Vietnamese-German University

COPYRIGHT WARNING

This paper is protected by copyright. You are advised to print or download **ONE COPY** of this paper for your own private reference, study and research purposes. You are prohibited having acts infringing upon copyright as stipulated in Laws and Regulations of Intellectual Property, including, but not limited to, appropriating, impersonating, publishing, distributing, modifying, altering, mutilating, distorting, reproducing, duplicating, displaying, communicating, disseminating, making derivative work, commercializing and converting to other forms the paper and/or any part of the paper. The acts could be done in actual life and/or via communication networks and by digital means without permission of copyright holders.

The users shall acknowledge and strictly respect to the copyright. The recitation must be reasonable and properly. If the users do not agree to all of these terms, do not use this paper. The users shall be responsible for legal issues if they make any copyright infringements. Failure to comply with this warning may expose you to:

- Disciplinary action by the Vietnamese-German University.
- Legal action for copyright infringement.
- Heavy legal penalties and consequences shall be applied by the competent authorities.

The Vietnamese-German University and the authors reserve all their intellectual property rights.

Vietnamese-German University

Copyright

RUHR-UNIVERSITÄT BOCHUM

MechEng
Mechanical Engineering

Vietnamese-German University

ENHANCING THE ABB ROBOTICS SCREW-DRIVING SYSTEM'S EFFICIENCY THROUGH THE IMRPOVEMENT OF MECHANICAL INSTALLATION

BACHELOR THESIS

Vietnamese-German University

BINH DUONG 2024

SUBMITTED BY: LE QUOC BAO

RUB STUDENT ID: 20267550

VGU STUDENT ID: 14598

SUPERVISOR: PROF. DR. NGUYEN QUOC HUNG

CO-SUPERVISOR: MSC. CHAU KHAC BAO CHUONG

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Mr. Nguyễn Minh Toàn, who directly guided me during my internship at ABB Robotics and also proposed the thesis topic as well as provided me with the equipment needed to complete it. During the working process, problems are inevitable and sometimes they lead to dead ends. However, thanks to Mr. Toàn's advice and dedicated assistance, I was able to complete this thesis smoothly. Additionally, I would like to thank Mr. Đoàn Minh Khang - MEN 2019, who worked directly with me on this thesis. Thanks to him, the progress and outcomes of the thesis were ensured.

And of course, I cannot forget to express my gratitude to the Vietnamese-German University. Throughout my studies here, I have been very fortunate to be taught by a dedicated faculty. Especially, the teachers in the Mechanical Engineering department who have imparted valuable knowledge to me, enabling me to complete this thesis and further my future career.

And finally, my heartfelt thanks go to my family. They have always been by my side, encouraging me through difficult times so that I could overcome them and complete this thesis. Without them, I certainly would not have been able to achieve this.

Sincerely, thank you!

TABLE OF CONTENT

VGU
Vietnamese-German University

Vietnamese-German University

LIST OF FIGURE AND TABLE

Vietnamese-German University

VGU
Vietnamese-German University

VGU
Vietnamese-German University

VGU
Vietnamese-German University

ABSTRACT

Robots are becoming increasingly popular as the world enters the era of Industry 4.0. Thanks to their versatility, robots can be utilized in a wide variety of life aspects and are even more extensively employed in the industrial and automation sectors. In conjunction with other mechanical devices such as conveyor belts and pre-designed tools, robots can significantly contribute to factory production lines by automating processes that previously required manual labor.

The purpose of this paper is to explain the process of operating and optimizing the efficiency of ABB Robotic's automatic screw driving system using ABB's industrial robot along with mechanical components. This project uses the ABB RobotStudio and ABB PLC as the main software to programming and controlling the robot's movement, and ABB Panel as the main human-machine interface.

VGU
Vietnamese-German University

Chapter 1. INTRODUCTION

1.1 Problem statement

The use of industrial robots to replace manual labor is becoming a highly favored trend in production lines. Thanks to their precision and processing speed, robots can achieve significantly higher efficiency than humans when performing such tasks. Until now, robots have already replaced humans in a wide range of industries, from light to heavy, including tasks such as pick-and-place, palletizing, 3D printing, assembly, and welding, etc….

While the screw-driving process may seem straightforward, it requires high precision and perseverance due to the repetitive nature of the task over an extended period. For humans, having to undergo this process over a long period and continuously can easily lead to fatigue both physically and mentally, resulting in a decline in effectiveness at work. Thus, The automatic screw driving system is designed to address this issue. By fully automating the entire process with industrial robots and conveyor belts, this system will ensure continuous operation and high performance, regardless of the duration of operation.

VGU
Vietnamese-German University

However, since the robot in this system is often programmed to move quickly and this can lead to screwing errors as the small screw positions can be difficult to target accurately. Therefore, Basic programming is just the first step. A lot of work still needs to be done to ensure that the system operate smoothly while still meeting the requirements.

1.2 Objectives

In this project, I am aiming to programming and operating the automatic screw-driving system based on a blueprint that has been calculated and manufactured by ABB Robotics. Throughout the process I, Doan Minh Khang, my thesis partner, and Mr. Nguyen Minh Toan, an ABB mechanical engineer, worked together. The objectives are to fully functioning the system and making sure it meets the requirements of cycle time and efficiency given by ABB Robotics.

1.3 Scope of implementation

- Content 1: Creating the basic movement of the robot.
- Content 2: Monitoring and evaluating current system performance.
- Content 3: Identify and address issues affecting system performance:
- Content 4: Design and manufacture new tool head.
- Content 5: Review and evaluate result of the new tool, finishing the system.
- Content 6: Finish the thesis.

1.4 Limitation

VGU
Vietnamese-German University

This project focus on programming and operating the industrial robot in the demo screw-driving station while focusing mostly on how to optimize and increase the efficiency. The station includes:

- ABB IRB 1100 robot and OmniCore controller.
- The screw driving tool set includes: screw feeder, screwdriver, custom-designed mechanical tool.
- The conveyor system consists of four conveyors, the two conveyors from the left and right has cylinder to lift up and down.
- Custom-designed workpiece fixture.
- ABB PLC, expansion module, and HMI.
- DSQC 1030 board for PLC and robot communication.

1.5 Outline

Chapter 1: Introduction.

- Introducing the paper, stating the reason on why choosing this topic.
- Objectives, research content, limitations, and project layout.
- Machine specifications.

Chapter 2: Background study.

- Background study and researches used to issue the problem, including communications, software and hardware.

Chapter 3: programming robot's movement.

- Overview of the system's operating procedure, flowchart.
- Programming robot based on the it's procedure.

Chapter 4: Evaluating and optimizing

- Monitor the operating process, identify any errors that occur, and find solutions to address them.

VGU
Vietnamese-German University

Chapter 5: Conclusion and future work

- Showing the result, and future aims of the project.

Chapter 2. BACKGROUND STUDY

2.1 Application of technology

2.1.1 Brief history of automation

Automation - the adaption of technology to perform tasks that were previously carried out by humans. There is no precise point in time when this concept was formed, as in fact it is a gradual process of development rather than a sudden event. Tracing back through history to even before the Common Era, automation has been employed when humans used simple tools and machines like the wheel, pulley, and lever to perform repetitive tasks, These are early examples of mechanical automation. Until the 18th century, the Industrial Revolution marked the advent of water - and steam - powered machines, automating many manufacturing processes. Notable examples include James Hargreaves's spinning jenny and Thomas Newcomen's steam engine. Moving on to the next century, the development of electricity and the internal combustion engine further fueled the advancement of automation. The first assembly lines were introduced, leading to mass production. Moving on to the next century, the development of electricity and the internal combustion engine further fueled the advancement of automation. This very development has enabled the formation of assembly lines and mass production. The 20th century witnessed the advent of computers and electronic devices, ushering in the development of more sophisticated automation systems. In 1968, the first Programmable Logic Controllers (PLCs) were designed and used at General Motors (GM) and since then, they have become an indispensable technology in controlling automation lines. And towards the final decades of the 20th century, robots began to be widely used in manufacturing, they along with the development of the internet, AI technology, and IoT, they have opened up a new era for the world, shaping the Fourth Industrial Revolution and driving the development of intelligent automation.

Vietnamese-German University

Automation has made a profound impact on society, the economy and the world. First and foremost, it helps to improve production efficiency, reduce costs, and enhance product quality. While automation can effectively replace some human-performed tasks, it also has the potential to generate new jobs in the fields of design, programming, and operation of automated systems. Given the current pace of development in the era of Industry 4.0, automation is expected to continue expanding in the coming years, driven by the emergence of new technologies such as AI, IoT, and quantum computing. This will lead to even more profound optimistic changes in our world.

2.1.2 Brief overview and development of Robots.

Robots have been one of humanity's greatest inventions for nearly 100 years, continuously evolving to become smarter, more useful, and safer. With advancements in science and technology, countries and industries worldwide are racing to innovate and manufacture the most advanced robots for every aspect of life and production. Nowadays, robotic arms are increasingly integrating advanced technologies such as artificial intelligence, machine learning, and internet connectivity. And in the future, there will be strong developments in automation capabilities, intelligent interaction, and close collaboration between humans and robots. The Development trends in the robotics industry include:

- Integration of AI and Machine Learning: With the advancement of Industry 4.0, robotic arms increasingly utilize artificial intelligence to automate decision-making processes and learn from the working environment.
- Collaborative Robots (Cobots): The emergence of collaborative robots capable of working alongside humans safely and efficiently.
- Wide Applications in Healthcare and Services: Robotic arms are rapidly developing in fields such as surgery, healthcare, and customer service.
- Exploration of Aerospace Technology: Applications include exploration and sampling on planets with conditions unsuitable for human presence, such as Mars. They also replace humans in industries dealing with hazardous waste and chemicals.

