Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

DEVELOPMENT AND IMPLEMENTATION OF A CONTROL CONCEPT FOR mHLS PROTOTYPES AND THE COMPLETE mHLS PLANT

A THESIS

Submitted in fulfillment of the

requirements for the award of the degree

of

MASTER OF ENGINEERING

in

MECHATRONICS ENGINEERING

by

Deepak Chopra

Matriculation Number: 869946

UNIVERSITY OF APPLIED SCIENCES

KAISERSLAUTERN, GERMANY

Hochschule
Kaiserslautern
University of
Applied Sciences

Supervisor: Prof. Dr.-Ing. Rainer Fremd

Co-supervisor: Dipl.-Ing. Dmitrij Laaber (DLR)

Co-supervisor: Dr.-Ing. Kai Wieghardt (DLR)

Beginning: 01/09/2015

Submission: 26/02/2016

Registration Number: 8701

Candidate's Declaration

I, Deepak Chopra, a student of Master of Engineering in Mechatronics Engineering, in the Department of Applied Engineering, University of Applied Sciences, Kaiserslautern with Matriculation No. 869946, for the session 2014-2016, hereby, declare that the dissertation entitled **"DEVELOPMENT AND IMPLEMENTATION OF A CONTROL CONCEPT FOR mHLS PROTOTYPES AND THE COMPLETE mHLS PLANT"** has been completed by me in 4th semester.

I hereby, declare that:

i. The matter embodied in this Dissertation is an original work and has not been submitted earlier for award of any degree to the best of my knowledge and belief. Moreover, the Dissertation does not breach any existing copyright or any other third party rights.

ii. The Dissertation contains no such material that may be illegal and offensive.

I hereby agree to indemnify HS Kaiserslautern and its teaching staff against any and all losses incurred in connection with any claim or proceeding assert plagiarism and/or copyright infringement if the investigation carried out determines that my work is the plagiarizing or infringing work.

Date: **Deepak Chopra**

Certificate

This is to certify that the thesis entitled **"DEVELOPMENT AND IMPLEMENTATION OF A CONTROL CONCEPT FOR mHLS PROTOTYPES AND THE COMPLETE mHLS PLANT"**, being submitted by **Deepak Chopra** under Matriculation No. 869946 to the Department of Applied Engineering, University of Applied Sciences, Kaiserslautern, Germany for the award of Degree of 'Master of Engineering' in Mechatronics Engineering, is a bona fide work carried out by him under my supervision and guidance.

His thesis has reached the standard of fulfilling the requirements of regulations relating to degree. The thesis is an original piece of research work and embodies the findings made by the research scholar himself.

Date: **Prof. Dr.-Ing. Rainer Fremd**

Abstract

This report presents the reader with the methodology undertaken to develop a control system for the mHLS prototypes and the plant at German Aerospace Center in Jülich. The scope of the Thesis project includes the functional as well as the safety design of the control system along with the programming using the open international standard IEC 61131-3 for programmable logic controllers.

mHLS (german for- *Modularer Hochleistungsstrahler,* or in English- *Modular high-flux emitter)* under construction currently would be the largest facility in the world to simulate and research on concentrated simulated solar radiation. The plant consists of 149 identical modules, each one equipped with a short-arc Xenon lamp that provides light with a spectrum similar to sun light. Such an experimental facility, considering the sheer scale of radiative output in Ultraviolet region, heat flux and electric components operating well above 100 amperes, requires a fail-safe control design.

Following a hazard and operability analysis, the operation functions of the plant were aligned with a safety strategy. The well-known standard IEC/EN 62061,"Safety of machinery: Functional safety of electrical, electronic and programmable electronic control systems," was referred to during the process. Since the operation of the plant includes motion of over 447 stepper motors, operating 149 high power rectifiers for short-arc Xenon lamps and other low power devices such as lasers and cooling fans; the possible modes of personnel injury or equipment loss include, uncontrolled motion, unintentional power to rectifiers and cooling failure, to name a few.

The development of the controller thus includes programming of a number of safety functions apart from providing the user with comfortable Graphical User Interface (GUI) to access and operate all the electrical appliances in the plant. The current work also includes the development of an algorithm to detect collision between different modules of the plant and another algorithm to generate set points for module motion. The coding is done using the Structured Text Language of IEC 61131-3 and is included in the report.

"I would like to thank my supervisors for their guidance, my family for their support and Sun god."

Table of Contents

List of Figures

List of Tables

Nomenclature

W	=	Watts
kg	=	Kilograms
V	=	Volts
A	=	Amperes
Nm	=	Newton-meters
mm	=	Millimeters
θ	=	Greek symbol Theta used for angular distance
ϕ	=	Greek symbol Phi used for angular distance

Abbreviations

HVAC	=	Heating Ventilation and Air Conditioning
I/O	=	Input/Output
SIL	=	Safety Integrity Level
IEC	=	International Electrotechnical Commission
NEMA	=	National Electrical Manufacturers Association
PLC	=	Programmable Logical Controllers
STL	=	Structured Text Language
GUI	=	Graphical User Interface
ADS	=	Automation Device Specifiation
FIFO	=	First In First Out
TÜV	=	Technischer Überwachungsverein

1. Introduction

The Institute of solar research, DLR is constructing a new facility to simulate and research on concentrated solar radiation with a planned radiative output of over 300 kW. The facility would become the largest of its kind and remarked as *SynLight,* the world's largest artificial Sun [1]. This facility would provide new possibilities to research on uses of concentrated solar radiation in scientific applications and as an alternate power source for the future.

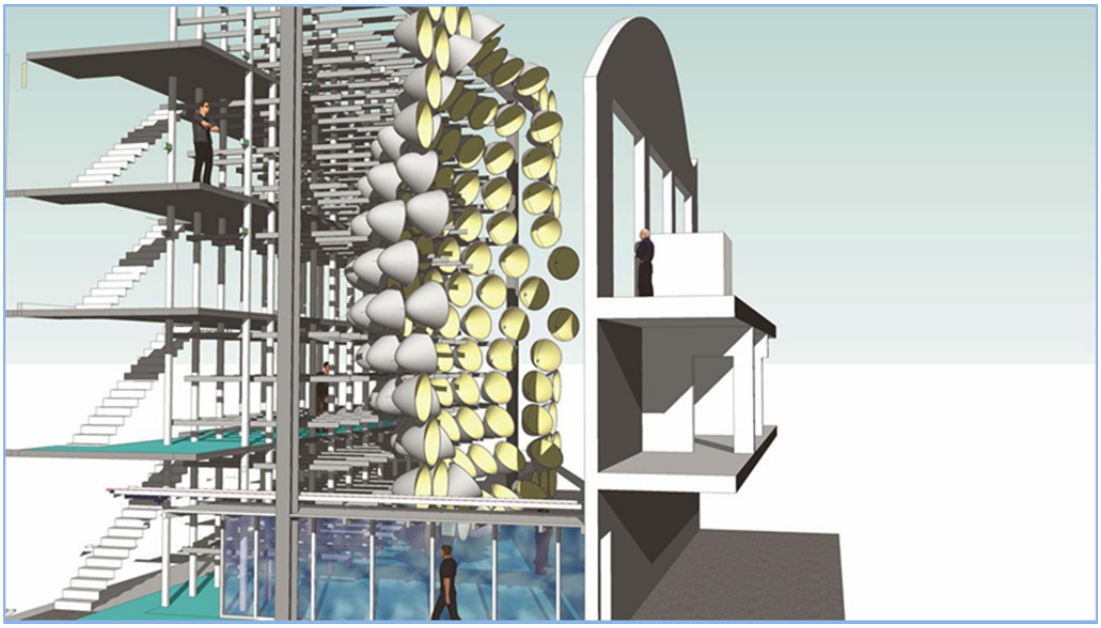


Figure 1: Planned mHLS plant

The current Master thesis is focused on developing the computerized control system for the facility and the prototypes at the DLR labs. Following successful implementation on the prototype and positive design review, the control system along with the selected hardware is to be implemented in the facility.

Following chapters would demonstrate the activities undertaken and methods followed during the course of thesis and their corresponding outputs. An overview of the activities is as follows:

- Brief hazard and operability analysis for the facility
- Development of safety concept and safety integration levels (SIL)
- Function description specification
- Programming operation and safety functions in TwinCAT 3 environment
- Programming for further integration with other building control systems (cooling and door management)
- Developing and programming an algorithm to generate set points for automated movements of the modules
- Developing and programming an algorithm to prevent collision between individual modules during manual or automated movement
- Testing control software on prototype module

- Optimization and documentation

2. Control parameters

The facility consists of 149 identical individual modules. The term control parameter herein refers to the physical state to be achieved by different parts of the assembly during operation. Each of these modules is equipped with a short-arc Xenon lamp that provides light with a spectrum similar to sunlight. The light from each of these modules is to be focused on a target which houses other scientific experiments such as a hydrogen generator being currently designed. The short-arc Xenon lamp is fitted inside an ellipsoid reflector with high quality aluminum coating. The reflector is mounted on a mechanism that provides three axial movements (one linear and two rotational). Following control parameters are of interest [2]:

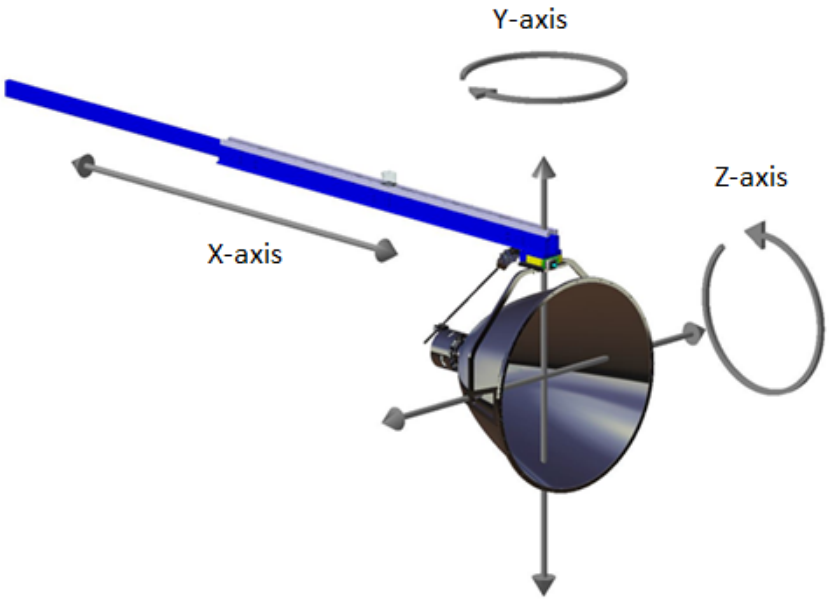


Figure 2: Position axes for reflector

2.1 Reflector position

The reflector position is to be controlled on three axes with dedicated mechanisms. The mechanism for each axis is explained in this section:

2.1.1 X-axis

A telescopic arm is used to provide the module with linear movement in X-axis as demonstrated above. The telescopic arm is a rack and pinion gearing mechanism driven with a bipolar stepper motor (2 A, 2 Nm). The torque requirement to move approximately 30 kg parts with this mechanism is less than 2 Nm and has been calculated in previous studies [2]. The motors are selected in such a way that the necessary torque needed for the motion is provided and there are no step-losses on account of dynamic torque requirements.

2.1.2 Y-axis

A rotary table is used to provide the rotation around Y-axis as shown in the figure below. The rotary table is used at the end of telescopic arm and is a standard unit powered with a 0.8 Nm stepper motor. The rotary table has a gear ratio 120:1. With micro-stepping up to 1/8, high accuracies can be achieved in positioning with this mechanism. The rated voltage is 5.4 volts for each stepper motor.

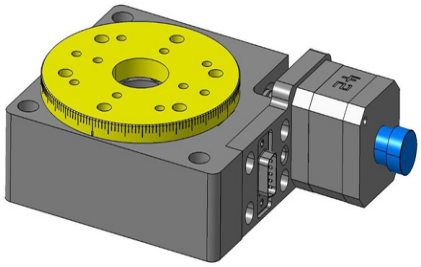
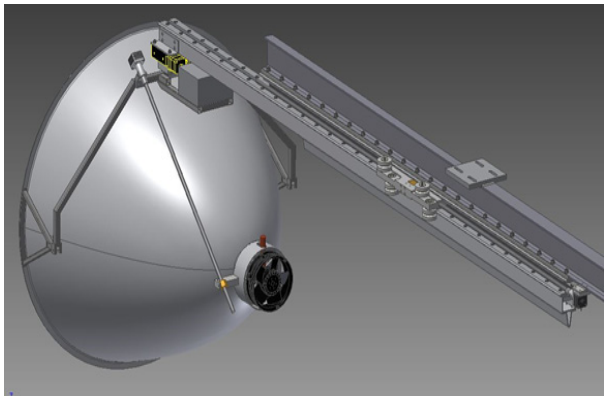


Figure 3: Rotary table mechanism

2.1.3 Z-axis

A spindle is used to provide elevation to the module. The spindle is a threaded screw of pitch 2mm and diameter 14 mm mounted at an angle of 60 degrees to the horizontal with one end fixed to the telescopic arm and other free to rotate inside a nut mounted near the center of reflector to provide a tilting effect in vertical direction. The spindle is coupled with a stepper motor of same rating as the rotary table.



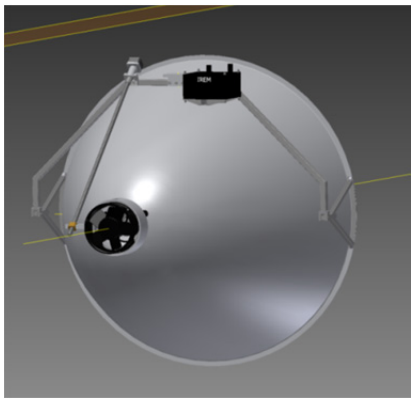
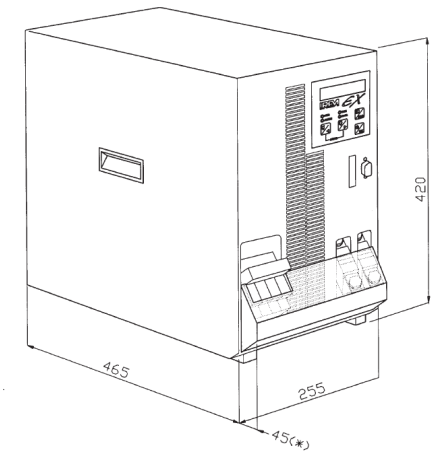


Figure 4: Spindle for elevation

2.2 Lamp power

The short-arc Xenon lamps belong to the class of gas discharge lamps. The principle is to produce light by passing electricity through xenon gas at high pressure. The arc length so produced is few millimeters but with a high power density of the order of few thousand Watts. The lamp used in current configuration operates at 40 V DC and 165 Amperes. For such high power requirement, a dedicated rectifier for each such lamp is used[3].



5a) Rectifier

5b) Lamp

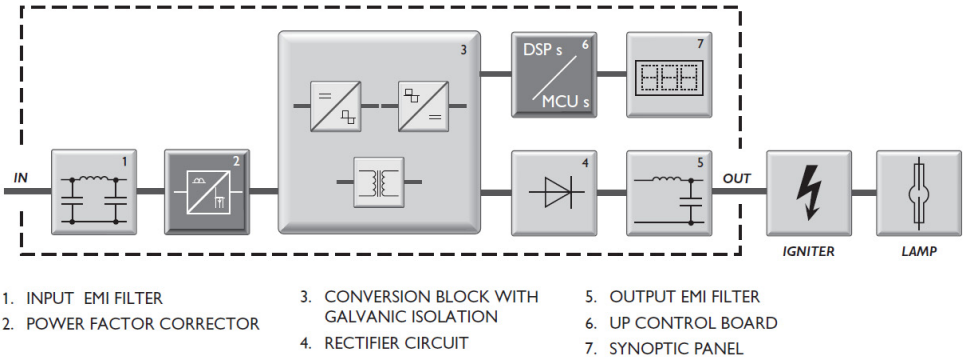


Figure 5: Lamp and rectifier working principle

The control of these rectifiers can be carried out either through analog dry contacts, or by means of the digital interface over RS 232 protocol. The control of the rectifier parameters, and in particular the electric current, is required for the correct lamp operation in all conditions.

Rectifier parameters	Range
Max. output power	10800 Watts
Output current range	80-210 Amperes
Output voltage range	30-58 Volts DC
Communication	RS 232 serial line (vendor's protocol)
Ripple	< 1%
Temperature control	Internal

Table 1: Rectifier parameters [3]

2.3 Cooling

An enormous heat flux generation is expected during the experiments that has to be removed through appropriate heat discharge system. The cooling load is divided over centralized cooling for the plant and individual cooling for the modules.

Individual cooling is required for successful operation of the lamps which is dependent on ambient temperatures. Following table shows the important operational parameters considered in design:

Lamp operation parameters	Range
Maximum permissible base temperature	230°C
Cooling	Forced cooling / fan
Min. air flow velocity around discharge vessel	6 m/s

Table 2: Lamp operation parameters [4]

As such a variable speed fan is mounted at the base of each lamp to provide the above mentioned conditions of operation. The speed of these fans is controlled through the supply voltage 0-10 Volts.

2.4 Lasers

To focus a module on the target wall, there are three lasers mounted on each reflector which should be powered on or off at desired time. They are set in such a way that an optimum distance of 8 meters between each reflector and target can be achieved for intended heat flux. The lasers require 2.5 Volts DC.

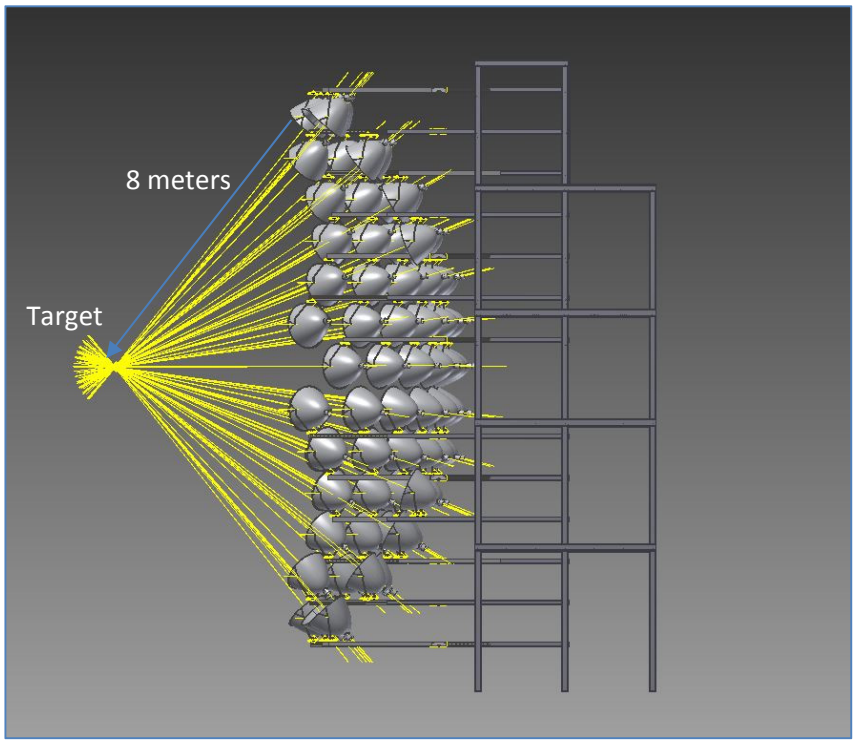


Figure 6: Modules focussed on target at 8 meters

3. Control signals

The above mentioned devices require control signals from a digital controller in order to function. These control signals are generated by dedicated addressable circuitry linked to a central controller

with a control bus. The following points summarize the digital signals required to operate/communicate with these devices.

- The stepper motors are driven by a pulse width modulated output signal with the help of a motor controller card.
- The speed of fan is controlled by an analog output signal.
- The lasers are to be controlled by a digital output which can be put to on or off state.
- The rectifiers used for the lamps have a RS232 protocol based embedded controller. There is a specific protocol supplied by the vendor for these devices, in order to communicate with the main bus.

4. Controller hardware

In this section, the description of the hardware used and configured to achieve the control system functions is presented.

