. We have not performed any high precision measurements but the ESPRESSO team has reported the positioning repeatability of 2 microns for their DC motors. The accuracy of the switch detection as well as its repeatability during motor initialization is very important for the operation of the motor. It depends on the speed at which the motor moves during the switch detection step. The introduced inaccuracy can be easily estimated. Since the switch detection is active on every PLC cycle, the maximum introduced error is equal to the distance that the motor travels during one PLC cycle.

For example, for the fine speed of 0.1 mm/sec and the PLC cycle of 1 ms, the introduced positioning error resulting from the reference switch detection can be less or equal to:

$$\text{error} = 0.1 \text{ mm/sec} * 0.001 \text{ sec} = 0.0001 \text{ mm} = 0.1 \text{ micron}$$

Therefore, the reference switch position detection error is probably an order of magnitude smaller than the other influential factors in the control system and therefore in many cases could be ignored.

Templates for control and configuration GUIs: The ESO Motor Library provides templates for the control and the configuration of motors. In order to create a GUI the PLC programmer only has to create instances of the template and set references to the motor name and the PLC application motor instance.

Persistent Configuration: As previously mentioned, during the initialization of a motor, a subset of the configuration is downloaded from the high level software running on a Linux machine to the PLC. This is fine assuming that both the Linux and the PLC software are available. However, at the very early stage of the instrument control project, most likely the higher level software is not available and there should be another way of configuring the system. In addition, it is necessary to keep the last motor configuration after a power cycle. From the provided Configuration GUI it is possible to configure all the parameters that would normally come from the workstation. After each successful initialization of the motor the configuration is saved into NOVRAM. After a power cycle the configuration is read back from NOVRAM on the first PLC cycle.

6. REMAINING TASKS

This section lists a number of issues that we are aware of and will have to be solved and implemented.

Scalability tests: We still have to perform scalability tests in order to ensure that high number of motors can be properly controlled from the same PLC and that the PLC is not overloaded. The maximum number of motors tested so far was six.[2] Currently we are in the process of testing ten motors, eight DC and two stepper, from a single embedded Beckhoff PLC CX2030.

User-defined functionality: We are planning to support execution of user defined sets of instructions that are not available in the generic library. This is mostly related to pre and post-INIT, or pre and post-MOVE functionality. This is still under investigation. One option would be to involve the high level software as well, in order to simplify implementation.

Support for brushless motors: The intention is to have a single FB for all type of motors. So far, we have tested stepper motors with and without encoder feedback and DC motors with incremental encoders. Brushless motors are to be tested next. Furthermore, encoder feedback of SSI or resolver type will be tested.

Test of motor controllers from different vendors: So far we have tested motor controllers from the company Beckhoff. To take the full advantage of the EtherCAT technology and the PLCopen standards, we are planning to test motion controllers from other EtherCAT and PLCopen certified vendors that are specializing in this area, e.g. *maxon*. This way we could have a list of recommended hardware solutions for specific applications where the best possible performance is desirable.

Better understanding of TwinCAT NC controller internals: We feel that we don't fully understand and therefore don't utilize the full capabilities of the integrated TwinCAT NC controller (position controller running on the PLC). We are working closely with Beckhoff on this.

Completion of support for tracking devices: The concept of tracking has been successfully tested. We were controlling motors at the position or velocity update rate of 10 Hz. These tests were based on the concept where the sky

transformations were calculated on the instrument WS and the updates were sent to the PLC. However, our preferred option is to do all calculations in the PLC in order to create an autonomous system. We would like to reuse the *slalib* C code in TC3 and run it on the PLC. This work still has to be finalized in order to provide a simple interface similar to the one used for non-tracking motors.

Finalization of GUI templates: The GUI templates provided in the ESO Motor library are still at the prototype level. They will have to be finalized.

7. CONCLUSIONS

The PLC Motion Control project is progressing well and according to the planning. The initial results are quite encouraging and we are confident that the PLC solution will be a complete replacement for the existing VME/VxWorks based solution and that it will bring many benefits by using open standards and COTS solutions, in particular when it comes to fighting obsolescence as well as the maintenance of the code in the future.

The PLC application is well integrated into the VLT SW Framework. The basic functionality needed for most of the VLT instruments is already provided. With the supported axis types and the motion modes, the library can be used for control of discrete and continuous functions, filters, mirrors, tracking devices like derotators and ADCs, etc. In combination with the higher level software more complex functionality can be achieved.

If proven successful, this technology and the motor control solution could be used for the E-ELT instruments.

REFERENCES

[1] Knudstrup, J. T., "ICB - Fieldbus Extension Design Description," ESO internal document ESO-043730, (2011)
[2] Popovic, D., "E-ELT Technology Demonstrator Report: EtherCAT Evaluation," ESO internal document ESO-192170, (2009)
[3] Wallander, A., "Technology Demonstrator Specification," ESO internal document ESO-193698, (2007)
[4] Knudstrup, J. T., "PLC Based Motion Control Technical Report," ESO internal document ESO-043750, (2012)
[5] Brast, R., Popovic, D., “PLC Motion Control for VLT Instruments Requirements Specification,” ESO internal document (in preparation), (2014)
[6] Popovic, D., Knudstrup, J. T., Kern, L., Jost, A., “VLTSW/ICB PLC Motion Control Interface Control Document,” ESO internal document ESO-041690, (2011)
[7] van Kesteren, A., “Technical specification for Electrical and Electronic Design,” ESO internal document ESO-044295 (2014)
[8] EtherCAT Technology Group, http://www.ethercat.org/
[9] PLCopen, http://www.plcopen.org/
[10] OPC Unified Architecture, https://opcfoundation.org/about/opc-technologies/opc-ua/
[11] Kiekebusch, M., "E-ELT Technology Demonstrator Report: OPC Unified Architecture Evaluation," ESO internal document ESO-191573, (2009).