The first industrial robot was created in the years of 1950s when George Devol and Joseph Engelberger's company, Unimation, developed the first robotic arm, known as "Unimate." Unimate was utilized in automobile manufacturing processes. Unimate stands out as one of the first industrial robots used in manufacturing and was deployed at General Motors' (GM) automobile plant in the 1960s.[1] This marked a significant milestone in the history of industrial automation and robotics. It was an industrial robot capable of performing repetitive tasks in the manufacturing environment. Unimate marked a pivotal moment in the technological advancement of robotics in industrial production. Specifically, Unimate was used for tasks such as welding, painting, and handling components in the automotive assembly process. The integration of robots in manufacturing enhanced productivity, standardized product quality, and reduced risks for workers in hazardous tasks. Unimate had a profound impact on the manufacturing industry, paving

the way for the widespread development and use of industrial robots in various applications. The 1960s were a crucial period in the history of industrial robot development, as they began to appear extensively in various industries, including electronics and chemical processing. Although still relatively rudimentary, this marked a breakthrough in the application of automated technology in manufacturing. And since then, an increasing number of robots as well as software and principles have been developed to serve and assist humans, such as: Shakey - The first mobile robot with the ability to perceive and reason about its surrounding environment [2], The collaborative production robot (coop-robot) capable of working alongside humans [3],… and many more. And with the advent of the Fourth Industrial Revolution, robots integrated with artificial intelligence can adapt to situations independently without the need for human assistance. Additionally, there are many robots integrated with the Internet of Things (IoT) that can easily acquire information and be remotely controlled via the internet.

2.1.3 Brief development of ABB robotics.

ABB (Asea Brown Boveri) is a global leader in technology, producing advanced products and solutions in automation, electrical systems, and industrial and energy sectors. ABB's growth and development have been an exciting and challenging journey. ABB began in 1988 when two electrical companies, ASEA from Sweden and Brown, Boveri & Cie from Switzerland, joined together. This made a big new technology company based in Zurich, Switzerland. Since then, ABB has grown fast and is now one of the world's top tech companies. Although ABB didn't originally focus on robotics but in order to keep up with the changes, ABB Robotics was established**.**

ABB Robotics has undergone a successful and innovative development process over more than 40 years. Starting from the early steps in the 1970s, focusing on developing industrial robots for the automotive and ceramics industries, to the emergence of cobots in the 2000s, ABB Robotics has continuously adapted to market trends and demands. In the new decade, especially from the 2010s onwards, ABB Robotics has focused on technology innovation, particularly in the fields of artificial intelligence and machine learning. ABB's intelligent robot systems have been developed with the ability to learn and interact with the surrounding environment, helping to optimize performance and flexibility in manufacturing. ABB Robotics' commitment to continuous innovation and improvement continues to play a crucial role in the development of the automation industry, delivering value to customers and the community.

2.2 Comunication technology.

2.2.1 Industrial Ethernet and PROFINET

- **Industrial Ethernet**

Industrial Ethernet involves the use of standard Ethernet technology for communication within industrial automation environments. It provides a flexible, scalable, and cost-effective networking platform for connecting control devices, sensors, drives, and other equipment. This communication protocol carries several advantages such as:

- **Flexibility**: Ethernet can support multiple industrial communication protocols, such as Modbus TCP/IP, EtherCAT, PROFINET, and EtherNet/IP.
- **Scalability**: Ethernet can easily scale to meet the needs of increasingly complex automation systems.
- **Cost-effectiveness**: Ethernet utilizes standard network components that are affordable and easy to install.
- **Performance**: Ethernet can provide high-speed data transmission to meet the demands of stringent automation applications.
- **Reliability**: Ethernet is a proven networking technology with high reliability.

VGU
Vietnamese-German University

It can be seen that nowadays, there are plenty of choices regarding communication protocols. However, selecting the appropriate protocol to use involves various factors such as performance requirements, network structure, compatibility, and cost. In general, Industrial Ethernet is a robust and flexible network solution for industrial automation applications. With many advantages such as flexibility, scalability, cost-effectiveness, performance, and reliability, Industrial Ethernet is increasingly being widely used in various industries.

- **PROFINET**

One of the most famous communication protocols: PROFINET, short for “Process Field Network”, is an industrial technical standard for transmitting data over industrial Ethernet. It is designed to collect data from and control devices in industrial systems, with particular strength in providing real-time data with tight timing constraints down to 1ms. In simple terms, PROFINET is a computer network used to connect automation devices together. It enables these devices to communicate with each other efficiently and reliably, helping improve the performance and productivity of automation systems. [6] PROFINET iself wields several advantages compared to other industrial networks such as:

- **Performance**: PROFINET can transmit data at high speeds, reducing latency and improving response times.
- **Flexibility**: PROFINET can be used with various network topologies, including linear bus, ring, and star topology.
- **Scalability**: PROFINET can easily scale to meet the needs of increasingly large automation systems.
- **Standardization**: PROFINET is a standards-based network, supported by various equipment providers.
- Security: PROFINET offers advanced security features to protect data from unauthorized access.

VGU
Vietnamese-German University

PROFINET operates based on the principle of providing deterministic and high-speed communication in industrial automation environments. To meet this requirement, PROFINET utilizes various communication channels such as TCP/IP, Real-Time (RT), Isochronous Real-Time (IRT), and Time-Sensitive Networking (TSN). For tasks that do not require real-time determinism, PROFINET uses TCP/IP or UDP/IP communication. However, to ensure determinism and high-speed for time-critical applications, PROFINET employs RT communication. This communication directly transmits data from Ethernet Layer 2 to PROFINET through Layer 7, bypassing the TCP/IP layers to avoid latency. [7]

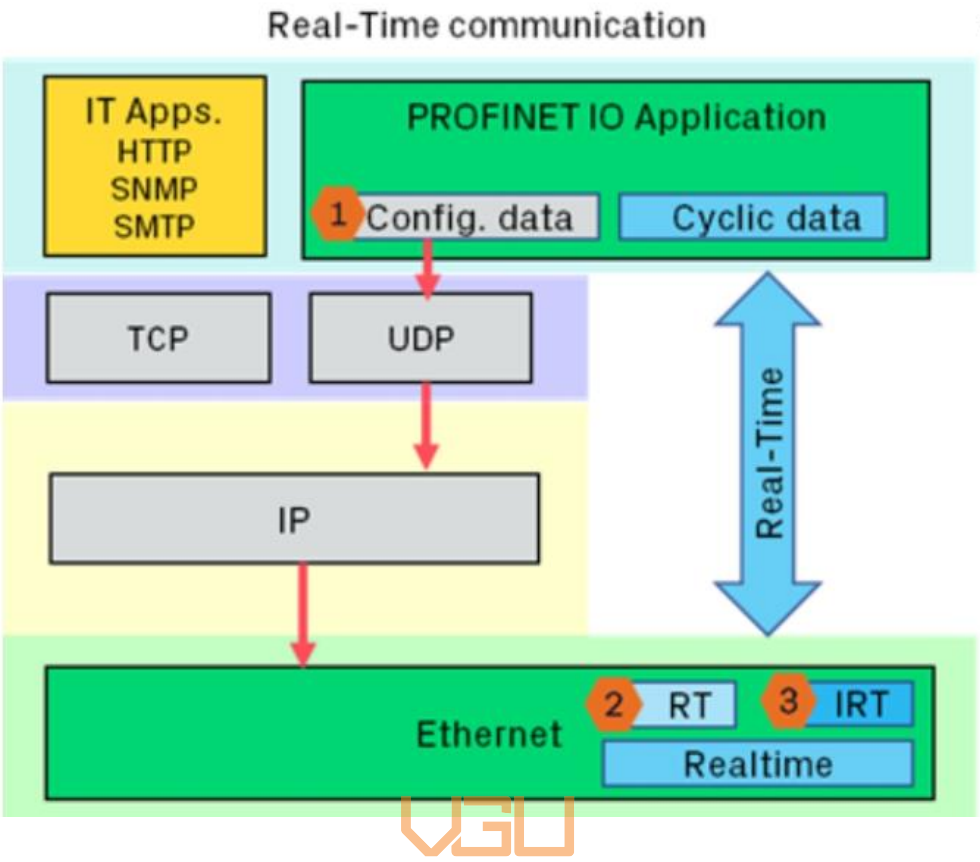


Figure 2.1: Operating principle of the PROFINET protocol. [7]

PROFINET is widely used across various industries, including automotive manufacturing, machinery production, shipbuilding, food processing…and many more. In general, PROFINET is a robust and flexible industrial network that can be used for various purposes. It's an excellent choice for automation applications that demand high performance, reliability, and security.

- **Similarities and differences between.**

Regarding similarities, both IE and PROFINET uses Ethernet technology. This brings them several common benefits. Firstly, since Ethernet utilizes standard network component, they are affordable and easy to install. Secondly, Ethernet can provide high-speed data transmission to meet the demanding requirements of automation applications and lastly, Ethernet is a proven networking technology with high reliability. And alongside their similarities, IE and PROFINET also have their own distinguishing features, listed at the table below:

Indsutrial Ethernet	PROFINET
The popular network communication method used for building worldwide networks.	The Industrial Ethernet solution is developed by PROFIBUS & PROFINET International (PI)
Used to connect nodes within a LAN network.	Used for exchanging data between devices and controllers.
Resides above the physical and data link layers.	Resides on the application layer of the ISO/OSI model as it's an application.
The transmission speed is slower compared to Profinet	Operates at a very high transmission speed compared to Ethernet.

Table: Differences between IE and PROFINET. [12]

2.2.2 Ladder programming language

Ladder Logic (also known as ladder diagram or LD/LAD) is a programming language used to program PLCs (Programmable Logic Controllers). It is a graphical programming language for PLCs that depicts logical operations with symbolic representations. Ladder logic is created from logic ladder rungs, forming something resembling a ladder - hence the name "Ladder Logic" or "Ladder Diagram"." It is popular because it's easy to understand, visual, and simulates how mechanical relays work, helping programmers easily imagine and write control programs for automation systems.

The rungs in a ladder diagram represent the supply wires of the relay logic circuit. There's a positive voltage supply rail on the left side and a zero voltage supply rail on the right side. In a ladder diagram, the logic flow is from the left rail to the right rail. The rungs in a ladder diagram represent the connections between the components of a relay control circuit. In the ladder diagram, symbols are used to represent the relay components. The symbols are placed in the rungs to form a network of logical expressions. When implementing ladder logic programming in a PLC, there are seven basic parts of the ladder diagram to know. They are the rung, ladder, input, output, logical expression, address/variable name symbols, and comments. Some of these elements are essential, and others are supplementary.