4.1 Industrial PC

The main controller used is an industrial PC, an x86 PC-based computing platform for industrial applications. Industrial PCs offer different features than consumer PCs in terms of reliability, compatibility, expansion options and long-term supply. The following features were the key deciding factors in the selection of IPC [5]:

- Heavier metal construction as compared to the typical office non-rugged computer
- Enclosure form factor that includes provision for mounting into the surrounding environment (wall-panel mount with DIN rail)
- Additional cooling
- Expansion card retention and support
- Enhanced EMI filtering
- Enhanced environmental protection such as dust proofing
- Higher grade power supply
- Controlled access to the I/O through the use of access covers
- Inclusion of a watchdog timer to reset the system automatically in case of software lock-up

Along with the main IPC, expansion cards are used to provide or expand on features not offered by the motherboard. The IPC along with the logical processing performs the bus mastering over expansion cards as well. Bus mastering is the feature that enables a device connected to the bus to initiate transactions.

After a careful review of potential suppliers, a Beckhoff Automation GmbH & Co. KG supplied configuration was selected for the design. Beckhoff Automation implements open automation systems based on PC Control technology. The main controller as leased from the manufacturer is an Embedded PC which is a modular IPC available in miniature format for DIN rail mounting

Beckhoff Automation also supplied the necessary Fieldbus components for all I/Os and fieldbus couplers. All Beckhoff Automation controllers are programmed using TwinCAT in accordance IEC 61131-3 programming standard [5].

4.2 Additional hardware

The main embedded controlled and the expansion cards require additionally the following electrical components for interconnection and operation:

4.2.1 Switches

IEC 61508 standard [6] states that certain machine functions must trigger on a single human action using a manually actuated control device. Following have been considered in the design:

- Emergency stop switch (E-stop): An emergency stop switch with lock and key system would be mounted in the facility. The stop switch is to remain operational at all times and programming is done in such a way as to stop the machine without creating additional hazards.
- Power ON switch: The activation of module control system is performed through a mechanical switch. This power ON switch would be mounted on the same panel as the main emergency switch.
- Maintenance switch: The maintenance switch is to be installed inside each electrical chassis. On pressing this switch the module would rotate on the azimuth axis and turn itself from the normal working position to face directly at a platform, provided to access each module. This switch sends a digital input to the controller over bus. On the software side this signal is acknowledged and any further output to the motors is prevented until the user inputs a restore switch.

Resetting the E-stopped system would require releasing the E-Stop that was originally activated. If more than one E-Stop activated, all must be released before the modules can restart. Just resetting E-Stops would not restart the modules; this action only permits restarting through normal procedures. Additionally, RCDs (Residual-current devices) would be fitted with a trip current of no more than 30 mA, which is required by the standard on sockets rated up to 20 A

4.2.2 Electrical chassis

The entire control electronics is housed inside an electronic chassis. One such chassis per 8 modules would be provided. Each chassis is provided with an independent 24 V DC power supply. General-purpose with a rating of 12 provided by National Electrical Manufacturer's Association or International Protection Marking IP55 would be used. These chassis are intended for indoor use and provide some protection against dust, falling dirt, and dripping noncorrosive liquids.

4.2.3 Cables and jacks

The cables with corresponding connectors required to interconnect various components of the control system are described here:

- Ethernet wiring with Cat 6 cables for each module with RJ45 jacks. Further Ethernet connections are provided in the main office and experiment rooms.
- D-sub 9 pin connectors for RS 232 communication
- Insulated conductors with varied diameters are required for PLC wiring. EN 60204-1 [8] states that insulated conductors be color coded as follows:

Wire color	Type
Black	AC and DC power circuits
Red	AC control circuits
Blue	DC control circuits
Orange	External voltage source
Light blue	Neutral
Green and yellow	Protective conductor
White	Analog input
Violet	Analog output
Brown	Digital input
Grey	Digital output

Table 3: Wire color coding

4.2.4 Power supply

A switched-mode power supply (SMPS) would be provided in each chassis, in which the AC mains input is directly rectified and then filtered to obtain a stable DC voltage. The resulting DC voltage is then switched on and off at a high frequency by electronic switching circuitry, thereby enabling the use of transformers and filter capacitors that are much smaller, lighter, and less expensive than those found in linear power supplies operating at mains frequency. Following safety features were kept in mind during power supply selection:

- The output will be electrically isolated from the mains
- SMPS are regulated, and to keep the output voltage constant, the power supply employs a feedback controller that monitors current drawn by the load.
- Includes safety features such as current limiting or a crowbar circuit to help protect the device and the user from harm. In the event that an abnormal high-current power draw is detected, the switched-mode supply can assume this is a direct short and will shut itself down before damage is done.
- Provide a power good signal to the motherboard; the absence of this signal prevents operation when abnormal supply voltages are present.

4.3 Auxiliary control systems

Apart from the main control system for the modules, additional separate systems would be provided in the plant by different vendors [2]:

- IP-cameras for visualization at different positions
- HVAC (Heating ventilation and air conditioning) system with a vendor-specific protocol
- Access management system with vendor-specific bus protocol

These systems or any part of these systems are not to be operated upon by the main control system. The module control system is designed to receive inputs from these control system. These inputs determine the safety and readiness of the plant.

5. Software design

The software is written considering the prototype available at the DLR lab and is scalable to implement it on the complete facility in future. Following are the key aspects of software design undertaken:

5.1 IEC 61131-3 programming

IEC 61131-3 is an open international standard and a part of general IEC 61131 for programmable logic controllers. For the current project STL (Structured Text Language) mentioned in this standard is chosen as the primary language for coding [9]. It is supplier independent and highly portable and reusable coding can be done with it.

The main program organization units in IEC 61131-3 are:

- **Functions**: User written or standard mathematical operators.
- **Function Blocks:** Libraries of functions, supplied by a vendor or third party.

TwinCAT® IDE provided by Beckhoff Automation GmbH & Co. KG, runs under the Windows® operating systems and includes both the programming environment in STL language and the run-time system. It creates a pure software PLC (Programmable logical controller) and allows up to four virtual "PLC CPUs", each running up to four user tasks, on one industrial PC. Many fieldbus cards from various manufacturers are supported in this integrated development environment. It is possible to operate multiple fieldbus cards per IPC. The TwinCAT architecture allows the use of C and C++ as the programming language as well [5].

5.2 Safety integrity specifications

During a set of meetings a structured and systematic examination of the proposed plant was carried out in order to identify and evaluate problems that may represent risks to personnel or equipment, or prevent efficient operation. To avoid adverse Safety, Health and Environmental (SH&E) consequences, all critical processes in the system were identified which on occurrence of any operational problem, may need to be put into a safe state. The safe state has to be achieved in a timely manner and within the process safety time.

The IEC/EN 62061 (Safety of machinery) standard was referred for this. It provides guidance to implement functional safety of electrical, electronic and programmable electronic control systems. It also provides requirements that are applicable to the system level design of all types of machinery safety-related electrical control systems and also for the design of non-complex subsystems or devices.

The risk assessment concluded in a risk reduction strategy following which the safety-related control specifications are identified. These specifications are described in the following sections:

5.2.1 Safety functions

The specific control functions performed to put the system in safe state are from here on referred to as Safety Functions (SF). They are implemented as part of an overall risk reduction strategy which is intended to eliminate the likelihood of a, previously identified, SH&E event. This involves collecting and analyzing information regarding the parts, mechanisms and functions of the prototype. It is necessary to consider all the types of human task interaction with the modules and the environment in which the modules will operate.

To create appropriate safety functions, cause and effect diagrams were made [10]. The methodology can be referred from 'Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety'. The following diagram for example identifies possible causes of human injury during operation. The programming of the software is done in order to prevent or reduce the probability of occurrence of these events.

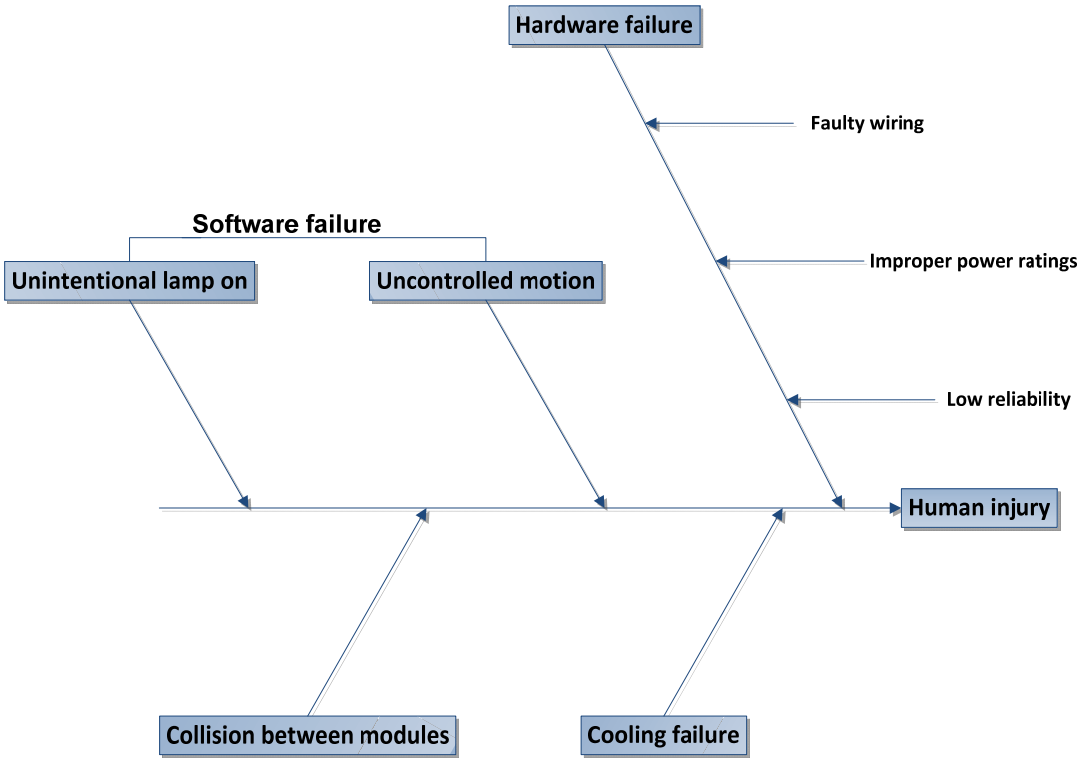


Figure 7: Cause and effect diagram

The hardware-side failures such as faulty wiring, improper power ratings etc. are human errors which have to be prevented during installation process. The factor of reliability has been taken due care in selecting hardware configuration.

The software-side failure modes have been described in the following sections with the help of Fault Tree Analysis diagrams. These diagrams help with understanding the overall system. The full knowledge of the system is very important for not missing any cause initiating an undesired event. For the undesired events, as shown in above figure all causes were sequenced in the order of possible occurrence and then used in constructing the fault tree diagrams.

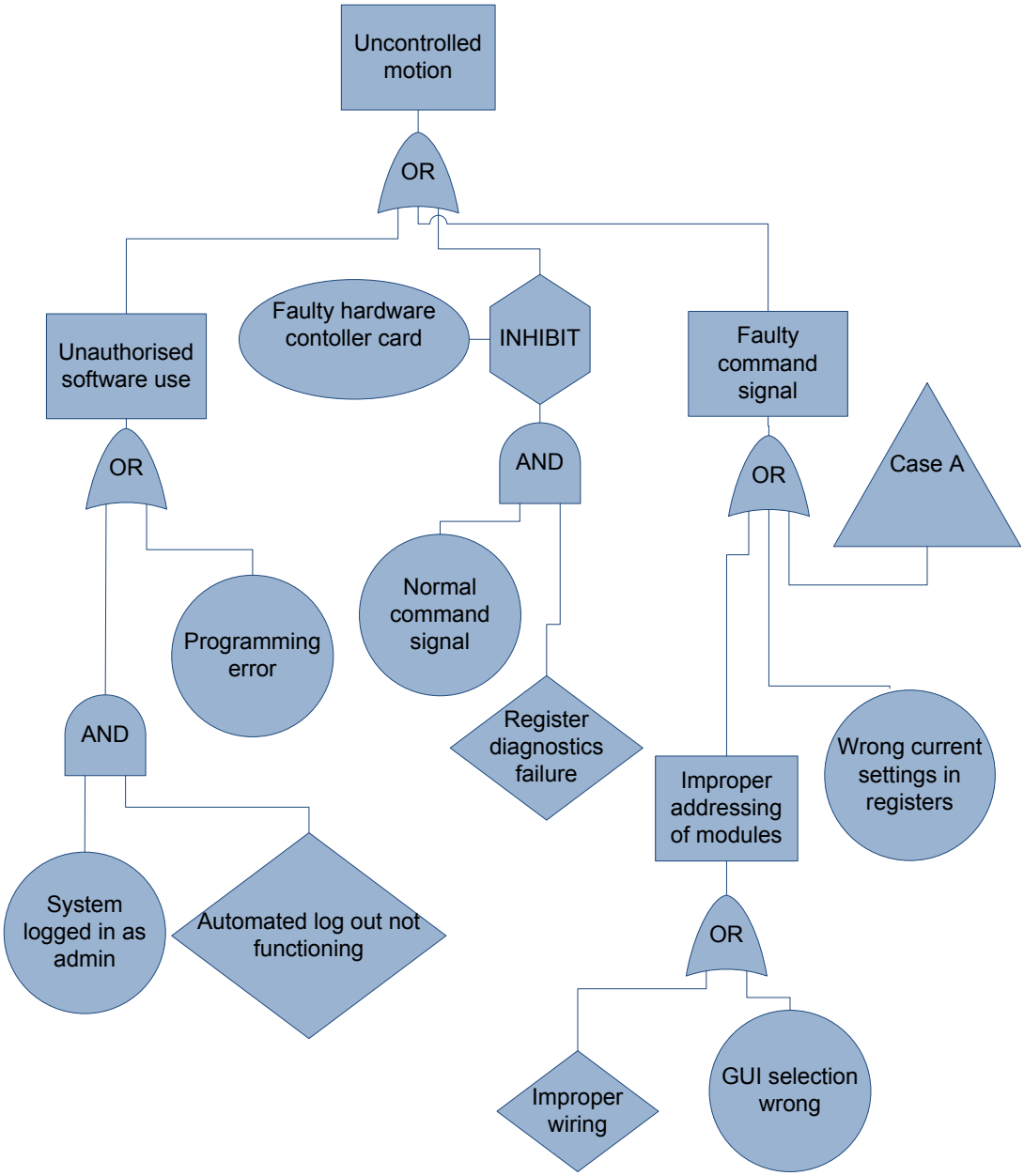


Figure 8: Fault tree analysis (Uncontrolled motion-I)

The uncontrolled motion could result in permanent partial disability or injuries that may result in hospitalization or financial loss due to equipment damage. To prevent this, the above fault tree analysis identifies possible failure modes. The principal modes of failure are an unauthorized software use or a faulty command signal to the motors.

These modes are prevented by careful programming of the software. Further safety measures include that motors can be moved only a set sequence which cannot be changed. The module can be brought back to zero position on the issue return command. Most limits and all stages of life-cycle including installation, commissioning, maintenance, decommissioning, correct use and operation as well as the consequences of reasonably foreseeable misuse or malfunction were considered in this phase.

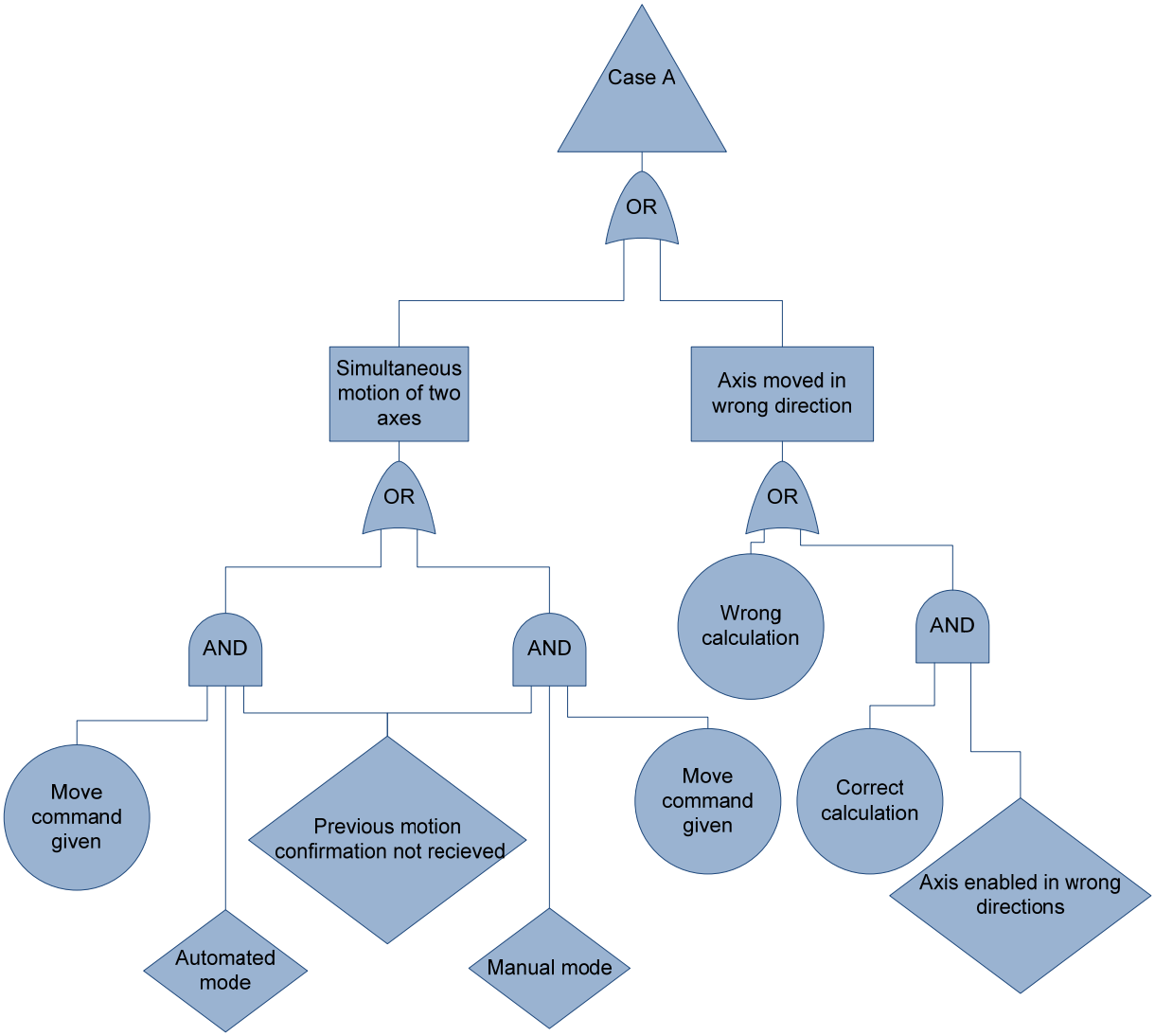


Figure 9: Fault tree analysis (Uncontrolled motion-II)

The above figures show a possible generation of faulty signal to motor controller on account of:

- Simultaneous motion command given to two different axes
- Faulty calculations or enabling a motor in wrong direction
- Wrong settings in the control card registers

As a risk assessment is an iterative process, therefore it shall be performed at different stages of the machine life cycle. The information available will vary in the latter stages of the life cycle. A risk assessment conducted by the end user would have access to every detail of the modules working environment [11]. Ideally the output of one such analysis will be the input for the next iteration.