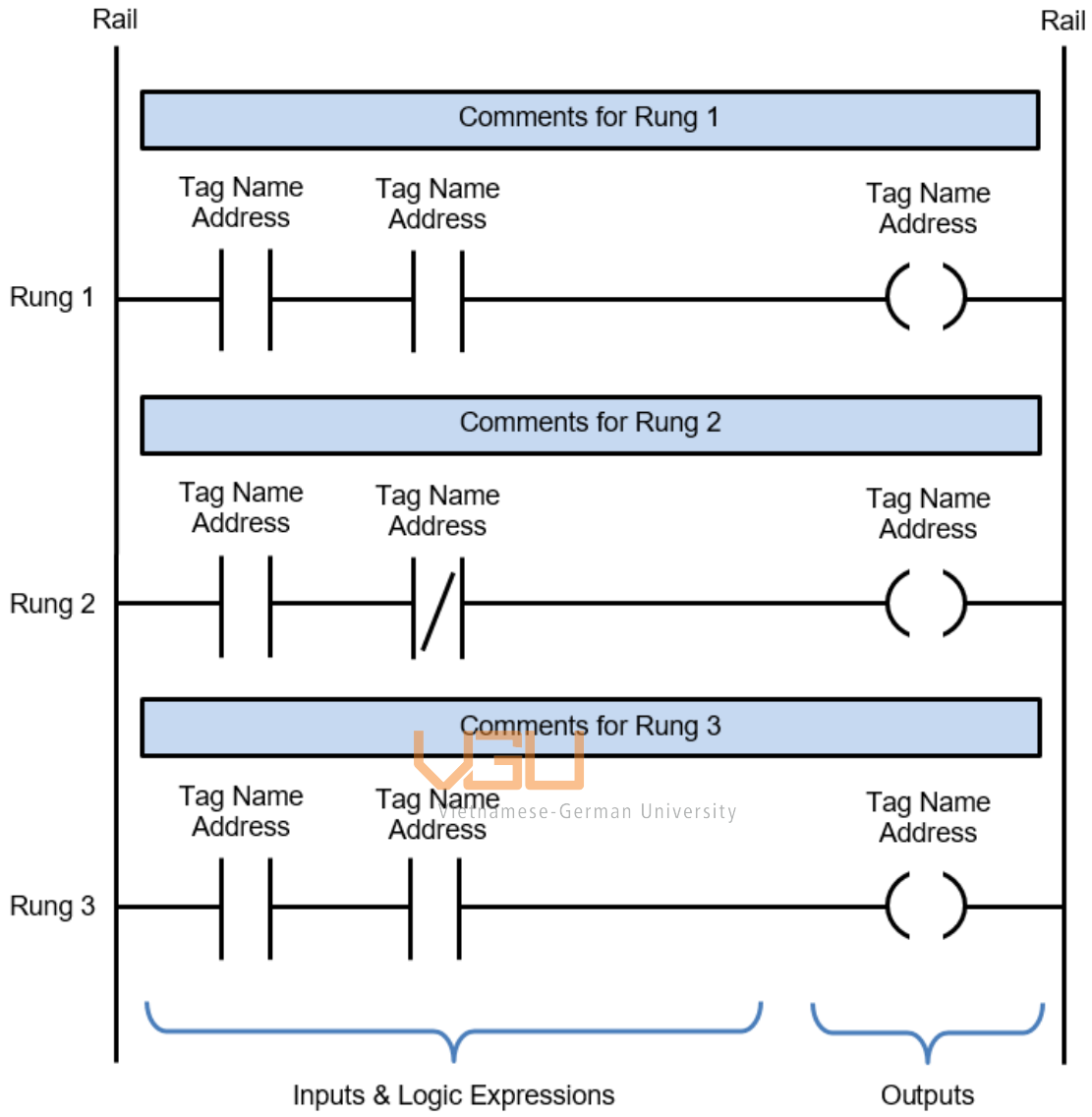


Figure 2.2: The basic structure of a PLC program. [11]

- **Rungs**: The rungs are drawn as horizontal lines connecting the ladder rungs with logical expressions. In relay circuits, they represent the wires connecting the power source to the switching components (push buttons, switches, etc.) and relays.
- **Inputs**: These are external control actions such as a pressed push button or an activated limit switch. The actual inputs are hardwired to the PLC terminals and are represented in the ladder diagram by normally open (NO) or normally closed (NC) contact symbols.

- **Outputs**: These are external devices turned on and off such as electric motors or solenoid valves. The outputs are also hardwired to the PLC terminals and are represented in the ladder diagram by coil symbols of relay.
- **Logical Expressions**: Used in combination with inputs and outputs to form desired control operations.
- **Address Symbols & Variable Names**: Address symbols describe the structure, defining addresses in the ladder logic memory for PLC inputs, outputs. Variable names are descriptions for allocated addresses.
- **Comments**: Typically displayed at the beginning of each rung and used to describe the logical expressions and control operations the rung or group of rungs are performing. Understanding the ladder diagram becomes much easier by using comments.

Symbols (Graphic Symbols) & Meanings in Ladder Logic:

- Rung inputs on the left (contacts):

-[ ]- **Normally open (NO) contact**: Inittialy in the open state when there is no signal or activating condition, meaning no current flows through it. When a signal or activating condition is applied, it switches to the closed state, allowing current to flow through it.

VGU
Vietnamese-German University

-[/]- **Normally closed (NC) contact**: Inttially in the close state when there is no signal or activating condition, meaning current can flow through it. When a signal or activating condition is applied, it switches to the open state, cutting off the current.

- Rung outputs on the right (coils):

-( )- **Normally inactive**: Intially inactive, active when current flows through its rung.

-(/)- **Normaly active**: intitally active, inactive when current flows through its rung.

The main principle of ladder language operation is based on standard logic circuits such as AND, OR, NOT, and their variations such as AND/OR, OR/AND in combine with the rung inputs and outputs. Signals from the inputs are processed through these logic circuits to generate control signals at the corresponding outputs. The structure of a ladder program typically follows the 'scan cycle' principle in automatic control, where ladder rungs are executed in sequence from top to

bottom, and from left to right. Each scan cycle, input signals are read, and logic circuits are processed to determine the control signals at the corresponding outputs.

A few logic operations that can be represented by ladder language include:

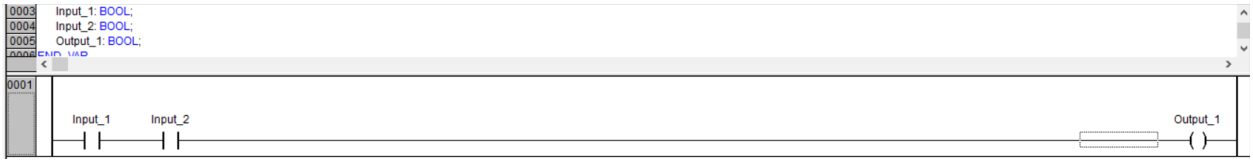


Figure 2.3: AND – logic operation.

- **The AND logic**: Presented by Input_1 and Input_2 placed in series on the circuit. In this case, if only one of these contacts is set active, there will still be no current flowing to the Output_1 or FALSE state. The Output_1 coil will only be activated if both Input_1 and Input_2 is set active or TRUE state.

VGU
Vietnamese-German University

Figure 2.4: OR – logic operation.

- **The OR logic:** Presented by Input_1 and Input_2 placed in parallel on the circuit. In this case, Just one of the two inputs needs to be set active for current to flow through the output or TRUE state.

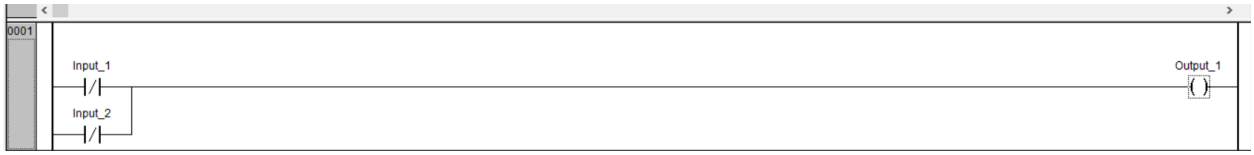


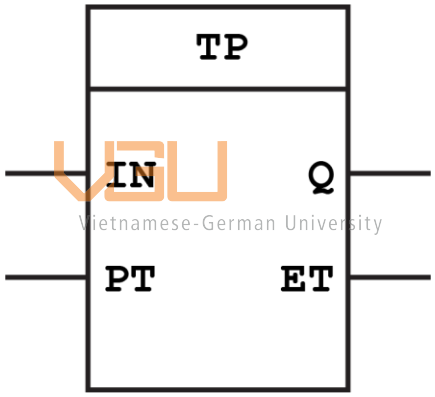
Figure 2.5: NAND – logic operation.

- **The NAND logic:** Also known as the NOT AND gate, the NAND logic gate is the opposite of the AND logic is presented by the two NC Inputs. In this case, the Output will always be active or TRUE even if both inputs are inactive or at FALSE state.

Figure 2.6: NOR – logic operation.

- **The NOR logic:** Also known as the NOT OR logic gate, the NOR gate is the opposite of the OR gate. In this logic, if one of the inputs is set active or TRUE state, the output will be inactive or FALSE state.

One of the official and widely used PLC programming languages is the Function Block Diagram (FBD). It is a simple and graphical way to program any functions together in a PLC program, used to describe the function between input variables and output variables. A function is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. Some of the commonly used function blocks such as: TIMER, COMPARATOR, COUNTER….



VGU
Vietnamese-German University

Figure 2.7: Timer Pulse (TP) function block.

Overall, Ladder is an efficient programming language for simple to moderate automation systems. It is easy to learn, easy to use, and widely supported. However, it has limitations in handling complex programs and lacks flexibility compared to other languages.

2.2.3 RAPID

In the world of ABB robotics, the RAPID language serves as the primary programming tool. It bears a resemblance to typical ST (Structured Text) languages and shares close similarities with C-style programming languages. RAPID (Robot Application Programming Interface) is a high-level programming language used to control ABB industrial robots. Introduced alongside the S4 control system in 1994 by ABB, RAPID replaced the previously used ARLA programming

language. The RAPID language contains embedded functionalities that the robot can use. These embedded functions allow the robot to move to various locations (for example, "MoveL" is a linear move to a taught position). Some functions compute mathematical instructions or determine the robot's sensitivity to external influences. By utilizing the pre-existing functions within the RAPID library alongside controlling input and output signals, users can program and control the movement of various ABB robot models, ranging from industrial robots to cobots, delta robots, and SCARA robots.

Due to its many similarities with the C language, RAPID is quite user-friendly, making it accessible even to beginners who can quickly familiarize themselves with it. Moreover, with its diverse library, RAPID offers users plenty of tools to address applications ranging from simple to complex, thereby saving cycle time and enhancing robot operational productivity. Furthermore, with a large user community, abundant instructional materials, and supportive tools available, users can easily seek assistance when needed.