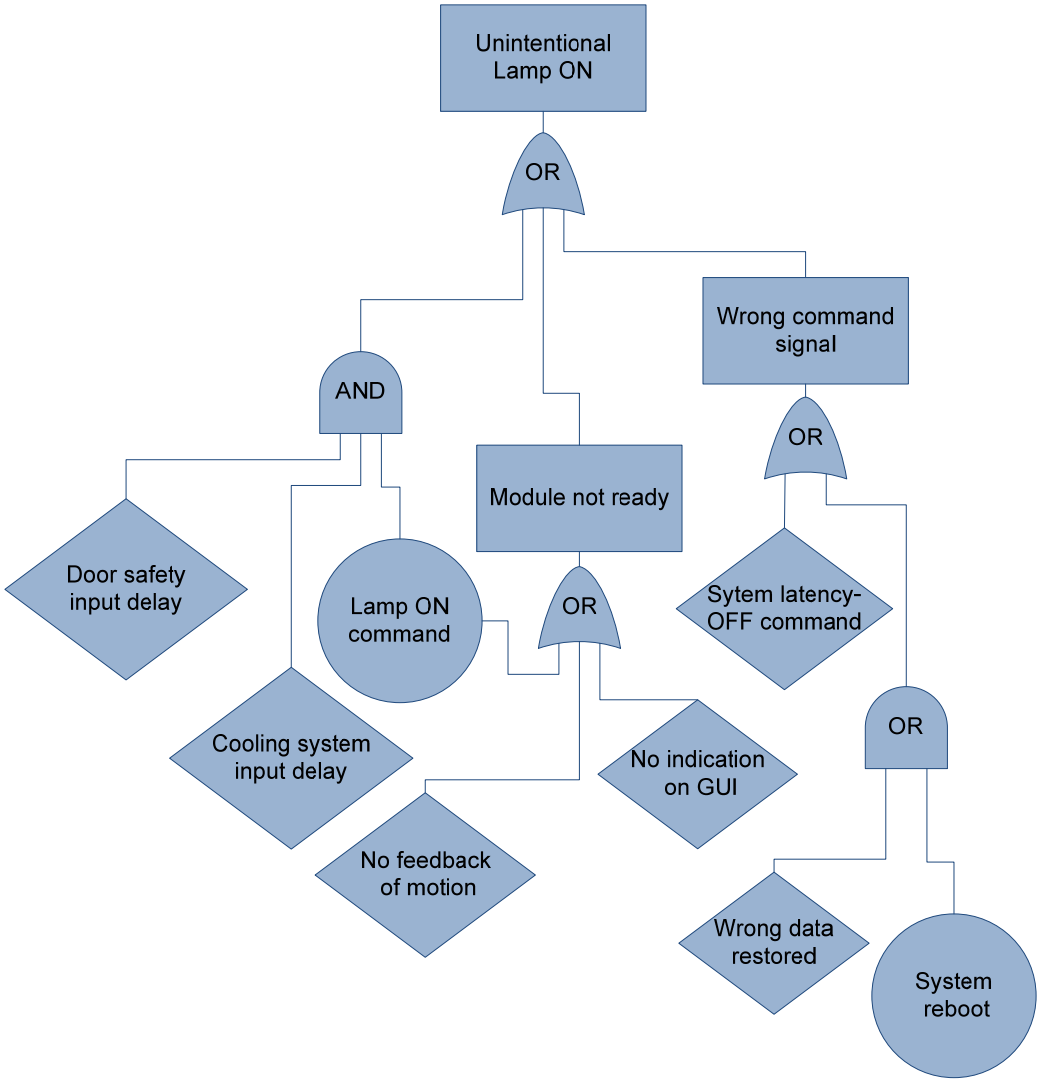


Figure 10: Fault tree analysis (Unintentional lamp on)

An unintentional lamp powered ON could range from minor equipment damage up to an event involving an uncontrolled catastrophic release of energy causing human injury. Following failure modes demonstrate that:

- A lamp powered ON automatically on a system reboot
- High system latency leading to wrong command signal
- A lamp powered ON before module being positioned completely

These situations have been prevented through programming and moreover the standard users can control only the lamps for which they have given access by the administrator. With the pressing of emergency stop button on the GUI screen, switch off command to all the lamps is given. The lamps are however switched off one after another with a delay of 10 seconds in order to avoid high current surges in the circuit.

The standard user is given the possibility to read the parameters like current, voltage of rectifiers on the GUI screen. They are however not given the rights to modify these parameters. The parameters can be changed using the dropdown list provided on the GUI.

Administrator and maintenance user can operate each lamp individually after the activation of corresponding module. Administrator can make additional adjustments to the rectifiers. This can be realized from a drop down menu.

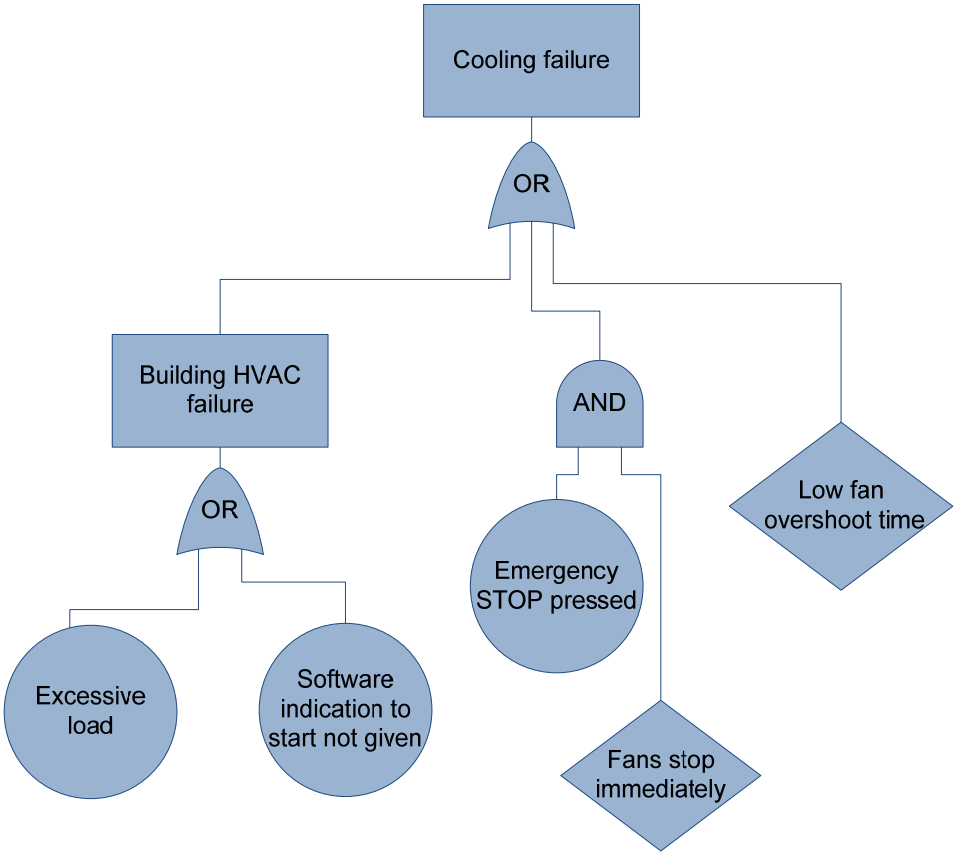


Figure 11: Fault tree analysis (Cooling failure)

Cooling failure as demonstrated in the figure refers thus to a condition, when during the process of operation or shutdown, building HVAC or the modules fan fail to operate. This may cause high temperatures, leading to equipment loss as well as adversely affecting the operation of lamps.

5.2.2 Safety integrity level (SIL)

SIL is a measurement of performance required for a safety function. A SIL is determined based on a number of quantitative factors in combination with qualitative factors such as development process and safety life cycle management.

In the European functional safety standards based on the IEC 61508[6] standard four SILs are defined, with SIL 4 the most dependable and SIL 1 the least. Electric and electronic devices are certified for use in Functional Safety applications according to IEC 61508, providing application developers the evidence required to demonstrate that the application including the device is also compliant. For the current project SIL 3 level is deemed sufficient for the required performance and the hardware has been selected from a TÜV certified manufacturer with SIL 3 level of safety.

PFD (probability of failure on demand) and RRF (risk reduction factor) of low demand operation for different SILs as defined in IEC EN 61508 are as follows:

SIL	PFD	PFD (power)	RRF
1	0.1–0.01	$10^{-1} – 10^{-2}$	10–100
2	0.01–0.001	$10^{-2} – 10^{-3}$	100–1000
3	0.001–0.0001	$10^{-3} – 10^{-4}$	1000-10,000
4	0.0001–0.00001	$10^{-4} – 10^{-5}$	10,000–100,000

Table 4: Safety integrity levels classification [6]

Thus the control system hardware is ensured to have a failure probability of less 10^{-3}.

6. Function description

Following documentation describes the requested behavior of the software with GUI and sections of code applicable. For the complete coding of software, please refer to **Appendix-I**.

6.1 Primary functions

The primary function refers to all the available functions of the software that are accessible to the user. These functions refers to sending specific signals to expansion cards connected to the main controller spread across the facility, in order to operate the facility in desired manner. These functions also include processing input from various temperature sensors, status from auxiliary control systems.

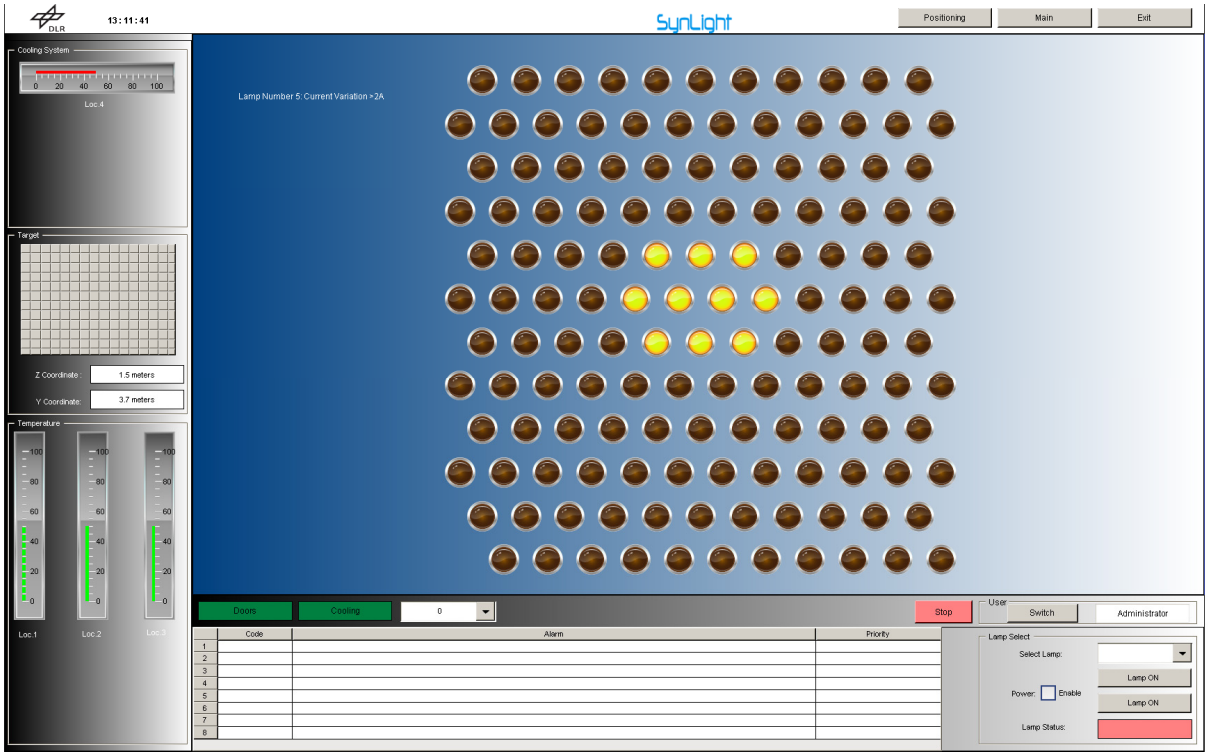


Figure 12: GUI-Home screen

These functions are carried out on one module at a time. The module is to be selected on the interface provided to the user. The selection feature is provided to the following users: Admin and User. The following figure shows a graphical control panel to power lasers/fans on or off:

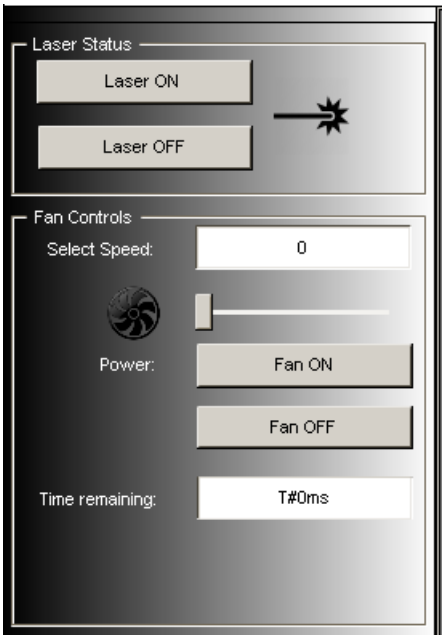


Figure 13: Fan and Laser controls

As discussed in the previous sections, there are 149 Xenon lamps to be mounted in the facility which are powered through dedicated rectifiers. These lamps can also be accessed on the GUI from the control panel as shown in the figure below. The user can select a lamp from the drop down menu, enable the lamp for operation and power it on and off as required. The feedback from the lamp can also be visualized.

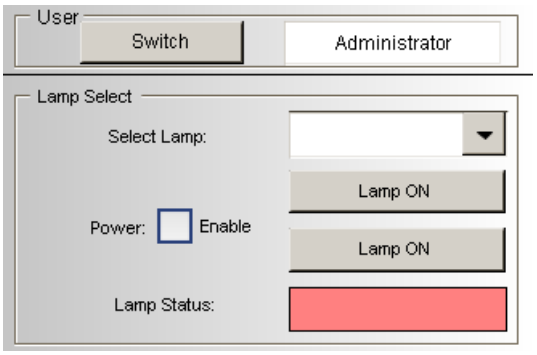


Figure 14: Lamp controls

6.1.1 Screening of username and experiment mode

In this section the process of screening and the coding for it is explained. The user is prompted on the start of the software to enter the username and password. These parameters are then compared to the stored string values inside the 'Main' program routine. For every matching case, the function 'Screening' is called with a parameter 'OperationMode'. The parameter 'OperationMode' is an instance of enumerated data type 'Mode' which is a list of all possible operation modes.

Inside the function 'Screening', each case of value 'OperationMode' is evaluated and the user rights are set accordingly. The output of each such evaluation is visualization buttons being set to activated or deactivated state.

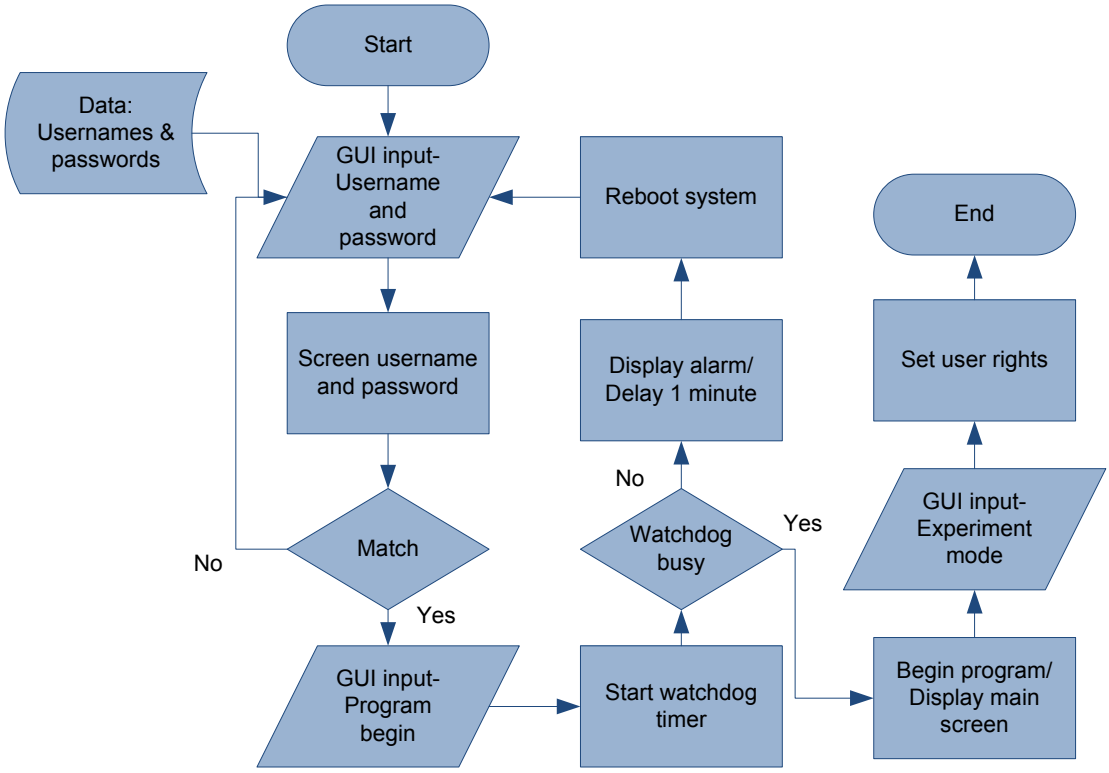


Figure 15: Flowchart-Screening function

The username is displayed on the GUI at all times, after a successful login. This can also be changed by using the 'Switch User' button provided on one of the control panels on home screen.

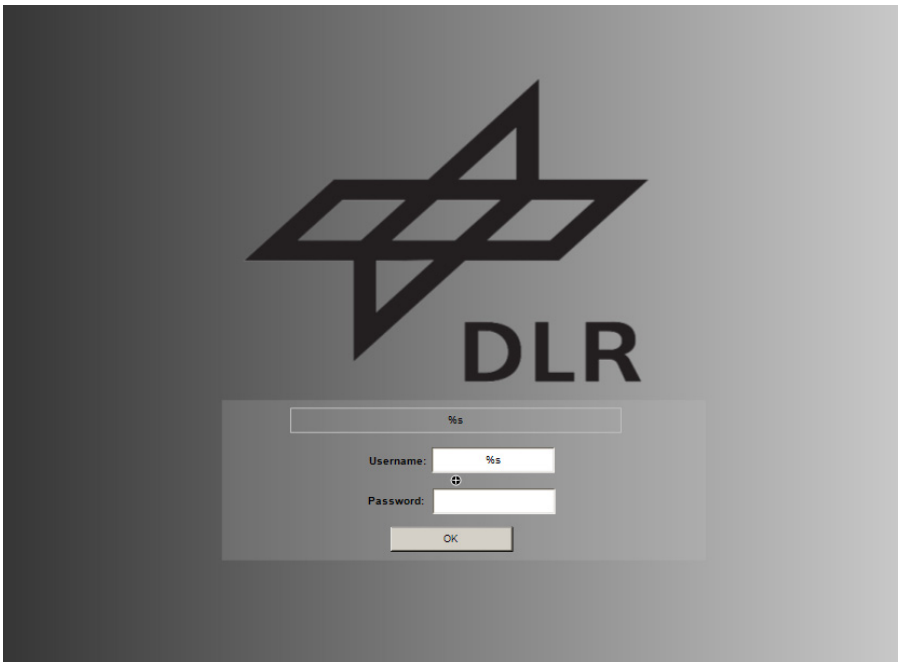


Figure 16: GUI- Username and password

The following code demonstrates how the above mentioned process is programmed in the software:

```
4
5       prog ( ) ;
6       /// Screening Begin///
7
8       header := 'Please enter your credentials...' ;
9
10      IF  username = 'dlr'  AND  password = 'admin'  THEN
11
12          header := 'Credentials verified' ;
13
14          operationmode := Admin ;
15
16          visuglob . UserID := 'Administrator' ;
17
18      END_IF
19
20      IF  username = 'dlr'  AND  password = 'user'  THEN
21
22          header := 'Credentials verified' ;
23
24          operationmode := User ;
25
26          visuglob . UserID := 'User' ;
27
28          //visuglob.okbutton2:=TRUE;
29
30      END_IF
31
32      Screening ( operationmode ) ;
33
34      ///Screening End///
```

On validating the credentials, the user is asked for confirmation to start the program. Upon confirmation the program starts the system watchdog timer. On True value (normal state of the controller hardware) of this watchdog timer the system passes on to 'ProgramBegin' state.

```
1       watchdogtime (
2       tTimeout := T#1S ,
3       bEnable := TRUE ,
4       benabled => programbegin
5       ) ;
6
```

On False value (hardware failure) of the watchdog timer the system display an alarm to the programmer. Further after a delay of 1 minute the system is programmed to reboot itself. The programmer has the ability to prevent the system from rebooting by forcing the watchdog timer output to True.

```
1       rebootcode (
2       netid := '5.33.178.110.1.1' ,
3       delay := 1 ,
4       start := TRUE ,
5       Tmout := T#3S
6       ) ;
7
```

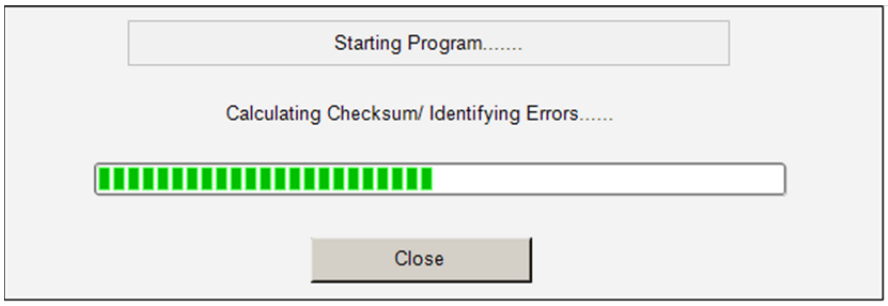


Figure 17: GUI- Program initializing and register read/write

The above GUI shows a status bar being displayed to the user during the time hardware is initialized (registers are read/written) or watchdog timer is checked.