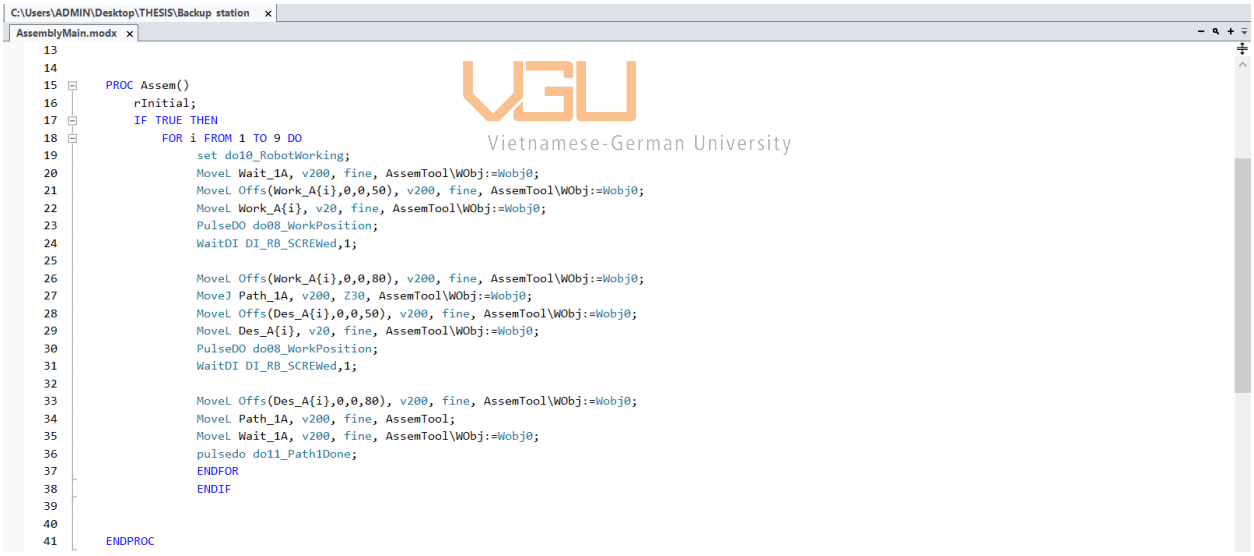


Figure 2.8: Basic programming language of RAPID.

2.3 Main hardware

2.3.1 IRB1100-4/0.475

- **Overview**

The IRB 1100 is one of the latest generation 6-axis industrial robots from ABB Robotics. With a compact design and a payload of up to 4kg, it is suitable for versatile applications and can communicate with various external systems. This robot arm can be installed in three different mounting: floor mounted, wall mounted, and it even suspended and there are no limitations on the angle of each mounting position. For each different mounting position, the robot's components will experience varying loads during operation, including forces in the XY direction, forces in the Z direction, as well as bending torque in any direction within the XY plane and the Z plane. With its emergency stop feature, the IRB 1100 robot will automatically halt its operation when subjected to forces exceeding the maximum allowable limit, thereby ensuring the safety of users and prolonging the robot's lifespan. The temperature range for the robot to operate over extended periods without affecting its lifespan is from 5 to 45 degrees Celsius under the condition where the maximum ambient humidity does not exceed 95%. And since the IRB1100 used for the screw-driving system is floor mounted, only the corresponding numbers are considered.

VGU
Vietnamese-German University

Floor mounted

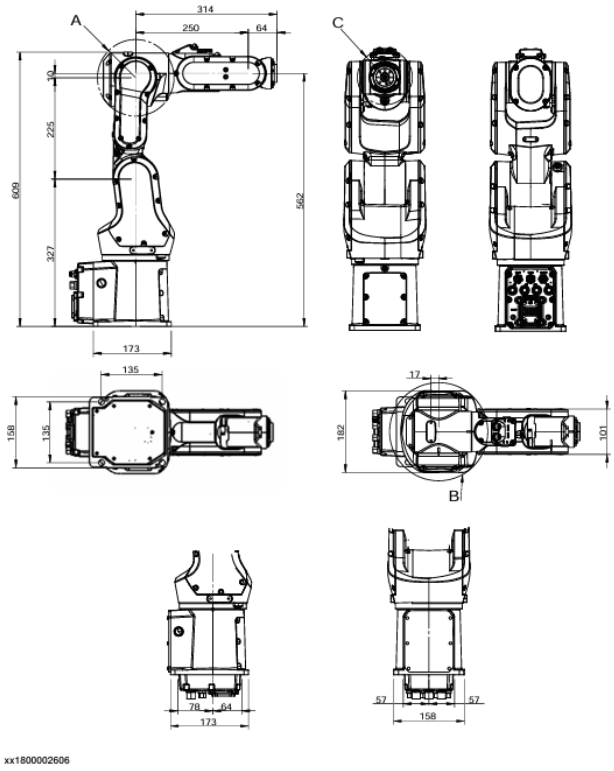
Force	Endurance load (in operation)	Maximum load (emergency stop)
Force xy	±420 N	±710N
Force z	+210 ±380 N	+210 ±510 N
Torque xy	±180 Nm	±330 Nm
Torque z	±90 Nm	±140 Nm

Figure 2.9: Endurance and maximum load for floor mounted configurations.[4]

- **Dimensions and working range.**

The figures below show the dimensions and the working range of the IRB1100 version 0.475. Numbers are in milimeters.

- Dimension:



- Working range

VGU
Vietnamese-German University

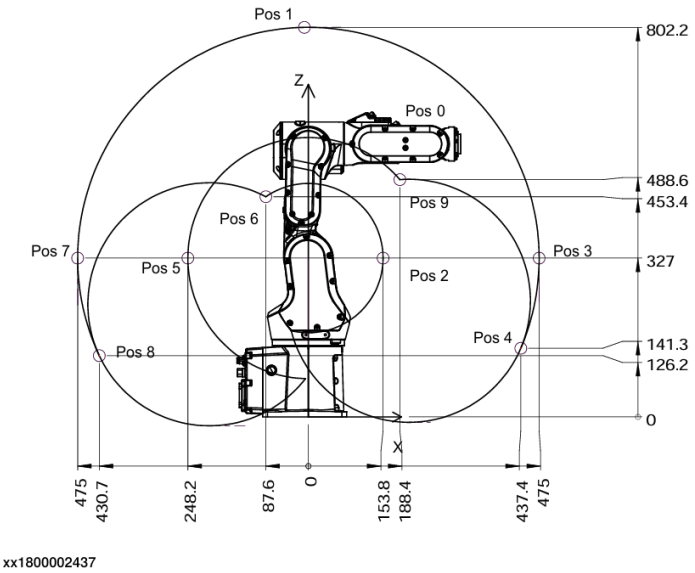


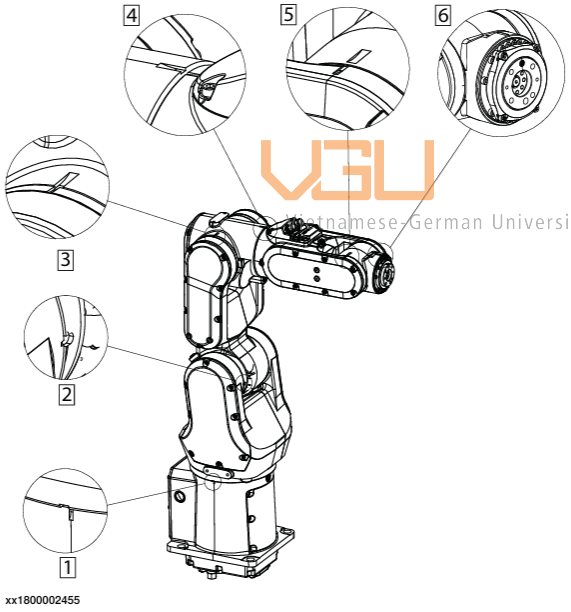
Figure 2.10: Dimension and working range of IRB1100.[4]

- **Calibration.**

Calibration is a step to standardize the axes of ABB robots. The purpose of this is to establish the standard coordinate origin of the robot and bring the robot to the zero position, meaning that the rotation angle of all axes is 0 degrees. The robot cannot start operation until it is calibrated because at this point, the linear movement, orientation features, as well as teaching new positions are fullly disabled. Only when calibrated, those features can be used again. The robot will only be able to move along each axis independently when not calibrated. This is to help users move each axis to the marked calibration positions called the “Synchronization marks”.

.

Synchronization marks, IRB 1100



VGU
Vietnamese-German University

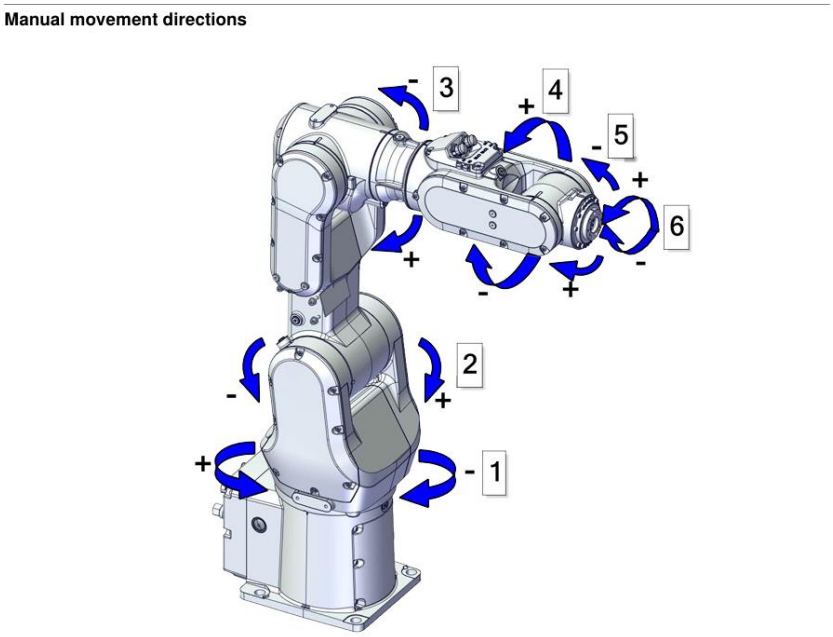


Figure 2.11: Synchronization marks and each axis's movement directions of IRB1100.[4]

VGU
Vietnamese-German University

2.3.2 Omnicore E10

- **Overview**

Omnicore is an advanced line of controllers by ABB specifically designed to control various types of robots in a variety of applications. This product line offers a range of advanced features and technologies to optimize performance and flexibility in the manufacturing process. And being an advanced product line, the Omnicore controller includes some standout features compared to previous models, a few of them includes:

- **Safe Collaboration Capability**: This allows robots and humans to work together without the need for traditional safety measures like safety fences.
- **Easy Integration**: Omnicore can easily connect with other systems and devices through standard networking.
- **Efficient Data and Program Management**: This controller efficiently receives and stores production programs, optimizing programming and job changes flexibly.
- **Integrated Artificial Intelligence**: By integrating AI, the robot can learn and optimize performance over time.

Omnicore is indeed the brain of the robot, as it not only performs basic functions like starting or shutting down the robot, controlling the speed with motor,… but also serves as the storage location for programs uploaded by users and many more. With its advanced features and technologies, it plays a crucial role in enhancing productivity and flexibility in modern manufacturing environments.

Figure 2.12: ABB's Omnicore-series controllers.

- **Omnicore E10**

Omnicore E10 is one of four versions of this product line, and it is also the controller used to operate the IRB1100 of the scew-driving station. With it's compact design with a width of 445mm, length of 340mm, and height of 100mm, this version can be easily transported and installed, without taking up too much space in the station's layout. All connection ports, including the HMI port, WAN port, LAN port, ports for additional I/O modules, as well as the connection for the FlexPendant teaching device, are located close to each other on the same side, facilitating convenient connection and usage. Similar to the IRB1100, the ideal temperature range for Omnicore operation is from 5 to 45 degrees Celsius. The connection of the controller is as follows:

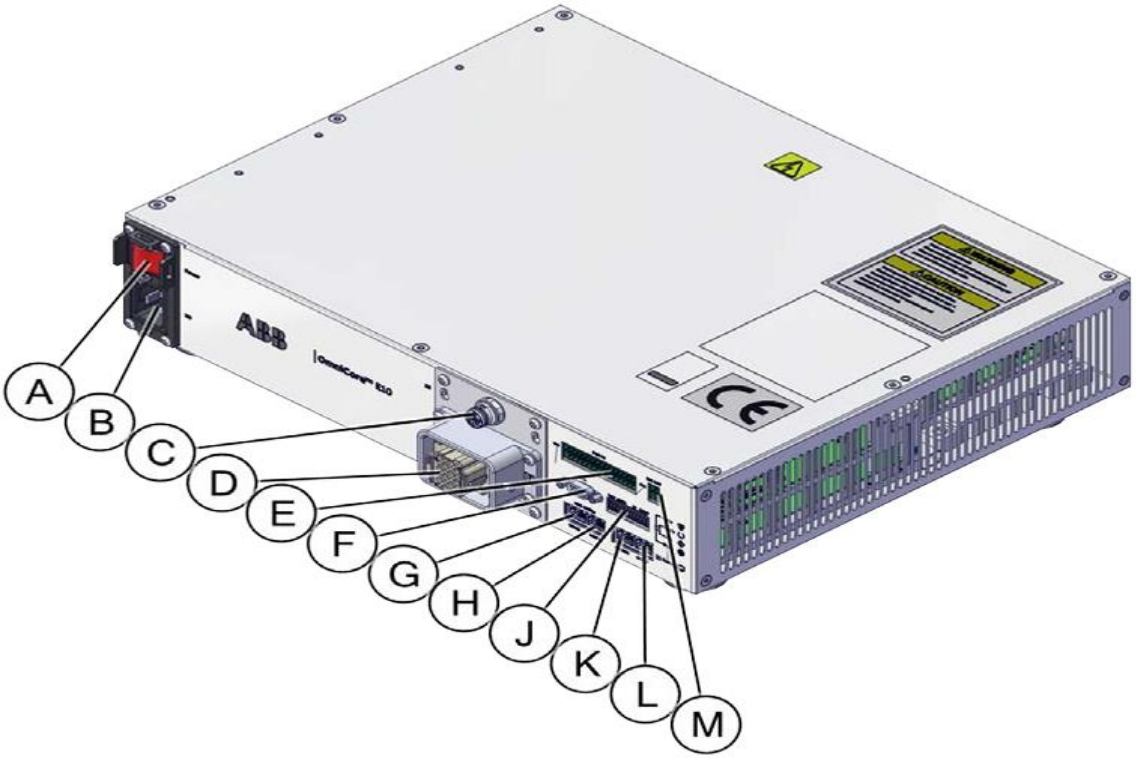


Figure 2.13: Omnicore E10 and connection ports. [5]

VGU
Vietnamese-German University

Node	Description
A	Power inlet switch
B	Power inlet connector
C	Manipulator signal connector (SMB)
D	Motor connector
E	I/O interface
F	FlexPendant adaptor connector (HMI)
G, H	WAN1, WAN2 port
J	Customer safety interface
K	Device port
L	Management port
M	External 24 V power inlet connector

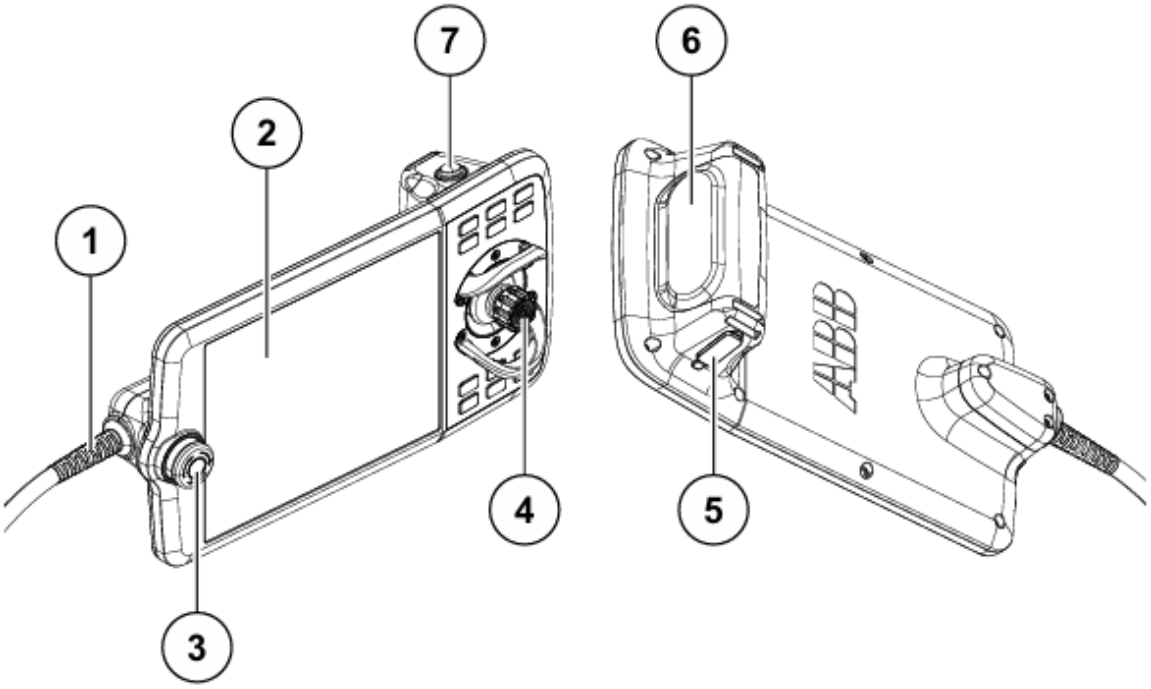
Table: Connection ports of Omnicore E10. [5]

Despite the numerous connectivity ports, once installed by ABB Robotics, our primary concern revolves around the location of the power node 'A' , the management port ‘L’ from which we will load programs into the robot and the device port ‘K’ that will be used to connect the external board DSQC 1030.

2.3.3 FlexPendant

The FlexPendant is a handheld device directly connected to the Omnicore controller. It is the device that users will directly interact with to control ABB robots, run and modify robot’s programs, and many more. The FlexPendant is made to work continuously in a challenging industrial environment. Its touch screen is splash-proof, water-and oil-resistant, and simple to clean. Consisting both the hardware and software, the FlexPendant is a complete computer by itself. It is connected to the Omnicore controller by an integrated cable and connector.

Figure 2.14: ABB’s Omnicore FlexPendant



1	Connector
2	Touch screen
3	Emergency stop button
4	Joystick
5	USB port and reset button
6	Three-position enabling device
7	Thumb button

VGU
Vietnamese-German University

Figure 2.15: Main parts of FlexPendant. [5]

The joystick (4) will be used to move the robot. This action is call jogging the robot. There are several different jogging methods such as jogging along each axis of the robot, linear jogging, and orientation jogging. However, initially, only jogging along the robot's axis is possible. Linear and orientation jogging can only be used once the robot has been calibrated. The reset button (5) is pressed if the FlexPendant freezed during usage. Resetting the FlexPendant via the reset button does not reset the controller's system.

Every button in the hard button panel on the right side of the Flexpendant has its own specific functions. Among the total of 12 buttons, there are 4 buttons that currently have no function assigned to them. Users can assign functions to them according to their own needs.