6.1.2 Motion program

The feedback over the motor position is done visually with the help of cameras, and thus no encoder or reference switch is used as such. The position of the module is controlled on three axes. Each of these axes has an individual calibration factor based on the mechanical configuration. The axes are moved in a specific order to prevent the collision between individual modules. The order of preference for motion along these axes is: Axial- Elevation- Azimuth. When the module is to be brought at the starting position the pattern of motion remains the same.

A function interface 'ITF1' to provide the motion functionalities of the module control system is created in the project. To move any axis, a control flag is generated and the control register is updated. Following which a move command is passed to the following data structure created.

```
STRUCT
        ControlDWord            : DWORD; (Control double word)
        Override                : DWORD; (Velocity override)
        AxisModeRequest         : DWORD;
        AxisModeDWord           : DWORD;
        AxisModeLReal           : LREAL; (optional mode parameter)
        PositionCorrection      : LREAL;
        ExtSetPos               : LREAL; (external position setpoint)
        ExtSetVelo              : LREAL; (external velocity setpoint)
        ExtSetAcc               : LREAL; (external acceleration setpoint)
        ExtSetDirection         : DINT;  (external direction setpoint)
        Reserved1               : DWORD;
        ExtControllerOutput     : LREAL; (external controller output)
        GearRatio1              : LREAL;
        MapState                : BYTE;
        Reserved_HIDDEN         : ARRAY [105..127] OF BYTE;
END_STRUCT
```

The data structure that is cyclically exchanged between PLC and the motor controller cards. This data structure is placed in the output process image of the PLC and linked in TwinCAT System Manager. An ADS (Automation Device Specifiation) data structure containing the ADS communication parameters for an axis is then used for bus communication. The data is then passed to the motor controller. The coding is provided in Appendix-I, Section-2.

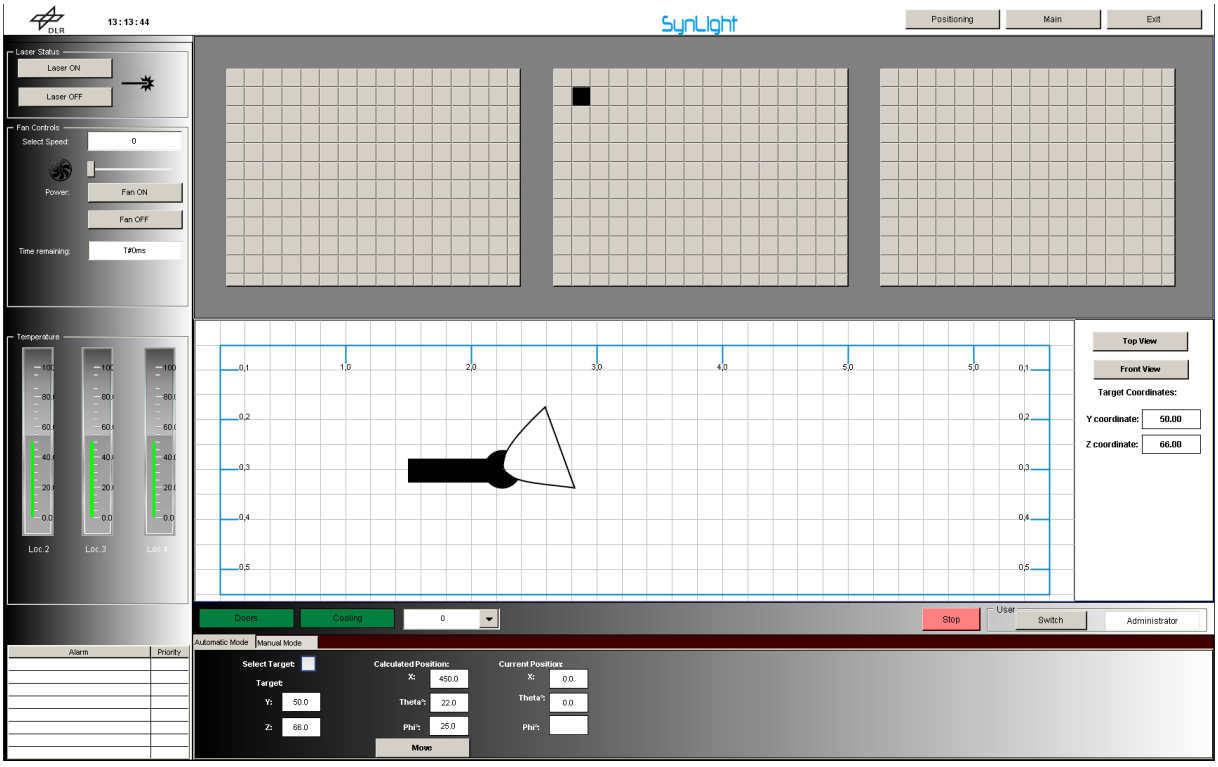


Figure 18: GUI-Motion program

6.1.3 Register read/write

The following data unit is created in the program to communicate with the registers of KL2541 motor controller cards that are provided by Beckhoff [5]:

```
7       RegError : ARRAY [ 1 .. 50 ] OF BOOL ;
8
9       RegErrorID : ARRAY [ 1 .. 50 ] OF UDINT ;
10
11      TrmTyp :  ARRAY [ 1 .. 50 ] OF UINT ;
12
13      OutRegNmb : ARRAY [ 1 .. 50 ] OF USINT ;
14
15      OutRegVal : ARRAY [ 1 .. 50 ] OF UINT ;
16
17      InRegNmb : ARRAY [ 1 .. 50 ] OF USINT ;
18
19      InRegVal : ARRAY [ 1 .. 50 ] OF UINT ;
20
21      ReadR : ARRAY [ 1 .. 50 ] OF BOOL ;
22
23      WriteR : ARRAY [ 1 .. 50 ] OF BOOL ;
24
25      ReadWriteRegBusy : BOOL ;
```

This data unit is also required at the start of the module control program to set the motor controllers to receive state and also to set current values as per the motor requirements which are currently 2 Amperes per motor.

An example of register communication for reading the firmware issue status from register 9 of a terminal is presented here:

Output Data:

- Bit 0.7 set indicates register communication active.
- Bit 0.6 not set indicates reading the register.
- Bit 0.5 to Bit 0.0 indicates with 00 1001_{bin} the register number 9.
- The output data word (Byte 1 and Byte 2) has no function at the reading access. When the register value is to be changed it is written into the output data word.

Byte 0: Control Byte	Byte 1: DataOUT1, high byte	Byte 2: DataOUT1, low byte
0x89 (1000 1001_{bin})	0xXX	0xXX

Table 5: Outbound register communication

Input Data (answer of the bus terminal):

Byte 0: Status Byte	Byte 1: DataIN1, high byte	Byte 2: DataIN1, low byte
0x89	0x33	0x41

Table 6: Inbound register data

Explanation:

- The terminal returns the value of the Control Byte in the Status Byte, as an acknowledgement.
- The terminal returns the Firmware Issue Status 0x3341 in ASCII code, in the input data word (Byte 1 and Byte 2). This has to be interpreted as ASCII code. ASCII code 0x33 stands for the cipher 3 and ASCII code 0x41 stands for the letter A. Therefore the firmware version is 3A.

The motion functionalities are split in two modes:

Automatic mode: In automatic mode, the user is prompted to select the module and a point on target wall shown on the GUI. Corresponding to selected point, the output from set point generation algorithm is shown to the user as explained in section 6.1.5.

On further pressing the 'Move' button on GUI, the collision detection algorithm is invoked. If there is a possibility of collision the user is notified and motion is not performed. If not, the motion program is passed on the values calculated from set point generation algorithm and the motion is actuated.

The user is able to read the current position of X, Theta and Phi axis on the GUI.

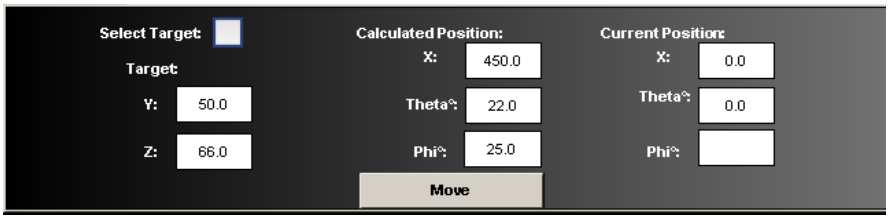


Figure 19: Automatic motion controls

Manual mode: In the manual mode, the user is prompted to enter the desired position of each axis in mm and degrees. The collision detection algorithm is activated is this mode as well. On pressing the move button the axis is moved to desired position.

The user can set the speed of individual axis for the current module or get the global settings for each axis.

Figure 20: Manual motion controls

6.1.4 Collision detection

The collision detection function prevents two modules from colliding into one another. This is done by storing the position of each module into a data structure. When a module is selected and given a move command, the motion direction is evaluated based on the given user input or from the calculated position in case set point generation algorithm is used.

The position value of nearest neighboring module in that direction is retrieved then and a limiting angle θ_L, as shown in the figure is calculated.

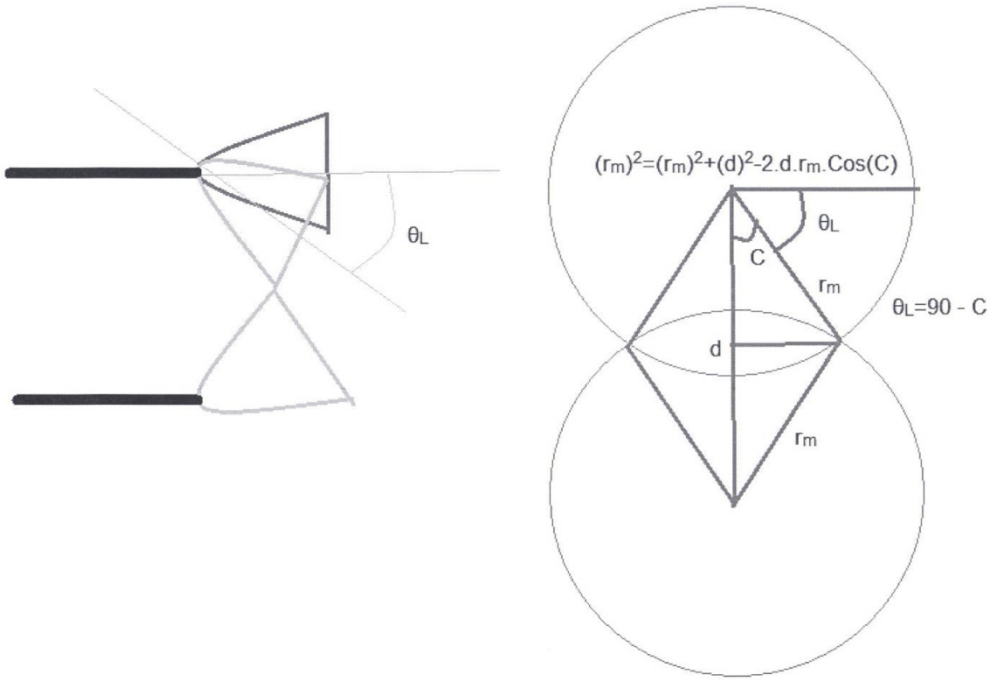


Figure 21: Calculation of limiting angle θ_L

To calculate the θ_L, the distance between the centers of adjacent modules in that particular direction is required. This distance depends on the final mechanical configuration of the plant. The value r_m represents the radius of rotation or the span of each module. This value is different for horizontal and vertical directions.

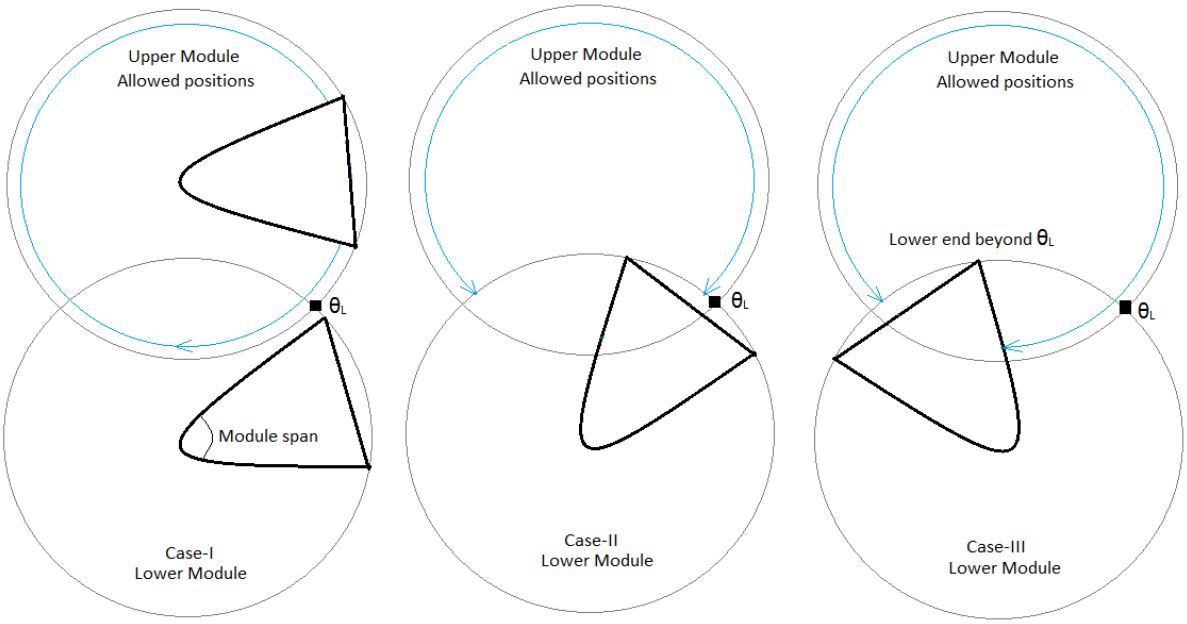


Figure 22: Allowed motion for different configurations

From the θ_L value so obtained a safety threshold is calculated. The safety threshold value depends on the position of adjacent modules. Let us consider two modules vertically adjacent to each other and the safety threshold is calculated as follows:

For the case when lower module is below the θ_L as shown in the figure below. The upper module to be moved can take any value calculated or input by the user without any possible collision.

For the case when lower module is above the θ_L but the other end of this module has not completely crossed over the θ_L limit. The upper module can only move until the point θ_L and any further movement will cause a collision. Therefore the user is notified on the GUI about possible collision.

In the case when the lower end of the lower module has crossed the θ_L value, the upper module can take a position further than θ_L but at a safe distance from the lower end of the lower module. The flowchart for the algorithm is shown below:

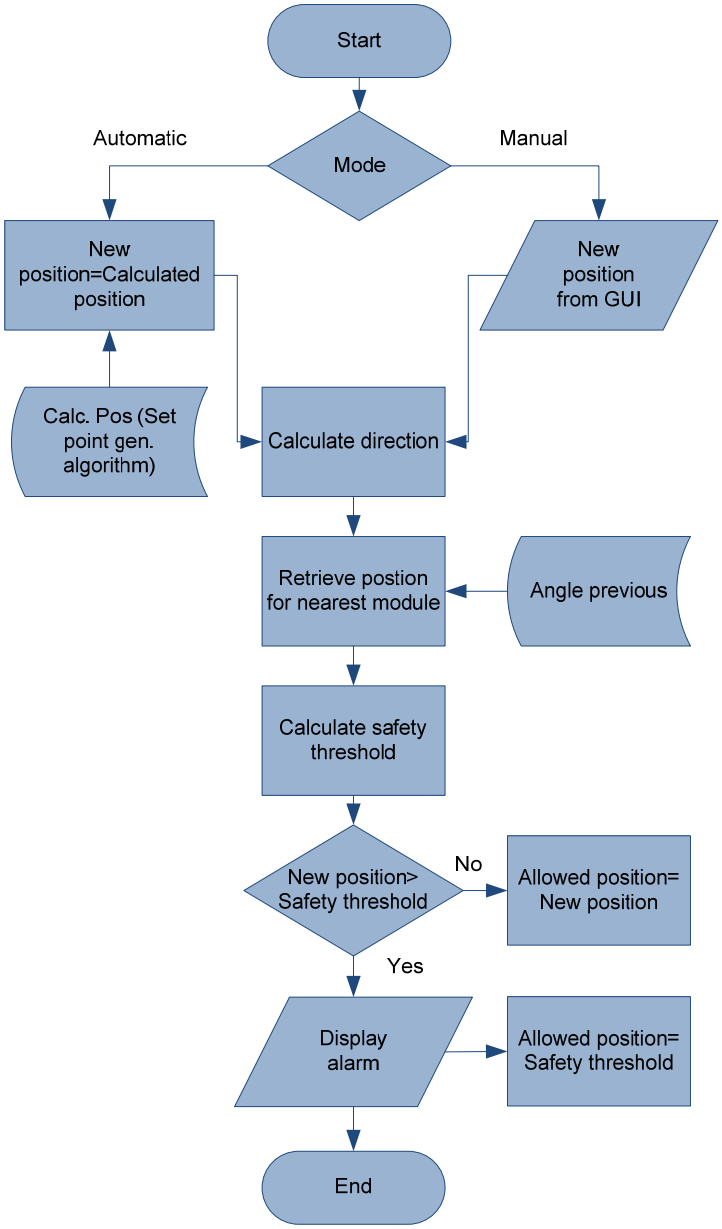


Figure 23: Flowchart-Collision prevention

When the program detects a collision possibility, an alarm is displayed on the GUI as shown in the figure below. The user can switch view to locate collision possibilities on Theta as well as Phi axis.