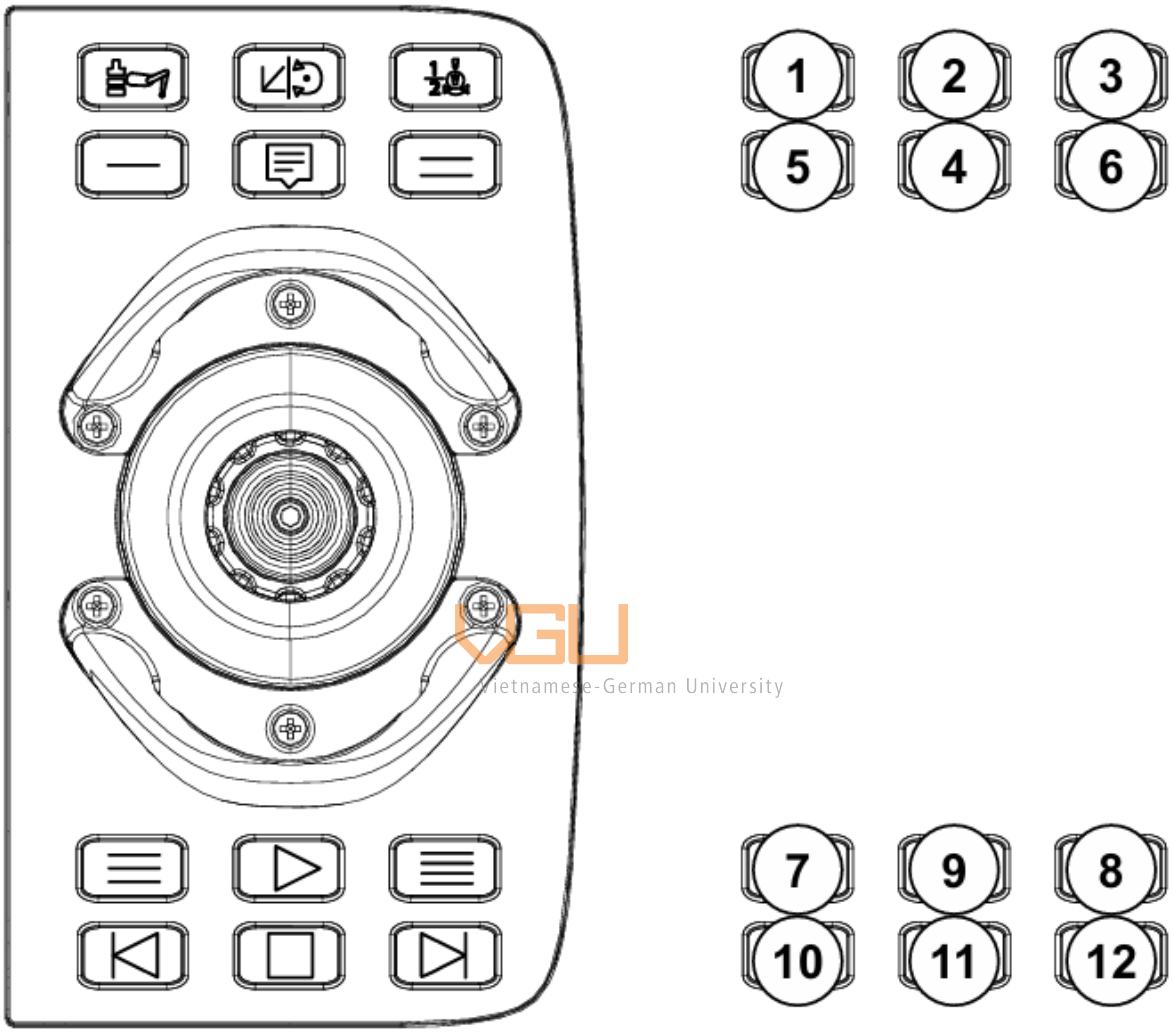


Figure 2.16: positions of the hard buttons on the Flexpendant. [5]

For the upper set buttons, if there are more than 1 mechanical unit being controlled by Omnicore, button number (1) will be used to select between those units. To toggle between jogging linear and orient, we use button (2). And to toggle between jogging the axis 1-3 or axis 4-6, button (3) will be used. The button number (4) is to show the operator's messages. For the lower set buttons, pressing button (9) will execute the program that is curently loaded in the controller's system, and

button (11) to stop it. Button (10) and (12) will only execute one instruction backward and forward from the current program's pointer. The remaining four buttons (5,6,7,8) are the user-defined buttons.

The three-position enabling device (6) is a button that the user needs to press and hold to start the robot's motor. Only then can the user begin jogging. This device has two different pressure modes. If the user applies moderate force to press it, they will enter the motor ON mode. However, if they press harder or release it completely, it will be in motor OFF mode. The figure below shows how users can hold the FlexPendant. Since the display screen can be rotate easily, user can both hold the FlexPendant on their left or right hand.

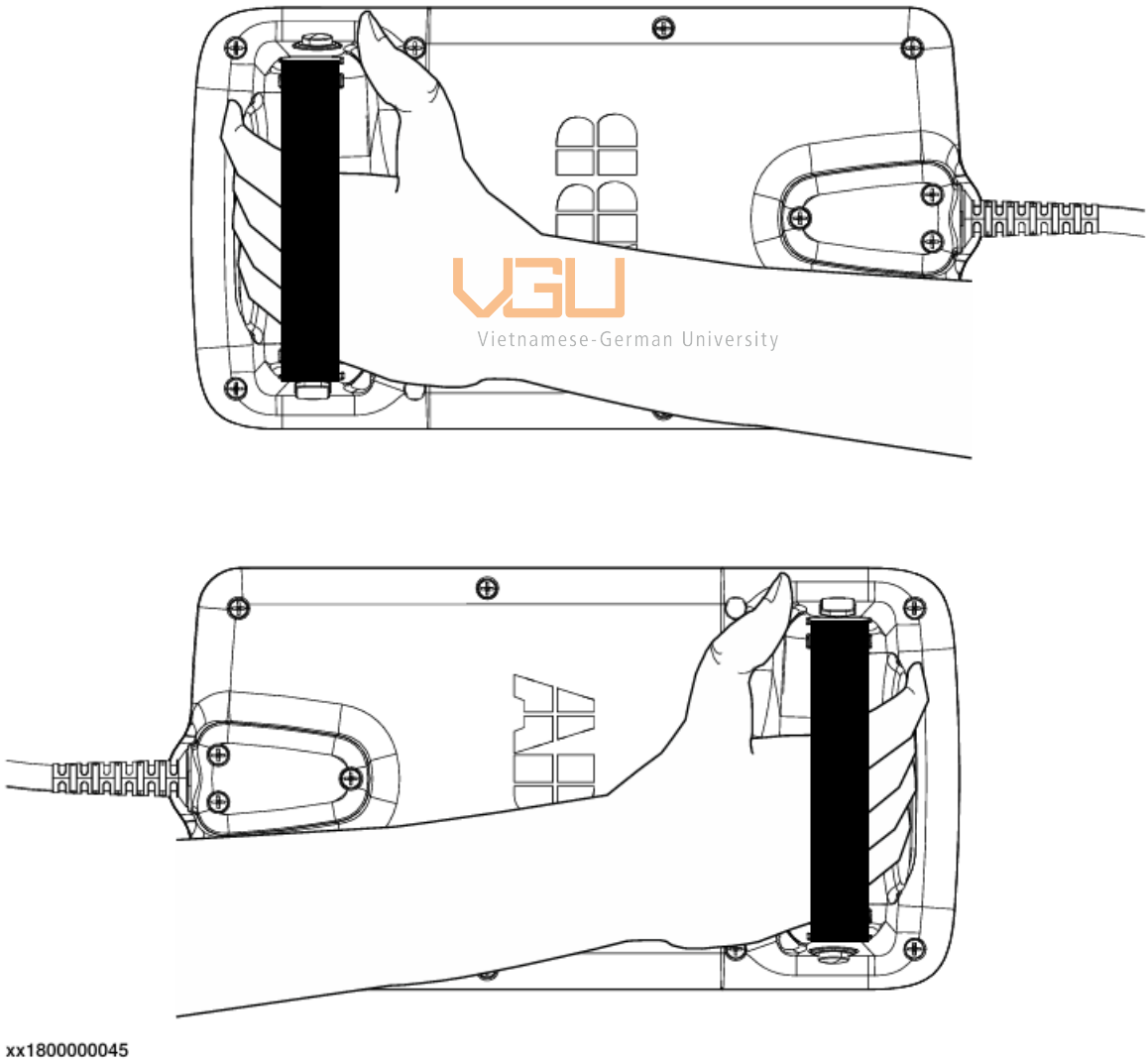


Figure 2.17: holding the Flexpendant. [5]

The touchscreen of the FlexPendant represents the icons in a very intuitive and understandable manner. With its sufficiently large design, users can comfortably perform their touchscreen operations without worrying about being obstructed.

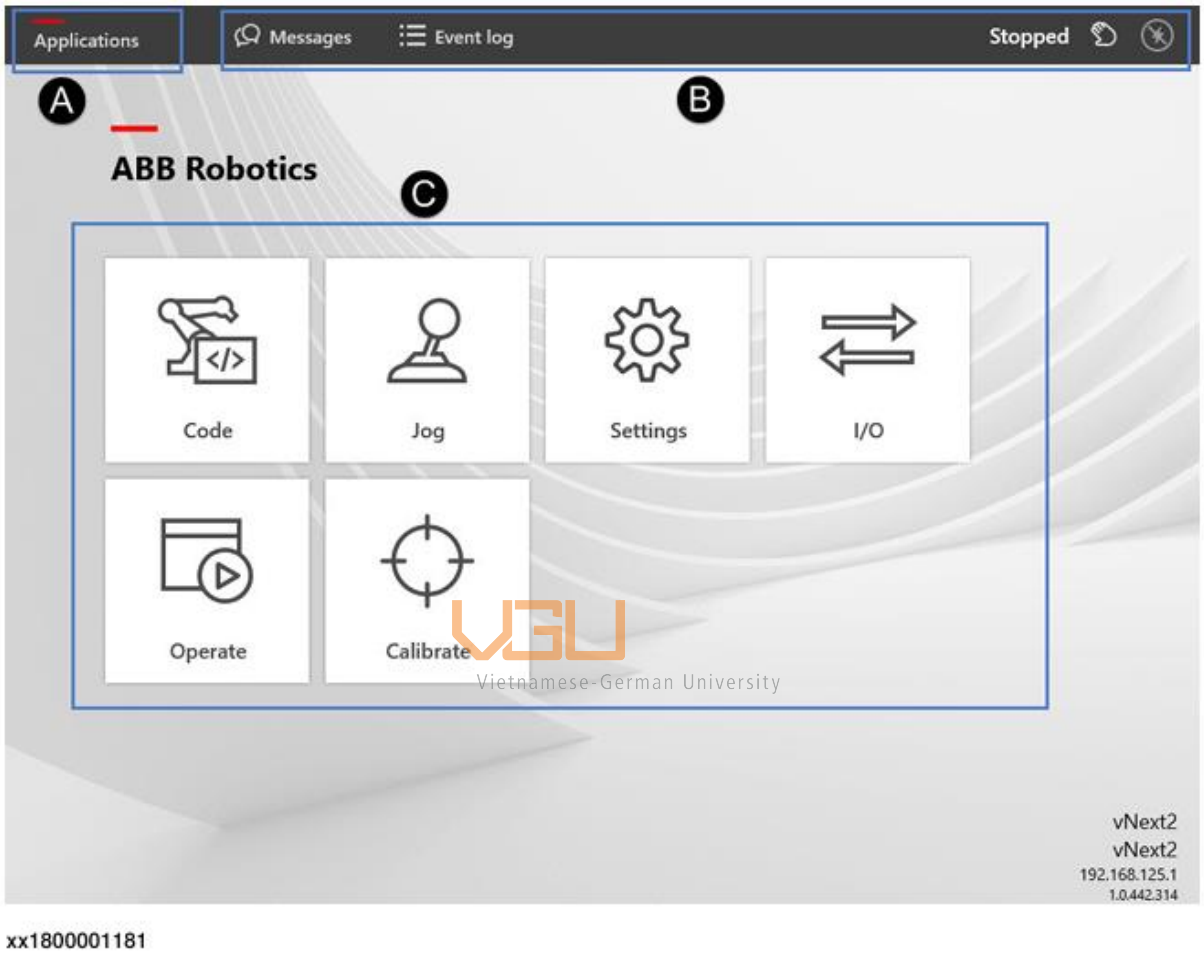


Figure 2.18: Main screen of FlexPendant. [5]

The main interface of the FlexPendant screen can be grouped into three main sections: A, B, and C as follows. Users can tap on Applications (A) at any time and at any window to return to this main screen that we will called Home screen. The status bar (B) will allows users to monitor the current robot's status, including the operator's message, event log, motor's status and more. The applications (C) that are required for operating the robot system are available in the Home Screen.