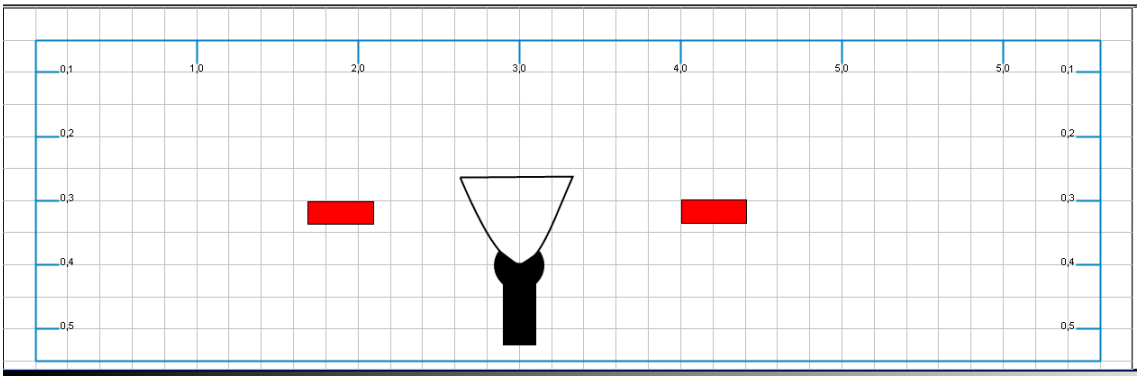


Figure 24: Collision possibility alarm

The coding for the algorithm is presented here:

```
PROGRAM prog

VAR

    alpha : INT ;

    beta : INT ;

    limiter : INT ;

    limiter2 : INT ;

    ycalc : INT ;

    moduleswap2 : int ;

    moduleswap : INT ;

    x : INT ;

    theta : INT ;

    gamma : INT ;

    xcalc : INT ;

    z : INT ;

    y : INT ;

    one : BOOL ;

    two : BOOL ;

    three : BOOL ;

    trig1 : r_trig ;

END_VAR
```

```
trig1 ( clk := one , q => two ) ;

IF trig1 . q = TRUE

    THEN

//Moving down

IF xcalc > 0

    THEN

// downward facing bottom module

    IF

        alpha > limiter

        THEN

        x := xcalc ;

    END_IF ;

//upward approaching bottom module
```

```
27      IF  alpha <=  limiter
28
29          THEN
30
31          IF  alpha <  limiter + moduleswap
32
33              THEN  x :=  xcalc + ( alpha - moduleswap - limiter ) * - 1 ;
34
35          ELSE
36
37          x :=  - 1 * limiter ;
38
39          END_IF
40
41      END_IF
42
43      // for upward motion of this module
44      ELSE
45
46          limiter2 := - 1 * limiter ;
47
48          moduleswap2 := - 1 * moduleswap ;
49
50      // upward facing upper module
51
52      IF  beta < limiter2
53
54          THEN
55
56          x := xcalc ;
57
58      END_IF
59      // downward approaching upper module
60
61          IF  beta >= limiter2
62
63              THEN
64
65              IF  beta >  limiter2 + moduleswap2
66
67              THEN  x :=  xcalc + ( beta - moduleswap2 - limiter2 ) * - 1 ;
68
69              ELSE
70
71              x :=  - 1 * limiter2 ;
72
73          END_IF
74
75          END_IF
76
77      END_IF ;
78
79      ////////////////////// Azimuthal angle/////////////////////////////////////////
80
81      //moving Right clockwise
82
83      IF  ycalc > 0
84
85          THEN
86      // right facing right module
87
88          IF
89
90              theta > limiter
91
92              THEN
93
```

```
94              y := ycalc ;
95
96          END_IF ;
97      //left approaching  right module
98
99      IF  theta <=  limiter
100
101         THEN
102
103         IF  theta <  limiter + moduleswap
104
105             THEN  y :=  ycalc + ( theta - moduleswap - limiter ) * - 1 ;
106
107         ELSE
108
109         y :=  - 1 * limiter ;
110
111         END_IF
112
113     END_IF
114
115     // for left motion of this module
116
117     ELSE
118
119         limiter2 := - 1 * limiter ;
120
121         moduleswap2 := - 1 * moduleswap ;
122
123     // left facing left module
124
125         IF  gamma < limiter2
126
127             THEN
128
129             y := ycalc ;
130
131         END_IF
132
133     // right approaching left module
134
135         IF  gamma >= limiter2
136
137             THEN
138
139             IF  gamma >  limiter2 + moduleswap2
140
141             THEN  y :=  ycalc + ( gamma - moduleswap2 - limiter2 ) * - 1 ;
142
143             ELSE
144
145             y :=  - 1 * limiter2 ;
146
147         END_IF
148
149         END_IF
150
151     END_IF ;
152
153     END_IF
154
```

6.1.5 Algorithm for set point generation

The target consists of three chambers of dimension 4m x 4m each. In the visualization, each of these chambers is represented by an array of selectable buttons. Each button has certain stored value

which corresponds to a particular position on target area. To calculate these values, following code is used:

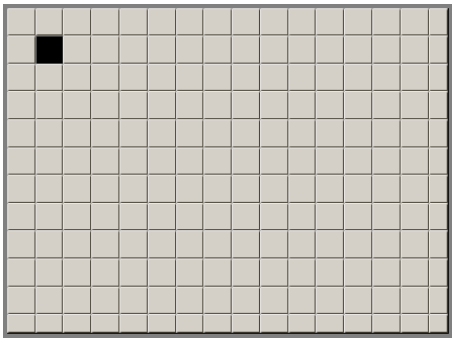


Figure 25: GUI-Target wall

```
133     ///coordinates begin///
134     FOR  c1 := 1  TO  16  BY  1  DO
135         FOR  c2 := 1  TO  12  BY  1  DO
136             targetarray [ c1 , c2 ] . CY := c1 * 25 ;
137             targetarray [ c1 , c2 ] . CZ := c2 * 33 ;
138         END_FOR
```

Each button thus represents an area of 25x33 centimeters. Higher level of precision in targeting can be achieved by implementing higher number of selectable button on the GUI.

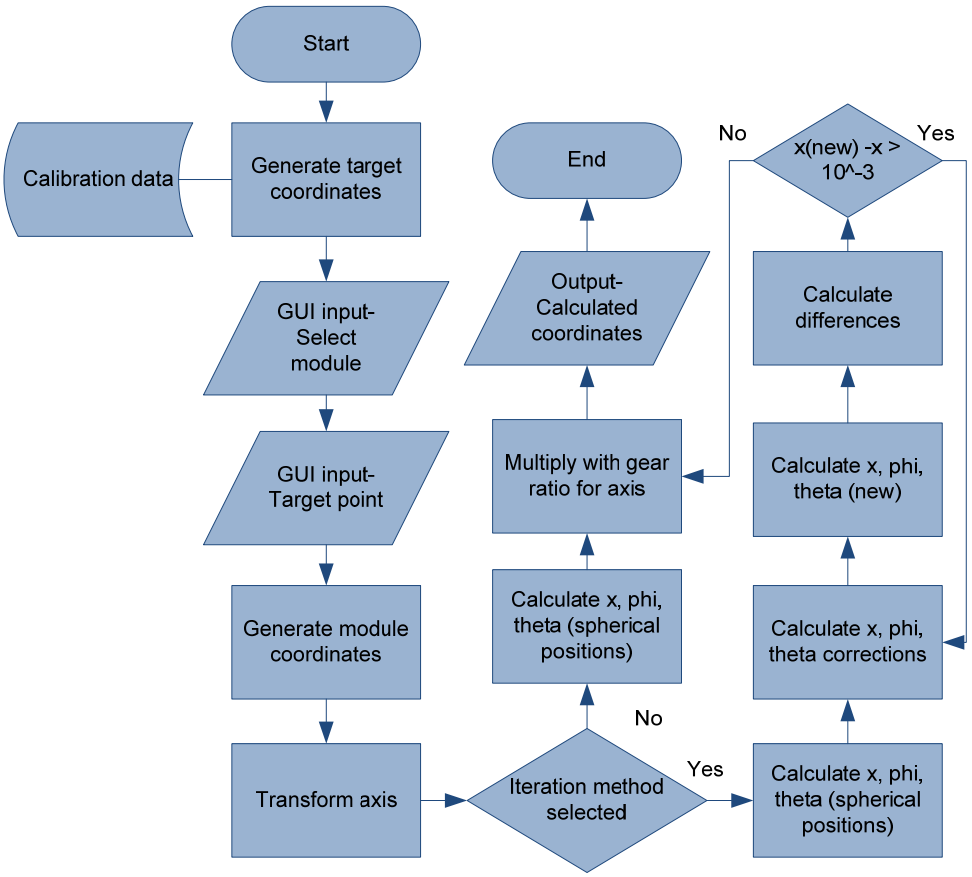


Figure 26: Set point generation algorithm

Following the selection of target point and the module to be moved, the algorithm calculates the relative coordinates of the module w.r.t the target point.

When the coordinates of target and the module are at hand, the axes are transformed into modules frame of reference and the calculation of phi and theta is done using the following set of formulae based on trigonometric evaluations:

$$\varphi = \cos^{-1}\left(\frac{z_{transformed}}{rad}\right)$$

$$\theta = \cos^{-1}\left(\frac{x_{transformed}}{rad * \sin\varphi}\right)$$

where, φ= azimuth angle

θ= elevation angle

rad= 8 meters (optimum distance of module)

$x_{transformed}$= Module position in Z axis (transformed coordinates)

$z_{transformed}$= Module position in Y axis (transformed coordinates)

The value of x, phi and theta are then to be multiplied with appropriate gear ratio of corresponding axis. The gear ratio depends on the final mechanical design.

Iterative algorithm: The point of interest for optimum positioning is the point light source created within the reflector.

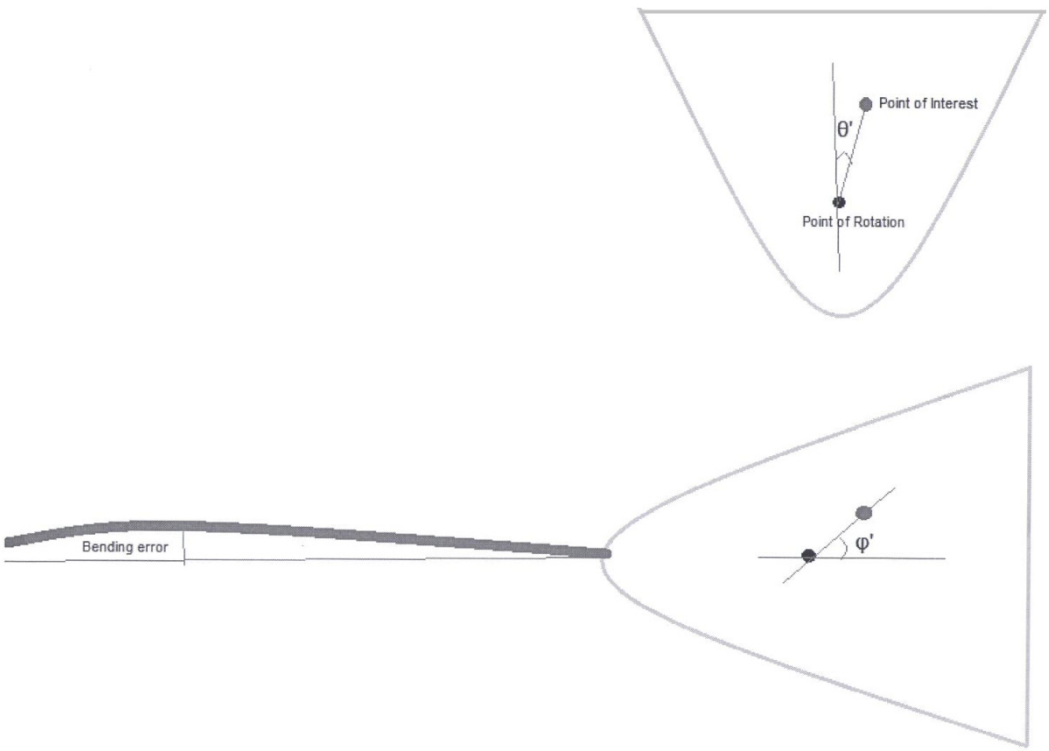


Figure 27: Offsets due to assembly

This point of interest in ideal case should be at the center of rotation for both vertical as well as horizontal directions. But in practice, there would be some angular as well as linear offsets θ' and ϕ' as shown in the figure. These offsets arise due to design, assembly or bending under dynamic loads.

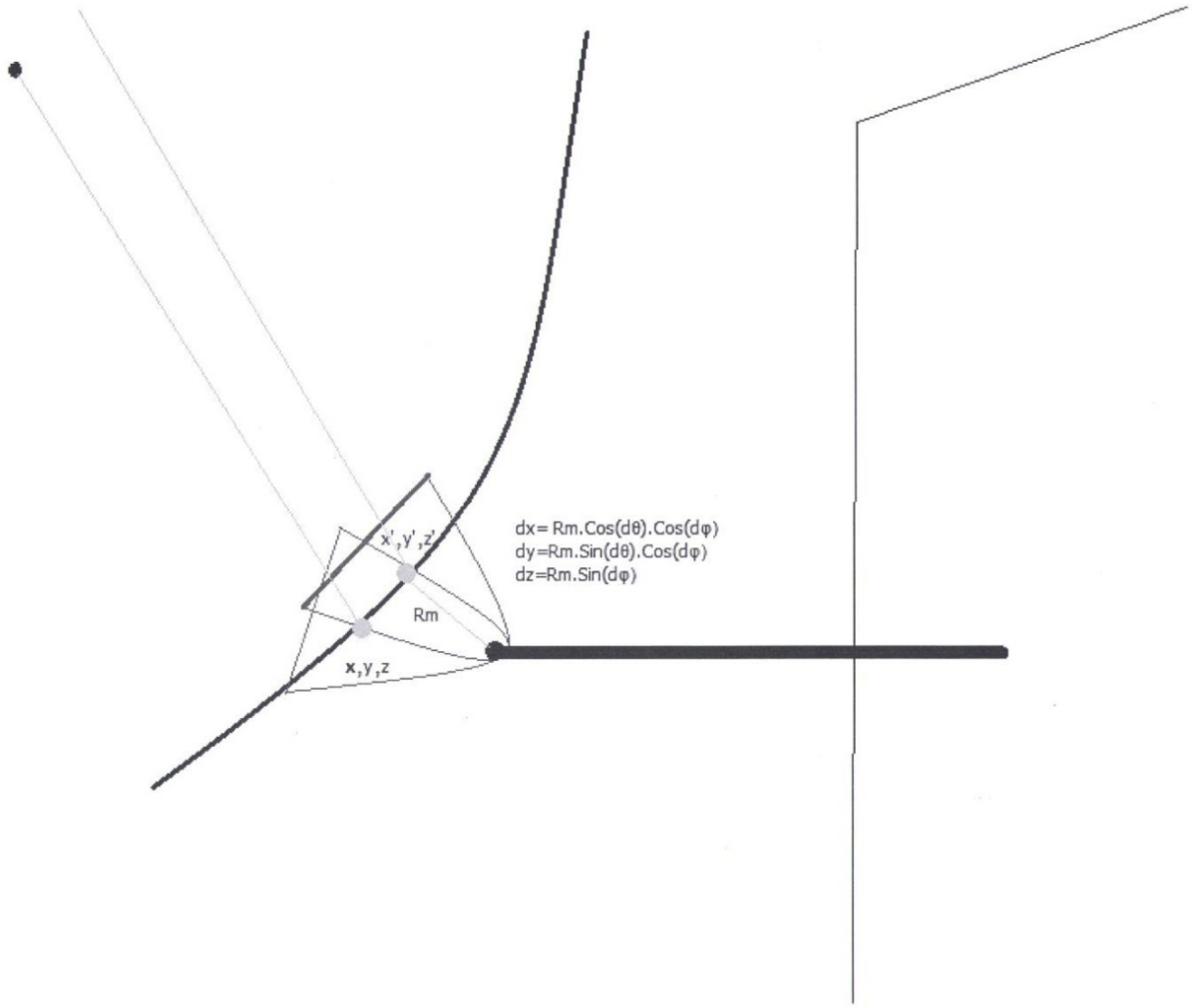


Figure 28: Effect of offsets on positioning

The effect of these offsets on the actual position of the point of interest is represented with the following formulae:

$$dx = R_m \cos d\theta' \cos d\varphi'$$

$$dy = R_m \sin d\theta' \cos d\varphi'$$

$$dz = R_m \sin d\varphi'$$

where, dx, dy, dz= change in coordinates of the module

R_m= Distance between point of rotation and point of interest

Thus the value of θ' and φ' has to be calculated over steps. And the effect on the actual position of the point of interest is to be reduced in each subsequent step.

```
PROGRAM prog_1_2

VAR_INPUT

    x : LREAL ;

    y : LREAL ;

END_VAR

VAR

    thetadash : REAL ;

    phidash : REAL ;

    xnew : REAL ;

    znew : REAL ;

    ynew : REAL ;

    z : LREAL ;

    step2 : BOOL ;

    initiator2 : BOOL ;

    Rad : REAL := 800 ;

    inter1 : REAL ;

    trig : r_trig ;

    thetadash_m : REAL ;

    phidash_M : REAL ;

    test_1 : REAL ;

    xcorrec : REAL ;

    ycorrec : REAL ;

    zcorrec : REAL ;

    trig2 : r_trig ;

    i : INT ;

END_VAR
```

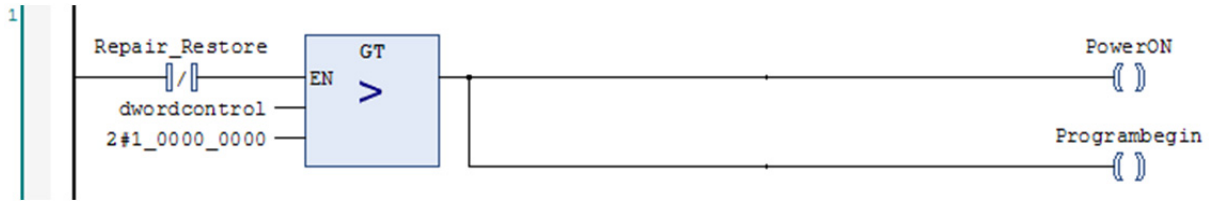
```
//initiate algorithm//

initiator2 := visuglob . autoselect ;

trig ( clk := initiator2 , q => step2 ) ;

IF step2 = TRUE THEN

FOR i := 1 TO 5 BY 1 DO


//update z
z := SQRT ( ( EXPT ( Rad , 2 ) ) - ( ( EXPT ( x , 2 ) + EXPT ( y , 2 ) ) ) ) ;

visuglob . calculatedX := Z ;

//for positive feasible values of x
```

```
18
19      IF  x > 0  THEN
20
21      //transform axis
22
23      znew := y ;
24
25      xnew := z ;
26
27      ynew := x ;
28
29      //calculate phi and theta
30
31      phidash := ACOS ( ( znew / rad ) ) ;
32
33      phidash_M := phidash * ( 180 / 3.14159 ) ;
34
35      visuglob . CalculatedZ := phidash_m ;
36
37      thetadash := ACOS ( xnew / ( rad * SIN ( phidash ) ) ) ;
38
39      thetadash_m := thetadash * ( 180 / 3.14159 ) ;
40
41      visuglob . Calculatedy := thetadash_m ;
42
43      //test_1:=rad*SIN(thetadash)*SIN(phidash);  (should equal z)
44
45      //calculate corrections
46
47      zcorrec := 10 * SIN ( phidash ) * COS ( thetadash ) ;
48
49      xcorrec :=  10 * SIN ( phidash ) * SIN ( thetadash ) ;
50
51      ycorrec :=  10 * COS ( phidash ) ;
52
53      //update new values
54
55      x := x - xcorrec ;
56
57      y := y - ycorrec ;
58
59      z := z - zcorrec ;
60
61      END_IF
62
63      END_FOR
64
65      END_IF
66
```

The iterative algorithms presented here is a model for a later more exact implementation on a separate environment like MATLAB. An iterative loop ideally should be kept separate than the main controller implementing safety functions. This is because, a case of page faulting (crash) of CPU, or indefinite computing time (indefinite loop) the safety of the complete program is undermined.

6.1.6 Restore

When system reboots itself after unintentional power failure during the experiment, the system checks the control register which is stored in the memory and if the value is above 2^8. The system sets the ‘Power ON’ and ‘ProgramBegin’ state variables to True state. Thus it bypasses the time required for user authentication. User authentication has to be still made to access the GUI.



6.2 Secondary functions

Secondary functions are not critical to the operation of the plant directly and these functions need administrative rights to be accessed. For example, when a lamp is powered on a time of operation is recorded and displayed to the administrator in order to calculate the power consumption which is necessary to calculate the costs of operation. The number of lamps powered on, module positioning data are also important parameters and are recorded at all times during experiment.

7. Safety functions

The control functions required to bring the system in safe state are explained in this section. These have been coded in the software as a part of overall risk reduction strategy previously mentioned. For the complete coding of these functions, please refer to **Appendix-I**.

7.1 Emergency stop button

On pressing the software side emergency button, a timer is instantiated, at whose output lamps are switched off one after another with a delay of 10 seconds. The cooling fans however continue to run for another 30 minutes.

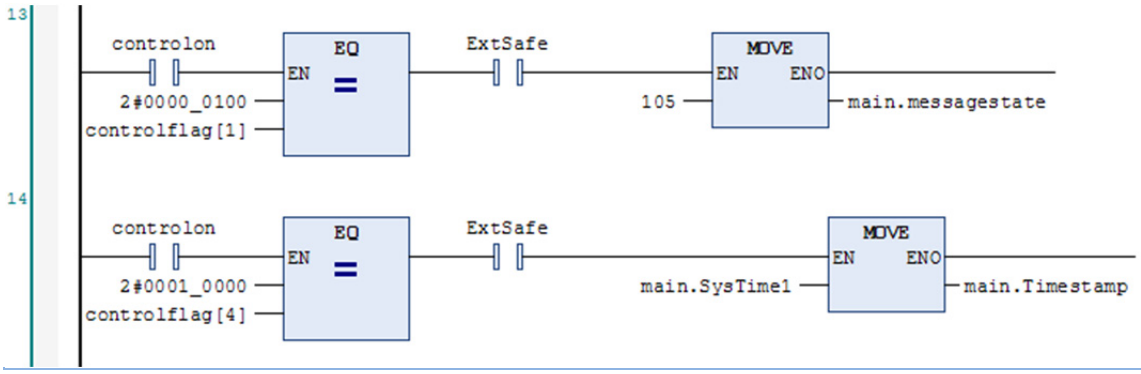
7.2 System time

The data structure ‘Daytime’ is created to get the system time for the PLC controller. The function ‘NT_GetTime’ is instantiated as ‘UpdateTime’ and variable ‘Daytime’ is passed to this function. The output of this function call is stored inside ‘Daytime’.

Since these values are in the form of Byte data, they have to be first converted to string datatypes. There are three different bytes, each storing hour, minute, and second respectively. These bytes after conversion to string values are concatenated and transferred to GUI as one single string. The coding is provided in Appendix-I, Section-1.

7.3 Alarm log

Each alarm triggers a specific 8 bit flag which is then passed on to the 32 bit control register. At each alarm the variable ‘MessageState’ is changed and the String value corresponding to alarm is displayed on the GUI.



The alarms are displayed on the GUI in the form of a table as shown in the figure below. Two buffers have been created in the system memory for this. One stores the alarm String, every time a new alarm is triggered and the other buffer is used to store the time data for each alarm. The buffers are one dimensional array of length 8. The function used to move the String value in the buffer is called 'Logger'. The function receives the pointer of previous alarm string, the length of previous alarm string, the address of buffer and the length of buffer. At every function call the previous string is moved in the buffer at a memory value equal to the length of the string and the new string is placed at the starting memory value of the buffer i.e. the pointer to the buffer. Thus a First In First Out queuing of data is achieved. The system time is also obtained on each alarm instance and is passed to the same function 'logger' and is displayed on the GUI along the alarm String.

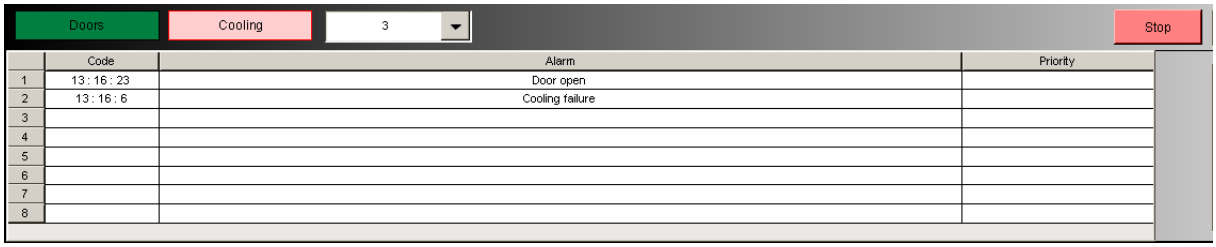


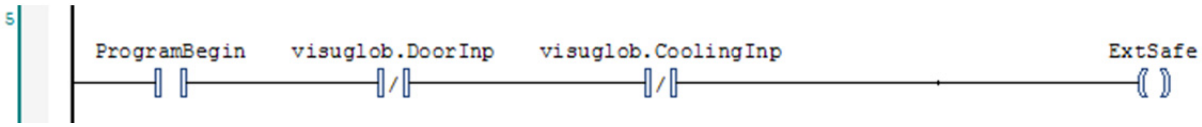
Figure 29: GUI-Alarm log

```
FUNCTION Logger : BOOL

VAR_INPUT

    cbdata : UDINT ;

    cbbuffer : UDINT ;

    pdata : POINTER TO BYTE ;

    pbuffer : POINTER TO BYTE ;

END_VAR

VAR

END_VAR
```

```
fw_memMove (

    pdest := pBuffer + cbdata ,

    psrc := pbuffer ,

    cblen := cbbuffer - cbdata
) ;

fw_memmove (

    pdest := pbuffer ,

    psrc := pdata ,

    cblen := cbdata
) ;
```

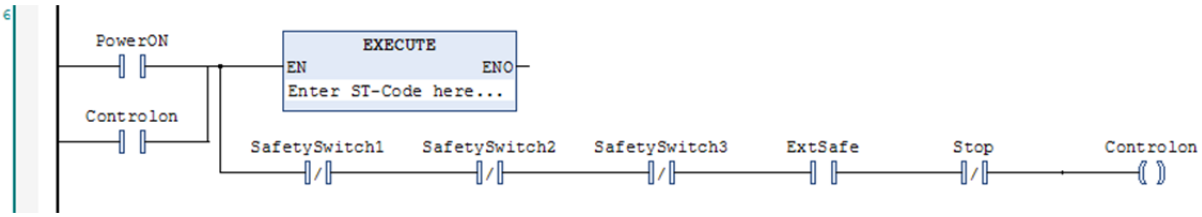
7.4 First safety check

In the state 'ProgramBegin' the PLC checks for inputs from other bus systems in the building namely: HVAC and Access management inputs. On successful confirmation of these inputs the system is put on externally safe state 'ExtSafe'. The coding is provided in Appendix-I, Section-3.



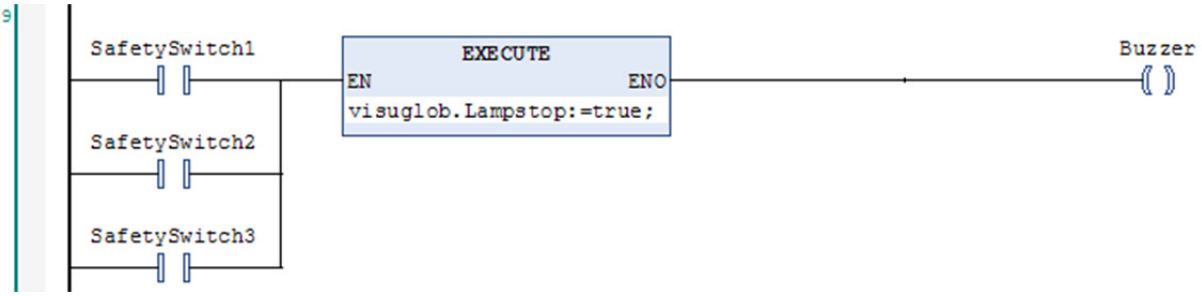
7.4 Second safety check

In the follow up of 1st safety check, the user is required to press the power on button to put start the module control system. This module control system is started only when all the safety switches in the building are in closed state.



7.5 Safety switches

When either of the safety switches is pressed, the software initiates the algorithm to stop the lamps after successive delays and signals the Buzzer output in the building.

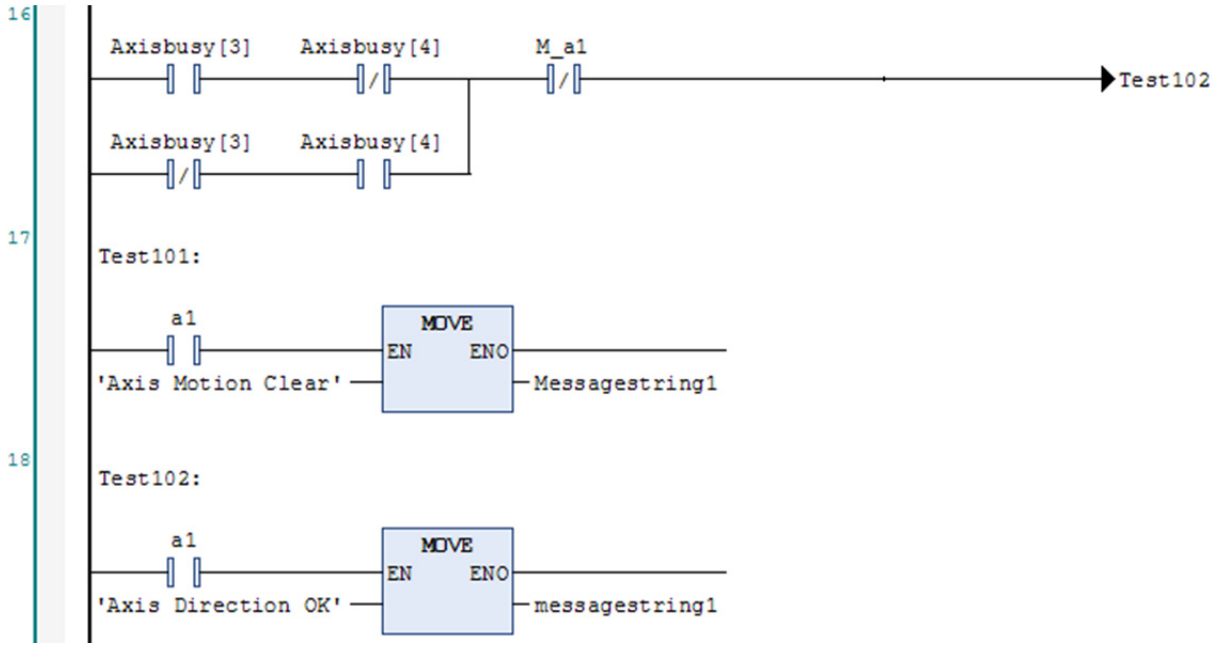


7.6 Axis sequence test

In this part, the program 'Registering_1' is passed 'Axis.Busy' output from motor controllers. This output is passed for the axis currently in selection and the axis last operated. If the 'Axis.Busy' output for both the axes is true, then alarm is generated.

7.7 Axis direction test

In this part, the controller checks the 'Feed positive enable' with the 'Feed negative enable' value. If both of these are true, an alarm is generated. These functions can be accessed as follows:



8. Repair mode

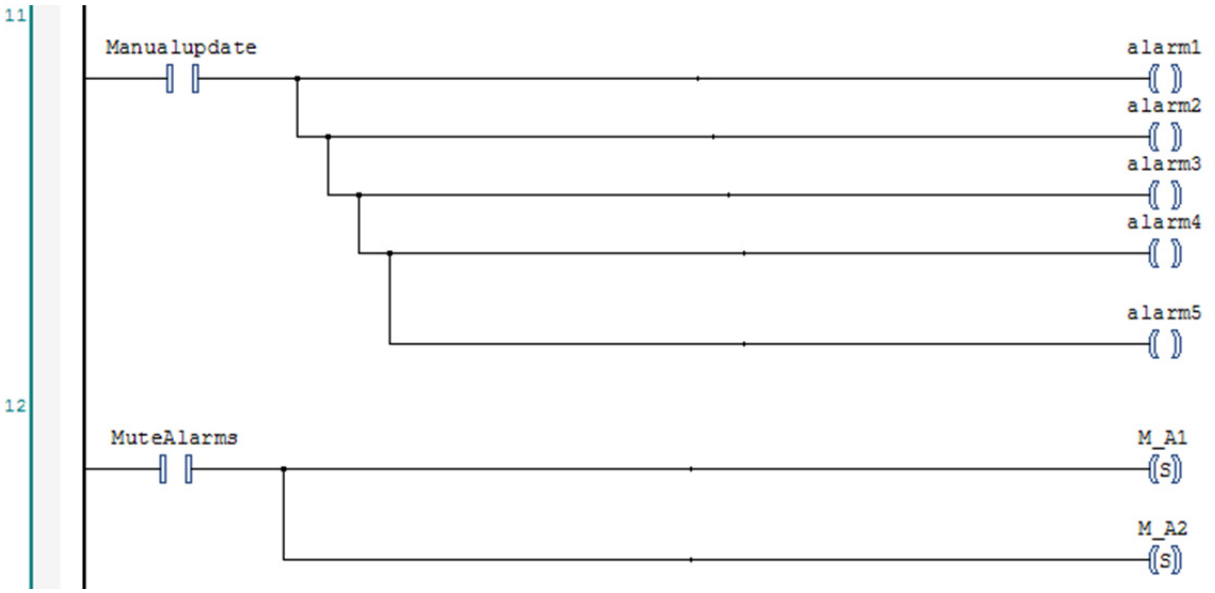
The variable 'Repair_Restore' when set to True state, provides the programmer to trouble shoot the software from troubleshooting program 'Registering_1' presented to programmer as ladder logic.

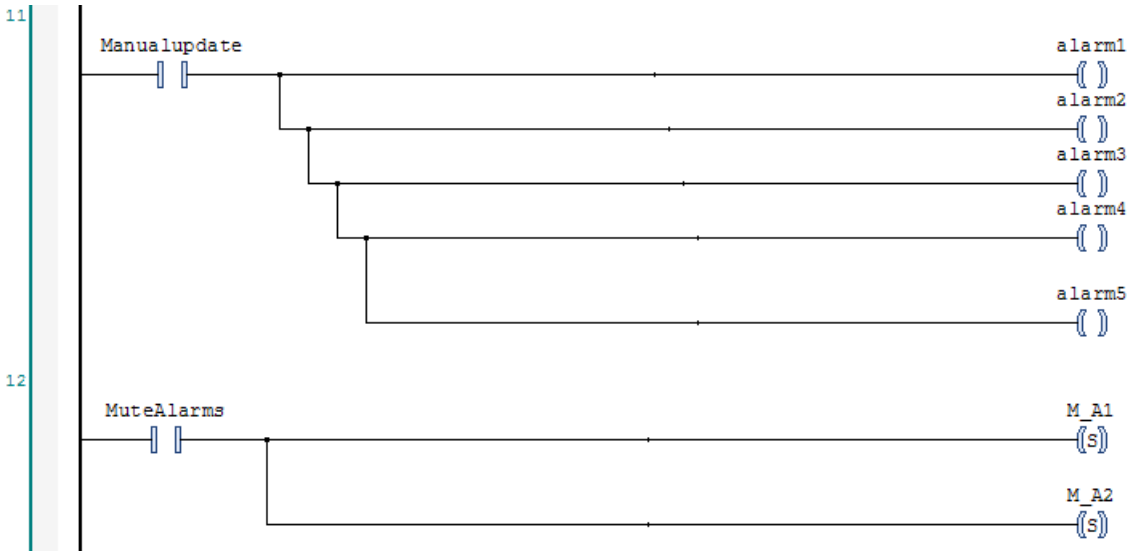
The programmer can stop the PLC and set it to configuration mode.

8.1 Update/Mute alarms

In the troubleshooting mode the alarms can be suppressed by the user on switching the 'MuteAlarms' variable in program 'Registering_1' to true state graphically.

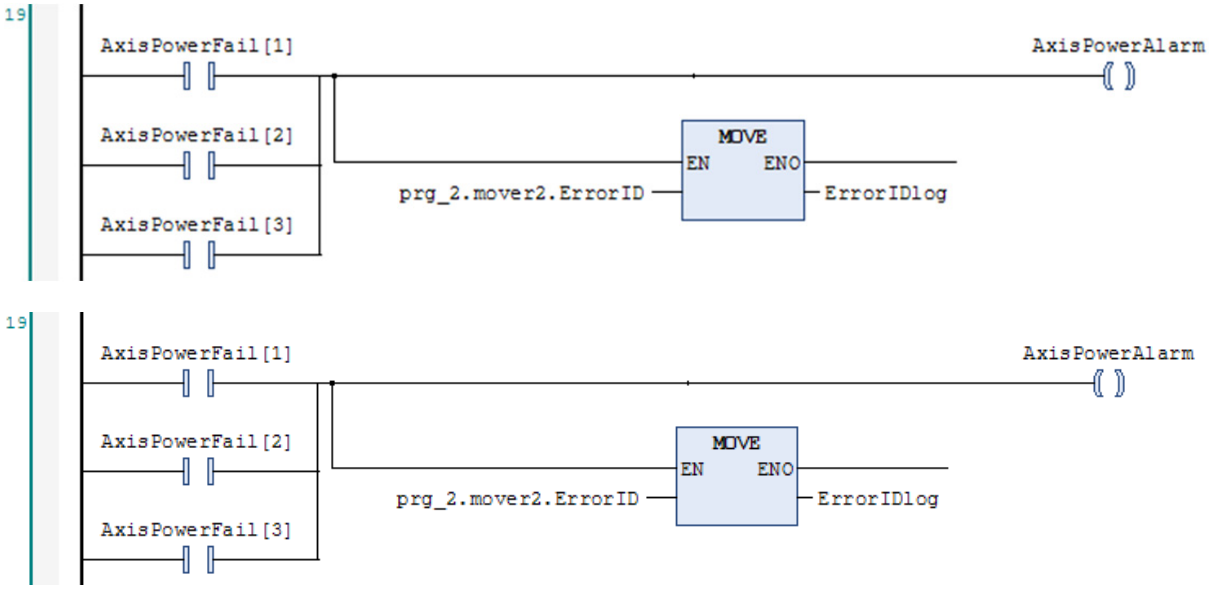
The alarms can also be updated by passing True value to 'ManualUpdate' variable in the program.





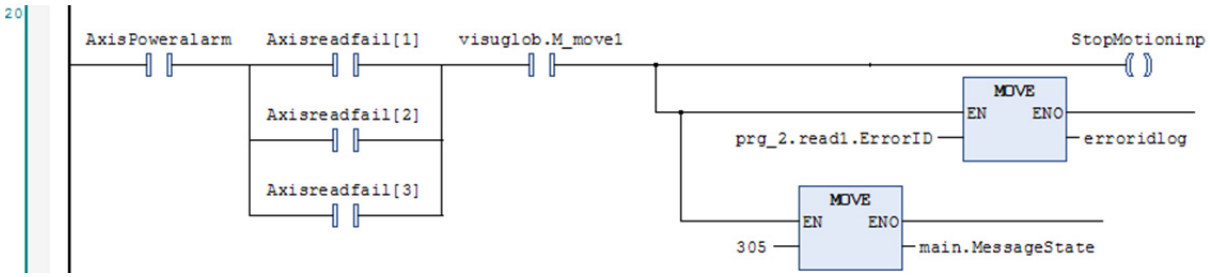
8.2 Axis power test

When the power failure is reported by any of the three axes in the module, the error ID is displayed to the programmer and 'AxisPowerAlarm' coil is set to True value.



8.3 Axis read test

When error output of motion read function in the main program goes to True state for any of the axes in the module, 'Axis read fail' variable is set to True value. If an axis power alarm is also present, then the 'StopMotionInp' coil is set to True value and any further values for motion are taken from GUI. The corresponding error ID is also displayed.



9. Conclusion and Future scope

The work undertaken was the functional as well as safety design of the control system along with the programming using an open international standard IEC 61131-3. Safety requirements set by the German Aerospace Centre have been thoroughly considered and the coding presented.

From the operational point of view of, the GUI developed presents a complete set of functions to position any radiation module, operate the rectifiers, lasers and the cooling fans. The testing was made on the prototype successfully. The limitations of the hardware and different motor configurations were identified in this manner. The collision algorithm was developed in addition to the DLR's requirements set for the Thesis.

From the safety perspective, a thorough analysis has been made on the primary failure modes and simulated on the software. IEC/EN 62061 guidelines have been incorporated in the code. Multiple safety checks have been incorporated in the programming. The user is also provided with an interface for troubleshooting the control system. The set point generation program provides an iterative procedure as well which can generate accurate module positioning under practical conditions of bending and assembly errors.

However, the development of a control system is an iterative process itself. Since the mechanical design of the modules is under review, the calibration data remains to be incorporated in the programs at a later stage. There is an immense scope of further improvements in the design and coding of the control system as well. As such, the current work is intended to serve as a model for future programming undertaken at DLR.

The future programming of the control system development shall include configuration of the hardware for 149 modules, and coding for the same based on the concepts developed in this Thesis. The successful implementation would certainly require a centralized controller with computing power more than the one used on the prototype in the current Thesis. On the software side, a GUI developed in C# shall also be considered.

For a more risk free implementation of the iterative algorithm for set point generation and the collision algorithm, a dedicated code can be written in MATLAB which can be called on request from the TwinCAT program. Such a model would prevent the PLC being exposed to an infinite loop and errors.