2.3.4 DSQC 1030

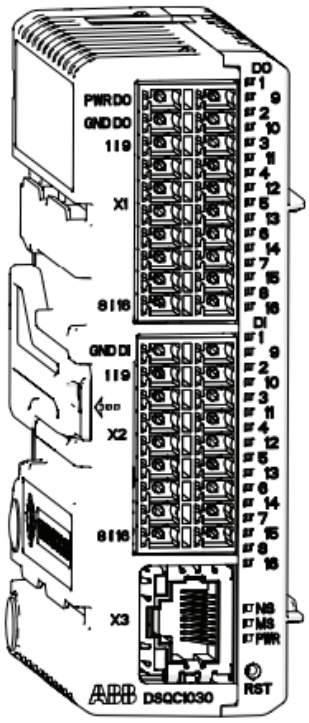
The DSQC 1030 is a basic digital IO module developed by ABB, one of the leading names in the automation industry. This module is primarily used for controlling and communicating with other automation devices and systems within a complex network. The device is a part ABB's Scalable I/O system, designed for industrial robots. This device provides digital inputs and outputs for industrial control systems. It is used in various automation applications, including industrial robots, assembly lines, and material handling systems. It boasts advantages such as a compact and sturdy design, easy installation and use, high anti-jamming level, and support for multiple communication protocols.

Techical data of the DSQC 1030:

- Number of I/O: 32. (16 input and 16 output)
- Type of signals: Digital. (TRUE/FALSE)
- Operating voltage: 24V DC.
- Current consumption: 5A
- Operating temperature: -20°C to +60°C

Figure 2.19: DSQC 1030

our station will utilize this DSQC 1030 module to provide I/O for communication between the robot and the PLC system. The location of connecting ports of the device is shown below:



xx1600002033

Connector	Description
X1	Digital outputs, process power
X2	Digital inputs
X3	EtherNet
X4	Logic power
X5	EtherNet

VGU Vietnamese German University

Figure 2.20: DSQC 1030's connection ports. [9]

2.3.5 PM583 - ETH

The PM583-ETH is a programmable logic controller (PLC) belonging to the AC500 series by ABB. It's designed to offer robust, flexible, and cost-effective automation solutions for various applications. It is designed to simplify automation tasks with its powerful 32-bit CPU and ample 1MB program memory. It supports common communication protocols like Ethernet, Modbus, PROFIBUS, and CAN, ensuring easy integration into various setups. You can easily expand its capabilities with extra I/O modules. Plus, its user-friendly programming interface, along with the

intuitive ABB Automation Builder software, makes it a straightforward choice for automation needs in different industries. Below are some technical specifications of this device:

Memory Size:	2048 kB
Memory Size User Data:	1024 kB
Memory Size User Program:	1024 kB
Memory Type User Data:	RAM
Controller Processing Time:	0.00005 ms
Number of Digital Configurable I/Os:	0
Degree of Protection:	IP20
Rated Voltage (U_r):	24 V DC
Supply Voltage:	20.4 ... 28.8 V DC
Ambient Air Temperature:	Operation 0 ... +60 °C Storage -40 ... +70 °C

VGU
Vietnamese-German University

Figure 2.21: Technical data of PLC PM583-ETH

2.3.6 Robot’s tool

In this station, the robot's tool head is a mechanical product designed and manufactured by ABB Robotics. The upper part of the tool head has a square piece angled relative to the rest to connect to the robot's axis 6 through the screw holes in the circular area. The screwing head is mounted on a bracket placed on a slider, allowing vertical adjustment to change the depth of the screwing head during screwing operations. The lower part is designed to resemble a duck’s beak, with a small angled hole to guide the screw down from the screw feeder to the screwdriver head. When the screwdriver’s head rotated and pressed down, the duck’s beak will be opened, allowing the screw to be twisted down on the workpiece.

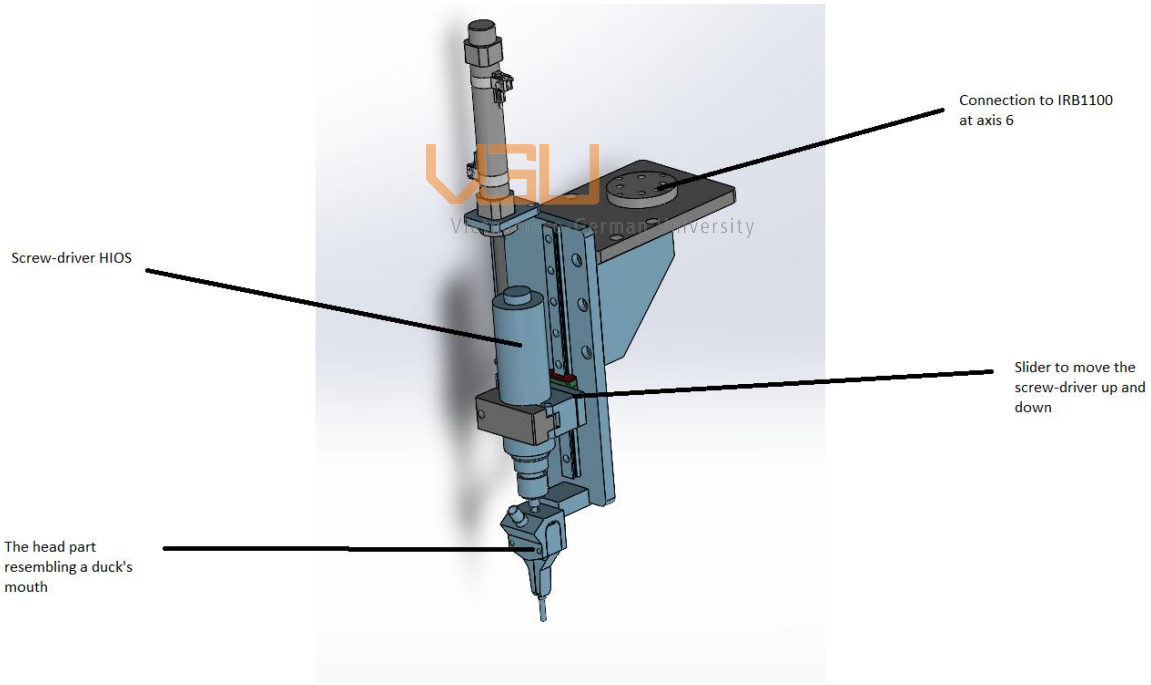
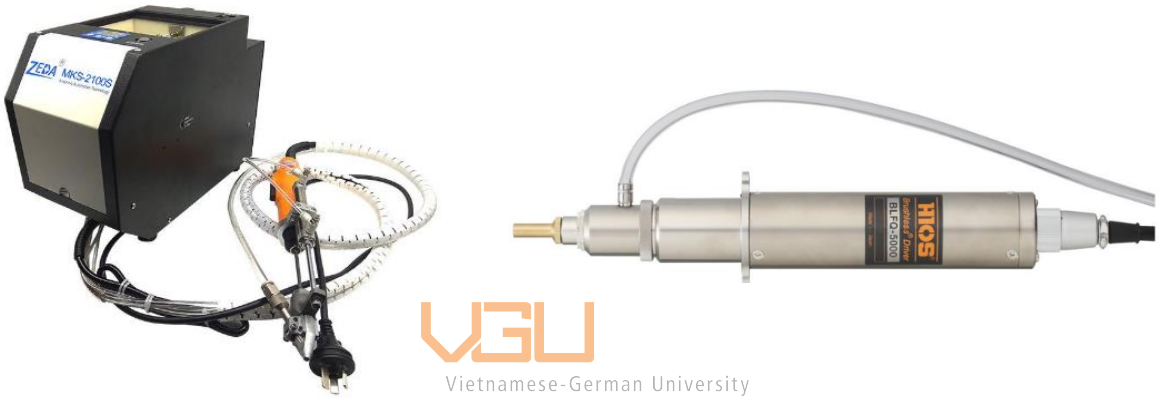


Figure 2.22: Tool’s head decription.

This tool is designed to be large and sturdy to ensure no shifting or vibration during operation and robot movement, thereby ensuring the station's efficiency. However, due to its large size, we must ensure the robot's path and angles of rotation to prevent the tool from colliding with other parts during operation.

2.3.7 Other devices and screwing principle.

In addition to the main hardware and software that we will primarily use to program and operate this screwing station, there are also electrical and mechanical devices surrounding it. These devices directly contribute to the screwing process: the Zeda MKS2100MV automatic screw feeder, and the Hios BLFQ-2000 electric screwdriver and air hoses to serve the purpose of supplying and screwing screws into the workpiece. While the Omron E3Z-T61A-L optical sensor, and the SY3120-5LZD-C6 solenoid valve will be ultilized by PLC to control the robot's movement.



VGU
Vietnamese-German University

Figure 2.23: The Zeda automatic screw feeder and the Hios screwdriver head.

The working principle of the screw-driving process is as follows: When the workpiece is secured and the robot moves to the screwing position, a signal is sent to the PLC, triggering another signal to the screw feeder to supply a screw. The screw is then conveyed from the feeder to the screwing tool head using air pressure. After the screw is in position, another signal will be sent back to the robot to move into work position, and lastly the electric screwdriver receives a signal to start pushing the screw down and screwing it into the workpiece. The idea is based on the Vacuum screwdriver with automatic feed system SEV of WEBER.