References

1. Kai Wieghardt, Karl-Heinz Funken, Gerd Dibowski , Bernhard Hoffschmidt, " SynLight-The World's Largest Artificial Sun".
2. German Aerospace Center, "Project Data- mHLS facility".
3. OSRAM GmbH, "Data sheet for XBO-XL (7000 W) lamp ".
4. IREM SpA, "Data sheet for rectifier series EX-200GM/3-E".
5. Beckhoff Automation GmbH & Co. KG. , "Datasheet for CX5130 embedded PC, TwinCAT software, KL2541 motor controller cards, BK 9000 bus couplers".
6. IEC 61508, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems".
7. IEC/EN 62061, "Safety of machinery: Functional safety of electrical, electronic and programmable electronic control systems".
8. EN 60204-1, "Documentation and markings on machinery".
9. IEC 61131-3, "Syntax and semantics of programming languages for programmable controllers".
10. D.Smith, K Simpson, "Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety, and ISO 13849"
11. Gould, J. (2000), "*Review of Hazard Identification Techniques*, HSE"
12. DeLong, Thomas (1970), "A Fault Tree Manual". Master's Thesis (Texas A&M University).

Appendix-I

Section-1: Part 1

```
PROGRAM  MAIN

VAR_INPUT

    Timestamp : STRING ;

    MessageState : INT ;

END_VAR

VAR

    OperationMode : Mode ;

    Rights : STRING ;

    u : INT ;

    ture :  BOOL ;

    SysTime1 : STRING ;

    //Register Read Write

    RegisterTimer : Ton ;

    //CoolingTimer//

    ClTimer : ton ;

    ///Alarm Log//

    LastMessageState : INT ;

    Message : STRING ;

    MessageLog : ARRAY [ 1 .. 8 ] OF  STRING ;

        ///Time log

    Timelog : ARRAY [ 1 .. 8 ] OF  STRING ;

    //coordinates

    c1 : INT ;

    c2 : INT ;

    checkflag :  BYTE ;

END_VAR
```

```
Registering_1 ( ) ;
pr1 ( ) ;
prg_2 ( ) ;
```

```
4
5       prog ( ) ;
6       /// Screening Begin///
7
8       header := 'Please enter your credentials...' ;
9
10      IF  username = 'dlr'  AND  password = 'admin'  THEN
11
12          header := 'Credentials verified' ;
13
14          operationmode := Admin ;
15
16          visuglob . UserID := 'Administrator' ;
17
18      END_IF
19
20      IF  username = 'dlr'  AND  password = 'user'  THEN
21
22          header := 'Credentials verified' ;
23
24          operationmode := User ;
25
26          visuglob . UserID := 'User' ;
27
28          //visuglob.okbutton2:=TRUE;
29
30      END_IF
31
32      Screening ( operationmode ) ;
33
34      ///Screening End///
35
36      //Regsiter Setup begin//
37
38      IF  visuglob . okbutton2 = TRUE  THEN
39
40          Registertimer ( in := TRUE , pt := T#5S ) ;
41
42          statusbar := TIME_TO_INT ( registertimer . ET ) / 200 ;
43
44          closebutton1 := TRUE ;
45
46          IF  Registertimer . Q = TRUE  THEN
47
48          closebutton1 := FALSE ;
49
50          END_IF
51
52      END_IF
53
54      //Register Setup End//
55
56      ////cooling Timer begin///
57
58      IF  lampstop = TRUE  THEN
59
```

```
60          cltimer ( in := TRUE , pt := T#1M4S ) ;
61
62          visuglob . CoolingTimer := cltimer . ET ;
63
64      END_IF
65
66      ///cooling Timer end///
67
68      ///lampstop begin//
69
70      IF  lampstop = TRUE  THEN
71
72          FOR  u := 1  TO  150  BY  1  DO
73
74              visuglob . LampS [ u ] := FALSE ;
75
76          END_FOR
77
78      ELSE  IF  lampstop = FALSE  THEN
79
80          cltimer ( in := FALSE ) ;
81
82          visuglob . CoolingTimer := cltimer . ET ;
83
84          visuglob . LampS [ 51 ] := TRUE ;
85
86          visuglob . LampS [ 52 ] := TRUE ;
87
88          visuglob . LampS [ 53 ] := TRUE ;
89
90          visuglob . LampS [ 70 ] := TRUE ;
91
92          visuglob . LampS [ 69 ] := TRUE ;
93
94          visuglob . LampS [ 68 ] := TRUE ;
95
96          visuglob . LampS [ 67 ] := TRUE ;
97
98          visuglob . LampS [ 80 ] := TRUE ;
99
100         visuglob . LampS [ 81 ] := TRUE ;
101
102         visuglob . LampS [ 82 ] := TRUE ;
103
104     END_IF
105
106     END_IF
107     ///lamp stop end//
108     ///alarm log begin///
109     CASE  messagestate  OF
110
111         101 :
112
113         message :=  'Cooling failure' ;
114
115         103 :
```

```
116
117          message := 'Door open' ;
118
119          911 :
120
121          message := 'Hardware Failure' ;
122
123          ELSE
124
125          message := 'in operation' ;
126
127      END_CASE
128
129          IF  lastmessagestate <> messagestate  THEN
130
131          logger (
132
133          cbdata := SIZEOF ( message ) ,
134
135          pdata := ADR ( message ) ,
136
137          cbbuffer := SIZEOF ( messagelog ) ,
138
139          pbuffer := ADR ( messagelog ) ) ;
140
141          logger (
142
143          cbdata := SIZEOF ( timestamp ) ,
144
145          pdata := ADR ( timestamp ) ,
146
147          cbbuffer := SIZEOF ( timelog ) ,
148
149          pbuffer := ADR ( timelog ) ) ;
150
151      alarmrow [ 1 ] . Alarm := messagelog [ 1 ] ;
152
153      alarmrow [ 1 ] . code := timelog [ 1 ] ;
154
155      alarmrow [ 2 ] . Alarm := messagelog [ 2 ] ;
156
157      alarmrow [ 2 ] . Code := timelog [ 2 ] ;
158
159      alarmrow [ 3 ] . Alarm := messagelog [ 3 ] ;
160
161      alarmrow [ 3 ] . code := timelog [ 3 ] ;
162
163      lastmessagestate := messagestate ;
164      END_IF
165      ///alarm log end
166      ///Alarms begin//
167          //messagestate:=103;//
168          IF  visuglob . CoolingInp = TRUE  THEN
169
170              messagestate := 103 ;
171
```

```
172             timestamp := systime ;
173
174         END_IF
175
176         //messagestate:=101;//
177         IF  visuglob . DoorInp = TRUE  THEN
178
179             messagestate := 101 ;
180
181             timestamp := systime ;
182
183         END_IF
184
185     ///alarms end//
186
187     ////Timing begin///
188         updatetime (
189
190         start := TRUE ,
191
192         timestr => daytime ,
193
194         tmout := T#3H
195
196         ) ;
197
198         seconds := WORD_TO_STRING ( daytime . wSecond ) ;
199
200         minute := WORD_TO_STRING ( daytime . wminute ) ;
201
202         hour := WORD_TO_STRING ( daytime . wHour ) ;
203
204         hour := concat ( str1 := hour , str2 := ' : ' ) ;
205
206         minute := concat ( str1 := minute , str2 := ' : ' ) ;
207
208         systime1 := concat ( str1 := hour , str2 := minute ) ;
209
210         systime := concat ( str1 := systime1 , str2 := seconds ) ;
211
212         updatetime (
213
214         start := FALSE ,
215
216         timestr => daytime ,
217
218         tmout := T#3H
219
220         ) ;
221
222         seconds := WORD_TO_STRING ( daytime . wSecond ) ;
223
224         minute := WORD_TO_STRING ( daytime . wminute ) ;
225
226         hour := WORD_TO_STRING ( daytime . wHour ) ;
227
```

```
228         hour := concat ( str1 := hour , str2 := ' : ' ) ;
229
230         minute := concat ( str1 := minute , str2 := ' : ' ) ;
231
232         systime1 := concat ( str1 := hour , str2 := minute ) ;
233
234         systime := concat ( str1 := systime1 , str2 := seconds ) ;
235     ///Timing end///
236     ///coordinates begin///
237     FOR  c1 := 1  TO  16  BY  1  DO
238
239         FOR  c2 := 1  TO  12  BY  1  DO
240
241             targetarray [ c1 , c2 ] . CY := c1 * 25 ;
242
243             targetarray [ c1 , c2 ] . CZ := c2 * 33 ;
244
245         END_FOR
246
247     END_FOR
248
249     FOR  c1 := 1  TO  16  BY  1  DO
250
251         FOR  c2 := 1  TO  12  BY  1  DO
252
253             IF  visuglob . TargetDisp [ c1 , c2 ] = TRUE  THEN
254
255                 targetZ := targetarray [ c1 , c2 ] . CZ ;
256
257                 targetY := targetarray [ c1 , c2 ] . CY ;
258
259                 prog . x := visuglob . targetY ;
260
261                 prog . y := visuglob . targetZ ;
262
263             END_IF
264
265         END_FOR
266
267     END_FOR
268
269     ///coordinates end///
270
271     ///control flags begin///
272
273     IF  alarmR [ 1 ] = TRUE  THEN
274
275         controltrig [ 1 ] ( ClK := TRUE ) ;
276
277         IF  controltrig [ 1 ] . Q = TRUE  THEN
278
279             controlFlag [ 1 ] := SHL ( controlbs , 2 ) ;
280
281             dwordcontrol := dwordcontrol  OR  controlFlag [ 1 ] ;
282
283         END_IF
```

```
284
285         ELSE  IF
286
287             alarmr [ 1 ] = FALSE  THEN
288
289             controlflag [ 1 ] := 2#0000_0000 ;
290
291             controltrig [ 1 ] ( CLK := FALSE ) ;
292
293             dwordcontrol := clearbit32 ( inval32 := dwordcontrol , bitno := 2 ) ;
294
295         END_IF
296
297     END_IF
298
299     IF  alarmR [ 2 ] = TRUE  THEN
300
301         controltrig [ 2 ] ( ClK := TRUE ) ;
302
303         IF  controltrig [ 2 ] . Q = TRUE  THEN
304
305         controlFlag [ 2 ] := SHL ( controlbs , 4 ) ;
306
307         dwordcontrol := dwordcontrol  OR  controlFlag [ 2 ] ;
308
309
310
311         END_IF
312
313         ELSE  IF
314
315             alarmr [ 2 ] = FALSE  THEN
316
317             controlflag [ 2 ] := 2#0000_0000 ;
318
319             controltrig [ 2 ] ( CLK := FALSE ) ;
320
321             dwordcontrol := clearbit32 ( inval32 := dwordcontrol , bitno := 4 ) ;
322
323         END_IF
324     END_IF
325
326
327         IF  ( dwordcontrol  AND  controlFlag [ 1 ] ) = 2#000_0100  THEN
328
329             messagestate := 101 ;
330
331         END_IF
332
333         IF  ( dwordcontrol  AND  controlFlag [ 2 ] ) = 2#0001_0000  THEN
334
335             messagestate := 103 ;
336
337         END_IF
338         IF  ( dwordcontrol  AND  ( controlFlag [ 1 ] OR  controlflag [ 2 ] ) ) =
    2#0001_0100  THEN
339
340             messagestate := 911 ;
341
342         END_IF
343
```

Section-1: Part 2

```
FUNCTION Screening : Bool

VAR_INPUT

    Access : Mode ;

END_VAR
```

```
CASE Access OF

    User :

    visuglob . FanOnVisib := TRUE ;

    visuglob . FanOffVisib := TRUE ;

    visuglob . FanIconVisib := TRUE ;

    visuglob . LaserIconVisib := TRUE ;

    visuglob . LaserOffVisib := TRUE ;

    visuglob . LaserOnVisib := TRUE ;

    Admin :

    visuglob . FanOnVisib := FALSE ;

    visuglob . FanOffVisib := FALSE ;

    visuglob . FanIconVisib := FALSE ;

    visuglob . LaserIconVisib := FALSE ;

    visuglob . LaserOffVisib := FALSE ;

    visuglob . LaserOnVisib := False ;

END_CASE
```

Section-1: Part 3

```
TYPE st1 :

    STRUCT

    Number : INT ;

    Code : STRING ;

    Alarm : STRING ;

    Priority : STRING ;

END_STRUCT

END_TYPE
```

```
TYPE  target  :

STRUCT

    CZ : REAL ;

    CY : REAL ;

END_STRUCT

END_TYPE
```

Section-1: Part 4

```
VAR_GLOBAL

    Daytime : timestruct ;

    Hour : STRING ;

    Minute : STRING ;

    Seconds : STRING ;

    UpdateTime : Nt_gettime ;

    ///control flag prog2//

    Control_Cont :  BYTE := 2#0000_0001 ;

    controlX : BYTE ;

    controlbase : BYTE ;

    //control flag general//

    controltrig : ARRAY [ 1 .. 50 ] OF  R_TRIG ;

    ControlBs : BYTE := 2#0000_0001 ;

    ControlX1 : BYTE ;

    ControlFlag : ARRAY [ 1 .. 8 ] of  BYTE ;

    ControlReg : BYTE := 2#0000_0001 ;

    AlarmR : ARRAY [ 1 .. 50 ] OF  BOOL ;

    dwordcontrol : DWORD ;

    controlchek : BYTE ;

END_VAR
```

Section-1: Part 5

```
VAR_GLOBAL

    username , header , password : STRING ;

    okbutton1 : BOOL ;

    okbutton2 : BOOL ;

    //Registering Start

    CloseButton1 : BOOL ;

    CloseButton2 : BOOL ;

    Statusbar : int ;

    ///visu1

    SysTime : STRING ;

    ExitBtn : BOOL ;

    DoorInp : BOOL ;

    CoolingInp : BOOL ;

    SwitchUsr : BOOL ;

    UserID : STRING ;

    Lampstop : Bool ;

    ///Visu 6

    LaserOn : BOOL ;

    LaserOFF : BOOL ;

    FanON : BOOL ;

    FanOFF : BOOL ;

    FanSpeedTxt : INT ;

    FanSpeedSel : INT ;

    FanStatus : BOOL ;

    FanOnVisib : BOOL ;

    FanOffVisib : BOOL ;

    FanIconVisib : BOOL ;

    LaserOnVisib : BOOL ;

    LaserOffVisib : BOOL ;

    LaserIconVisib : BOOL ;

    CoolingTimer : TIME ;
```

```
62
63          M_value1 :  REAL ;
64
65          M_move1 :  BOOL ;
66
67          M_value2 :  REAL ;
68
69          M_move2 :  BOOL ;
70
71          M_value3 :  REAL ;
72
73          M_move3 :  BOOL ;
74
75          LampS :  ARRAY [ 1 .. 150 ] OF  BOOL ;
76
77          AlarmRow :  ARRAY [ 1 .. 8 ]  OF  st1 ;
78
79          targetbutton :  BOOL ;
80
81          spin : INT ;
82
83          spinshow :  BOOL := TRUE ;
84
85          targetY : LREAL ;
86
87          TargetArray : ARRAY  [ 1 .. 16 , 1 .. 12 ] OF  target ;
88
89          TargetDisp : ARRAY [ 1 .. 16 , 1 .. 12 ] OF  BOOL ;
90
91          targetZ :  LREAL ;
92
93          DisplayMotion1 :  INT ;
94
95          displaymotion2 :  INT ;
96
97          positionx : LREAL ;
98
99          positiony : LREAL ;
100
101         autobutton : BOOL ;
102
103         autoselect :  BOOL ;
104
105         manual : BOOL ;
106
107         xaxisfinish :  BOOL ;
108
109         yaxisfinish : BOOL ;
110
111         calculatedX : LREAL ;
112
113         Calculatedy : LREAL ;
114
115         CalculatedZ : LREAL ;
116
117         /// visu8_1
118
119         X_Vel_Manual : UDINT ;
120
121         fbutton :  BOOL ;
122
123         bbutton :  BOOL ;
```

```
124
125         lbutton : BOOL ;
126
127         rbutton : BOOL ;
128
129         ubutton :  BOOL ;
130
131         dbutton : BOOL ;
132
133         pos_x : LREAL ;
134
135         pos_theta : LREAL ;
136
137         pos_phi : LREAL ;
138
139         stopM1 : BOOL ;
140
141         stopM2 : BOOL ;
142
143         stopM3 : BOOL ;
144
145      END_VAR
146
```

Section-1: Part 6

```
1     FUNCTION_BLOCK  PUBLIC  MC  IMPLEMENTS  itf1
2
3     VAR_INPUT
4
5         Xaxis : Axis ;
6
7         Yaxis : Axis ;
8
9         Zaxis : Axis ;
10
11        A : ARRAY [ 1 .. 3 ] OF   axis_ref ;
12
13        exec : BOOL ;
14
15        dist : REAL ;
16
17    END_VAR
18
```

Section-2:

```
PROGRAM Prg_2
VAR
    ret_axs : ARRAY [ 1 .. 240 ] OF axis_ref ;

    ActPos : LREAL ;

    power1 : BOOL ;

    Tm1 : Ton ;

    Tm1_input : BOOL ;

    M : ARRAY [ 1 .. 80 ] OF MC ;

    i : USINT ;

    k : USINT ;

    Sel_1 : INT ;

    Sel_2 : INT ;

    finish : BOOL ;

    mover1 : mc_moveabsolute ;

    mover2 : mc_moveabsolute ;

    doneR : BOOL ;

    doneR2 : BOOL ;

    inter1 : INT ;

    read1 , read2 : mc_readactualposition ;

    stopbutton1 : BOOL ;

    jog1 , jog3 , jog4 : mc_jog ;

    jog2 : mc_jog ;

    halt1 : BOOL ;

    halt2 : BOOL ;

    finish2 : BOOL ;

END_VAR
```

```
Tm1 ( in := Tm1_input , PT := T#100MS ) ;

IF Tm1 . Q = TRUE THEN

    FOR i := 1 TO 80 BY 1 DO
```

```
            FOR k := 1 TO 3 BY 1 DO
                ret_axs [ ( i - 1 ) * 3 + k ] := M [ i ] . A [ k ] ;
            END_FOR
        END_FOR
    Tm1_input := FALSE ;
    END_IF
    Sel_1 := spin ;
    IF sel_1 <> 0 THEN
    M [ sel_1 ] . Xaxis . Power
    ( axis := M [ sel_1 ] . A [ 1 ] ,
    enable := TRUE ,
    enable_negative := TRUE ,
    enable_positive := TRUE ,
    override := 100 ) ;
    finish := visuglob . M_move1 ;
    IF finish = TRUE THEN
        controlbase := 2#0000_0001 ;
        controlX := SHL ( controlbase , 1 ) ;
        control_cont := control_cont OR controlx ;
        IF ( control_cont AND controlx ) = 2#0000_0010 THEN
            mover1 ( axis := m [ sel_1 ] . A [ 1 ] ,
            position := m_value1 ,
            execute := TRUE ,
            velocity := visuglob . X_Vel_Manual ,
            busy => doner ) ;
        IF mover1 . Done = TRUE THEN
            finish := FALSE ;
            control_cont := controlbase ;
```

```
62
63              m_move1 := FALSE ;
64
65              mover1 (
66
67              axis := m [ sel_1 ] . A [ 1 ] ,
68
69              position := m_value1 ,
70
71              execute := FALSE ,
72
73              velocity := 100 ,
74
75              busy => doner ) ;
76
77          END_IF
78
79          END_IF
80
81          // reset after stop command
82      ELSE
83
84      mover1 ( axis := m [ sel_1 ] . A [ 1 ] ,  execute := FALSE ) ;
85
86      END_IF
87
88      read1 ( axis := m [ sel_1 ] . A [ 1 ] ,
89
90          enable := TRUE ,
91
92          position => actpos ) ;
93
94          //prevent move and jog and once
95
96          IF  doner = FALSE  THEN
97
98          jog1 ( axis := m [ sel_1 ] . A [ 1 ] ,
99
100             jogforward := fbutton ,
101
102             velocity := visuglob . X_Vel_Manual ) ;
103
104             END_IF
105
106     jog2 ( axis := m [ sel_1 ] . A [ 1 ] ,
107
108     jogbackwards := bbutton ) ;
109
110     IF  stopbutton1 = TRUE  THEN
111
112     m [ sel_1 ] . Xaxis . stop (
113
114     axis := m [ sel_1 ] . A [ 1 ] ,
115
116     execute := TRUE , done => halt1 ) ;
117
```

```
118     //reset inputs for stop and move
119
120     IF  halt1 = TRUE  THEN
121
122     stopbutton1 := FALSE ;
123
124     m_move1 := FALSE ;
125
126     END_IF
127
128     ELSE
129
130     m [ sel_1 ] . Xaxis . stop (
131
132     axis := m [ sel_1 ] . A [ 1 ] ,
133
134     execute := FALSE ) ;
135
136     END_IF
137
138     //// copy for theta axis/////
139     M [ sel_1 ] . yaxis . Power
140
141     ( axis := M [ sel_1 ] . A [ 2 ] ,
142
143     enable := TRUE ,
144
145     enable_negative := TRUE ,
146
147     enable_positive := TRUE ,
148
149     override := 100 ) ;
150
151     finish2 := visuglob . M_move2 ;
152
153     IF  finish2  = TRUE  THEN
154
155         controlbase := 2#0000_0001 ;
156
157         controlX := SHL ( controlbase , 1 ) ;
158
159         control_cont := control_cont  OR  controlx ;
160
161         IF  ( control_cont  AND  controlx ) = 2#0000_0010  THEN
162
163             mover2 ( axis := m [ sel_1 ] . A [ 2 ] ,
164
165             position := m_value2 ,
166
167             execute := TRUE ,
168
169             velocity := visuglob . X_Vel_Manual ,
170
171             busy => doner2 ) ;
172
173         IF  mover2 . Done = TRUE  THEN
```

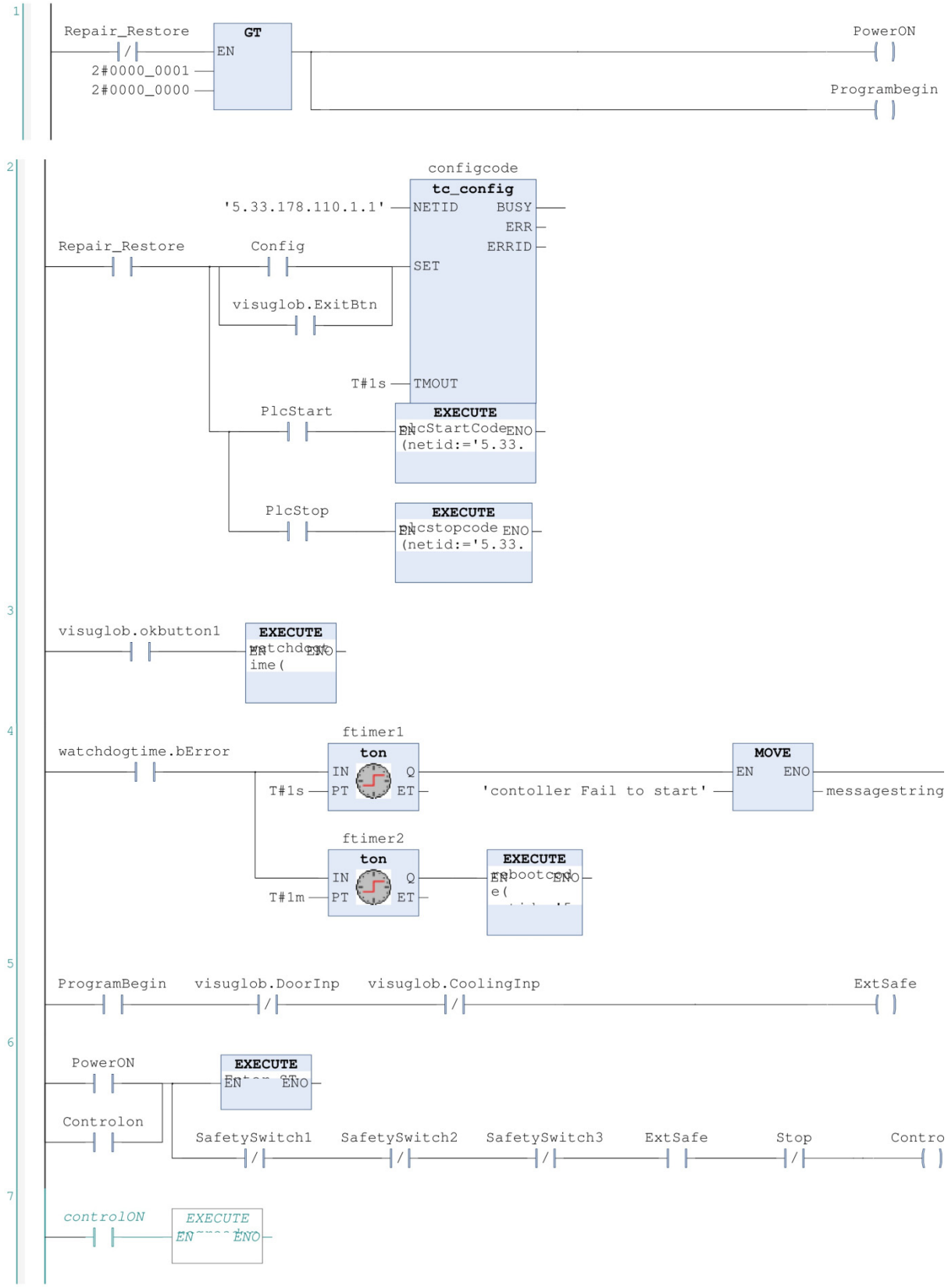
```
174
175             finish := FALSE ;
176
177             control_cont := controlbase ;
178
179             m_move2 := FALSE ;
180
181             mover2 (
182
183             axis := m [ sel_1 ] . A [ 2 ] ,
184
185             position := m_value2 ,
186
187             execute := FALSE ,
188
189             velocity := 100 ,
190
191             busy => doner2 ) ;
192
193         END_IF
194
195         END_IF
196
197         // reset after stop command
198         ELSE
199
200     mover2 ( axis := m [ sel_1 ] . A [ 2 ] ,  execute := FALSE ) ;
201
202         END_IF
203
204         read2 ( axis := m [ sel_1 ] . A [ 2 ] ,
205
206         enable := TRUE ,
207
208         position => visuglob . pos_theta ) ;
209         //prevent move and jog and once
210         IF  doner2 = FALSE  THEN
211
212             jog3 (
213
214             axis := m [ sel_1 ] . A [ 2 ] ,
215
216             jogforward := lbutton ,
217
218             velocity := visuglob . X_Vel_Manual
219             ) ;
220
221         END_IF
222
223             jog4 (
224
225             axis := m [ sel_1 ] . A [ 2 ] ,
226
227             jogbackwards := rbutton
228             ) ;
229
```

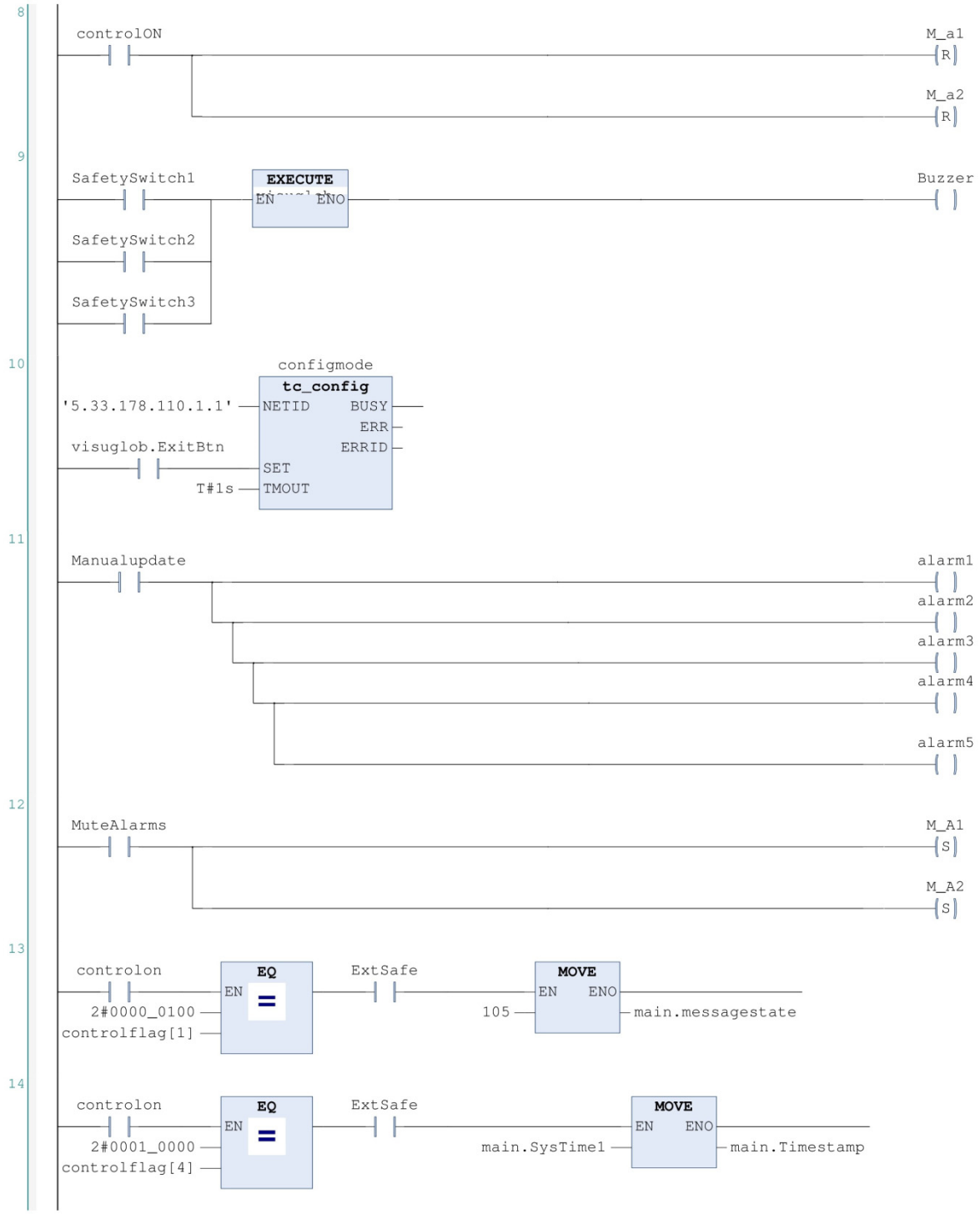
```
230         IF  visuglob . stopM2 = TRUE  THEN
231
232             m_move2 := FALSE ;
233
234             m [ sel_1 ] . yaxis . stop (
235
236             axis := m [ sel_1 ] . A [ 2 ] ,
237
238             execute := TRUE , done => halt2
239
240             ) ;
241     //reset inputs for stop and move
242             IF  halt2 = TRUE  THEN
243
244             visuglob . stopM2 := FALSE ;
245
246             END_IF
247         ELSE
248
249             m [ sel_1 ] . yaxis . stop (
250
251             axis := m [ sel_1 ] . A [ 2 ] ,
252
253             execute := FALSE ) ;
254
255     END_IF
256
257     END_IF
258
```

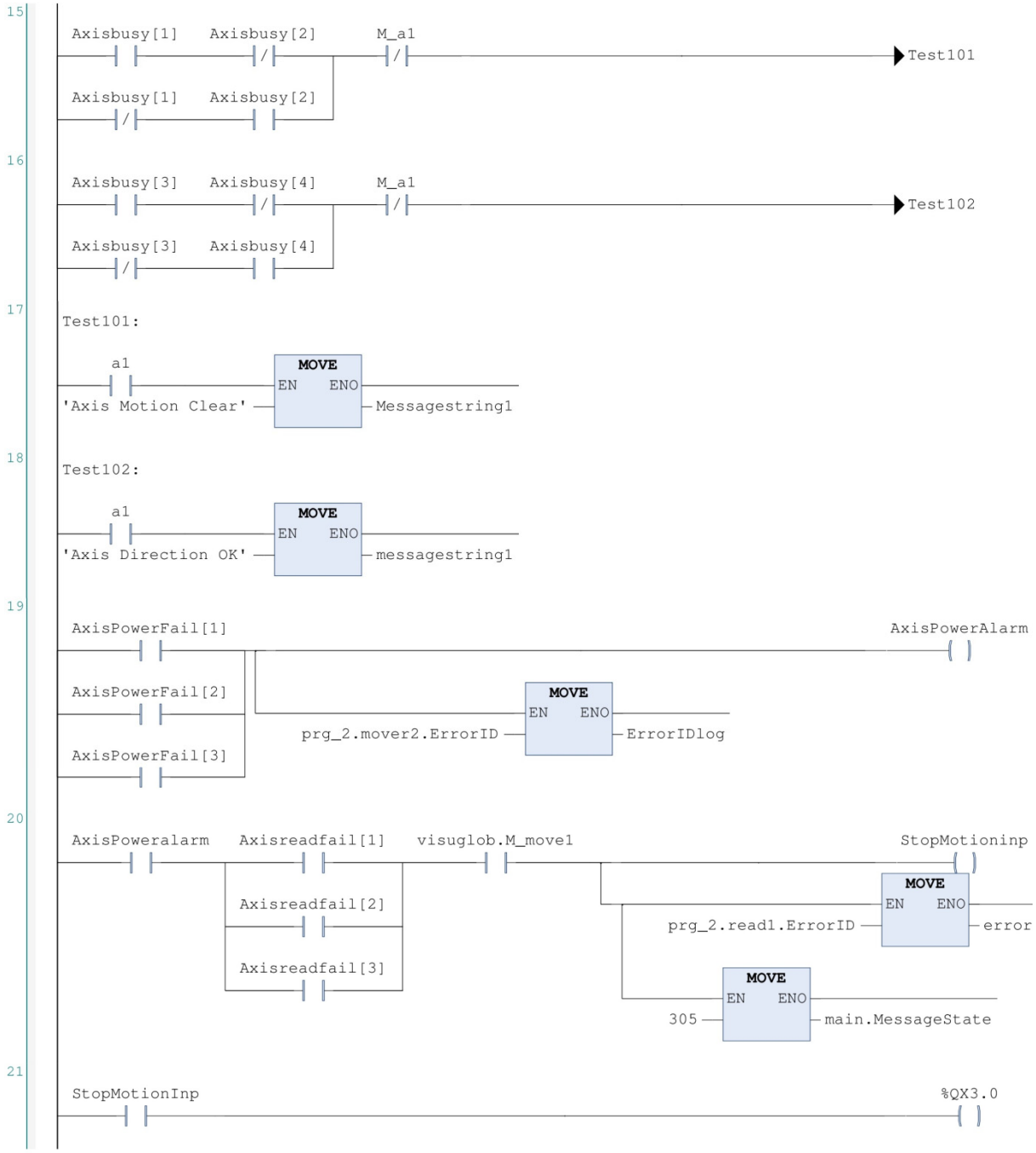
Section-3:

```
PROGRAM Registering_1

VAR

RegReadWrite : FB_RegisterComKL25xx ;

RegError : ARRAY [ 1 .. 50 ] OF BOOL ;

RegErrorID : ARRAY [ 1 .. 50 ] OF UDINT ;

TrmTyp : ARRAY [ 1 .. 50 ] OF UINT ;

OutRegNmb : ARRAY [ 1 .. 50 ] OF USINT ;

OutRegVal : ARRAY [ 1 .. 50 ] OF UINT ;

InRegNmb : ARRAY [ 1 .. 50 ] OF USINT ;

InRegVal : ARRAY [ 1 .. 50 ] OF UINT ;

ReadR : ARRAY [ 1 .. 50 ] OF BOOL ;

WriteR : ARRAY [ 1 .. 50 ] OF BOOL ;

ReadWriteRegBusy : BOOL ;

Repair_Restore : BOOL ;

Config : BOOL ;

PlcStop : BOOL ;

PlcStart : BOOL ;

ConfigCode : tc_config ;

PlcStartCode : plc_start ;

PlcStopCode : plc_stop ;

WatchDogTime : fb_pcwatchdog ;

RebootCode : nt_reboot ;

MessageString1 : STRING ;

FTimer1 : ton ;

ftimer2 : ton ;

ProgramBegin : BOOL ;

ExtSafe : BOOL ;

PowerON : BOOL ;

SafetySwitch1 : BOOL ;

SafetySwitch2 : BOOL ;

SafetySwitch3 : BOOL ;
```

```
62
63      STOP :  BOOL ;
64
65      ConrolON :  BOOL ;
66
67      ControlON :  BOOL ;
68
69      Buzzer :  BOOL ;
70
71       ConfigMode : tc_config ;
72
73       ManualUpdate : BOOL ;
74
75       chk1 :  BOOL ;
76
77       tarde :  BOOL ;
78
79       Alarm1 :  BOOL ;
80
81       alarm2 :  BOOL ;
82
83       alarm3 :  bool ;
84
85       alarm4 :  BOOL ;
86
87       alarm5 :  BOOL ;
88
89       a1 :  BOOL ;
90
91       a2 :  BOOL ;
92
93       a3 :  BOOL ;
94
95       MuteAlarms :  BOOL ;
96
97       M_A1 :  BOOL ;
98
99       M_A2 :  BOOL ;
100
101      AxisBusy :  ARRAY [ 1 .. 4 ]  OF  BOOL ;
102
103      Ftrig1 : r_trig ;
104
105      Ftrig2 : r_trig ;
106
107      FanOutput  AT %QW100 : INT ;
108
109      AxisPowerFail :  ARRAY [ 1 .. 3 ]  OF  BOOL ;
110
111      AxisPowerAlarm : BOOL ;
112
113      AxisReadfail :  ARRAY [ 1 .. 3 ]  OF  BOOL ;
114
115      del :  BOOL ;
116
117      StopMotionInp :  BOOL ;
118
119      ErrorIDlog :  UDINT ;
120
121      END_VAR
122
```


1
Repair_Restore
GT
EN
2#0000_0001
2#0000_0000
PowerON
Programbegin
2
configcode
tc_config
'5.33.178.110.1.1'
NETID
BUSY
ERR
ERRID
SET
T#1s
TMOUT
Repair_Restore
Config
visuglob.ExitBtn
PlcStart
EXECUTE
(netid:='5.33.
PlcStop
EXECUTE
(netid:='5.33.
3
visuglob.okbutton1
EXECUTE
4
watchdogtime.bError
ftimer1
ton
IN
PT
Q
ET
T#1s
MOVE
EN
ENO
'contoller Fail to start'
messagestring
ftimer2
ton
T#1m
EXECUTE
5
ProgramBegin
visuglob.DoorInp
visuglob.CoolingInp
ExtSafe
6
PowerON
EXECUTE
Controlon
SafetySwitch1
SafetySwitch2
SafetySwitch3
ExtSafe
Stop
7
controlON
EXECUTE


8
controlON
M_a1
R
M_a2
R
9
SafetySwitch1
EXECUTE
EN
ENO
Buzzer
SafetySwitch2
SafetySwitch3
10
configmode
tc_config
'5.33.178.110.1.1'
NETID
BUSY
ERR
ERRID
visuglob.ExitBtn
SET
T#1s
TMOUT
11
Manualupdate
alarm1
alarm2
alarm3
alarm4
alarm5
12
MuteAlarms
M_A1
S
M_A2
S
13
controlon
EQ
EN
=
2#0000_0100
controlflag[1]
ExtSafe
MOVE
EN
ENO
105
main.messagestate
14
controlon
EQ
EN
=
2#0001_0000
controlflag[4]
ExtSafe
MOVE
EN
ENO
main.SysTime1
main.Timestamp


15
Axisbusy[1]
Axisbusy[2]
M_a1
Test101
Axisbusy[1]
Axisbusy[2]
16
Axisbusy[3]
Axisbusy[4]
M_a1
Test102
Axisbusy[3]
Axisbusy[4]
17
Test101:
a1
MOVE
EN
ENO
'Axis Motion Clear'
Messagestring1
18
Test102:
a1
MOVE
EN
ENO
'Axis Direction OK'
messagestring1
19
AxisPowerFail[1]
AxisPowerAlarm
AxisPowerFail[2]
MOVE
EN
ENO
prg_2.mover2.ErrorID
ErrorIDlog
AxisPowerFail[3]
20
AxisPoweralarm
Axisreadfail[1]
visuglob.M_move1
StopMotioninp
MOVE
EN
ENO
Axisreadfail[2]
prg_2.read1.ErrorID
error
Axisreadfail[3]
MOVE
EN
ENO
305
main.MessageState
21
StopMotionInp
%QX3.0