Chapter 3. PROGRAMMING ROBOT'S MOVEMENT

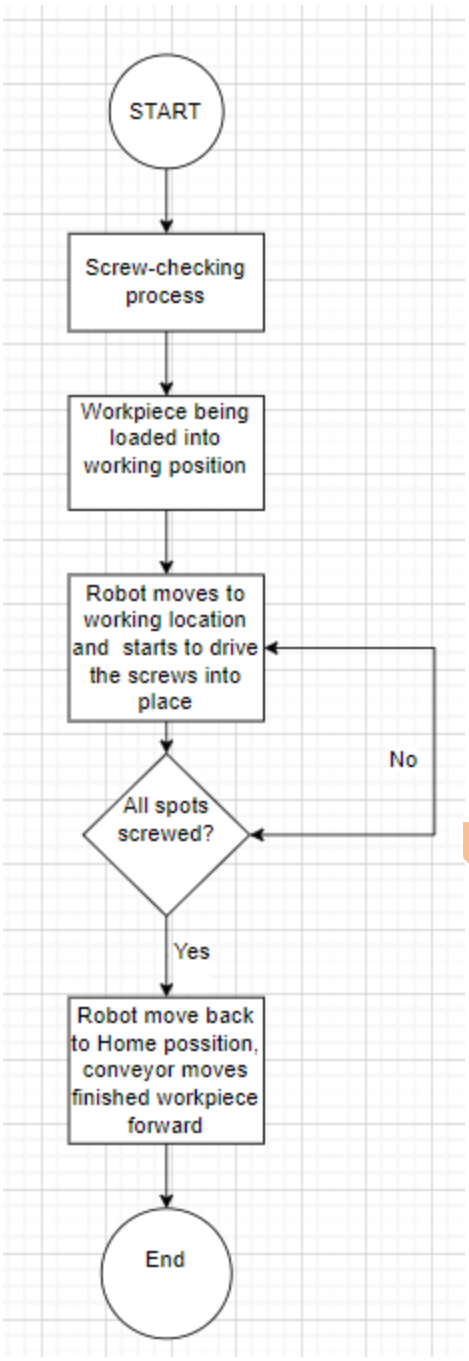
3.1 Operating principle

According to the proposal from ABB Robotics, the robot must meet the following requirements: the robot itself has to move to the Home position from any position when the user presses the start button to run the program. Then, the robot will execute a simple process of moving down and releasing any excess screws stuck in the tool (if any), then return to the home position. After that, the screwing process can begin.

The operation process of the screwing process is as follows: The purpose of the two conveyors with built-in cylinder, one for lifting and one for lowering the workpiece, is to move the workpiece in a closed loop to simulate the production line in the actual factory, where different workpieces will be sequentially brought in for screwing. In this demo station, the workpiece will be placed on the lifting conveyor on the left side. After the robot finished checking any excess screw, the placed workpiece will be lifted it up until the Omron sensor detects a signal. Then, conveyor system will move the workpiece toward the work position where there will be another sensor to detect when the workpiece passed through and the stopper will pull up to stop the piece from continue moving, along with a fixturing mechanism to pull the workpiece up from the conveyor and in working position. Once the workpiece is fixed in place, another signal is sent by the PLC to the robot to indicate that the workpiece is ready for screwing. And then robot will start to work.

Vietnamese-German University

The robot's movement will be as follows: Robot will start to move to the first position. But before it reaches for the exact position, it will first stop at the offset position. Once the robot is at that offset location, it will send out a signal to let the PLC knows to make the screw supplier start to supply one screw to the tool. And when the screw is ready at the tool's head, the robot will move from that offset position to the work position, then the tool will start to push and turn the screw down into the place. Once the screw is in place, robot will move the tool's head back to the offset position and then move to the next position, where the process will be repeated until the whole workpiece is finished. The flow chart below illusrates the working logic of the robot in the auto-mode.



VGU
Vietnamese-German University

Figure 3.1: Flow-chart of robot's movement

3.2 Robot programming

3.2.1 RobotStudio software

To programming and operating ABB's robots, user will have to familiarize themselves with the software ABB RobotStudio. RobotStudio is a versatile and powerful tool for building, developing, and programming industrial robots thanks to its extensive collection of capabilities. Several features that can be mentioned include:

- **3D Robot Simulation**: Accurate and realistic simulations are made possible by the ability to generate intricate 3D models of industrial robots and the environments around them.
- **Offline Programming**: With RobotStudio, offline programming is made easier, allowing users to create and improve robot programs without requiring a physical robot. In the development stage, this helps save time and resources.
- **Path Optimization**: The software enables users to optimize robot paths for improved efficiency and cycle time. This helps in enhancing the overall performance of robotic systems.
- **Collision Detection**: RobotStudio includes collision detection features, allowing users to identify and resolve potential collisions in the virtual environment, minimizing the risk of issues during actual operation.
- **Support for Various Robot Models**: The software supports a wide range of ABB robot models, making it suitable for different applications and industries.

VGU
Vietnamese-German University

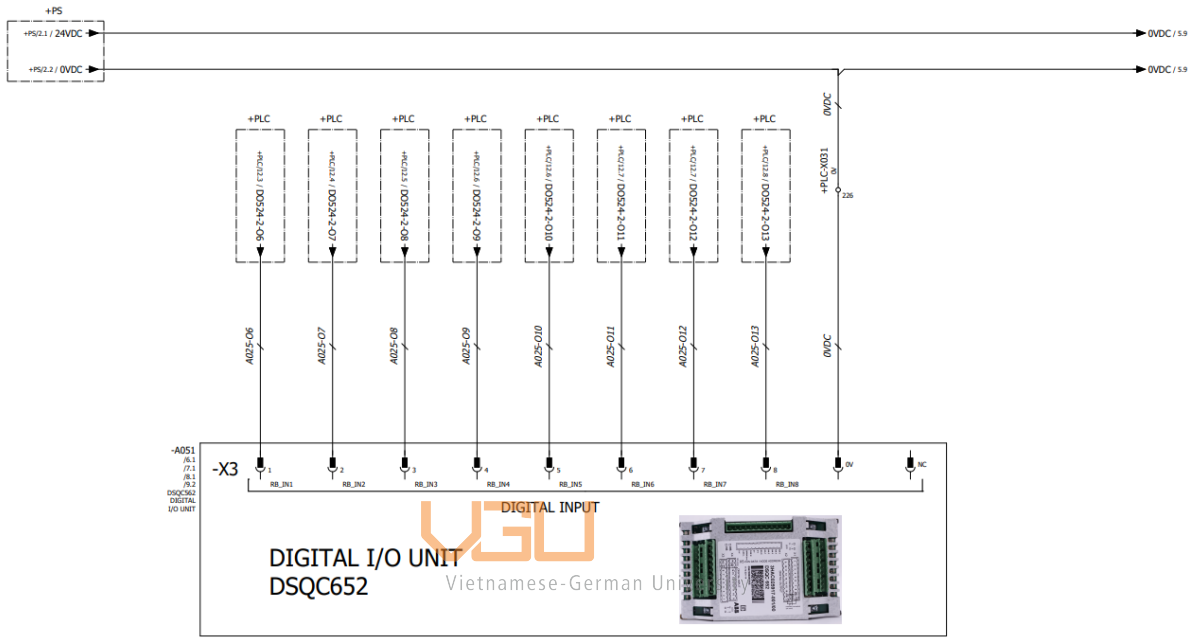
In summary, RobotStudio serves as a comprehensive tool for the entire lifecycle of industrial robot implementation, from design and simulation to programming, testing, and optimization. Its capabilities contribute to increased efficiency, reduced costs, and improved overall performance in robotic automation processes.

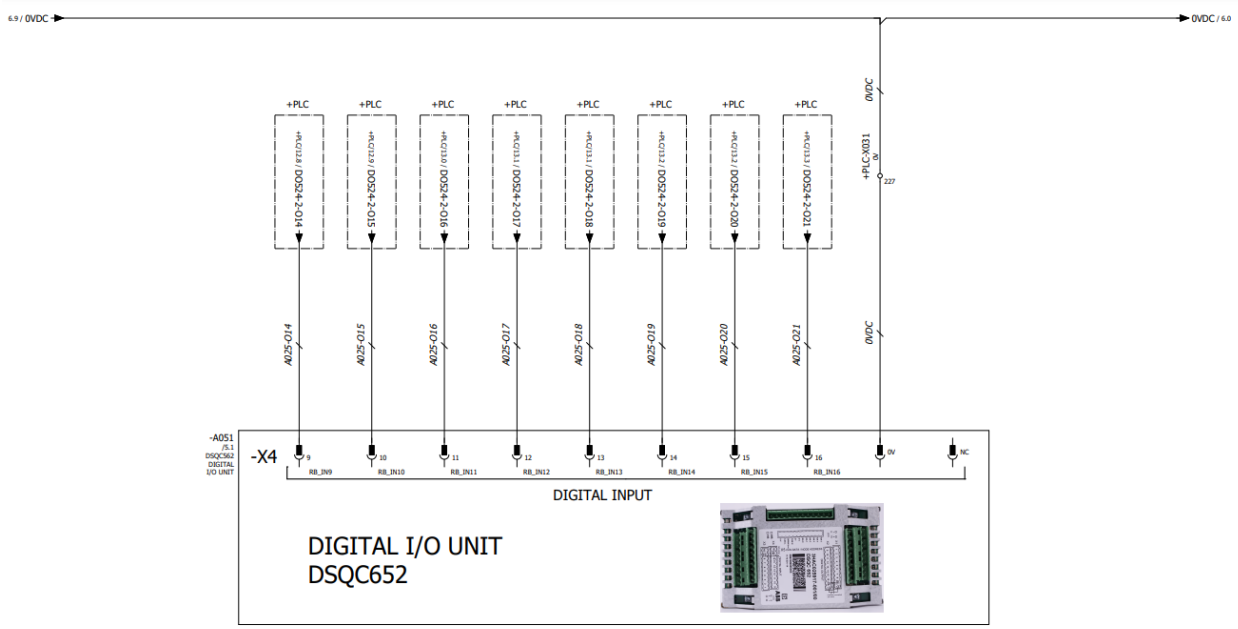
3.2.2 Creating robot's movement

- **Mapping signal.**

Mapping signals is a simple task yet holds significant importance in the operation of our screw-driving station. Since the robot and PLC uses two different sets of outputs and inputs: Robot uses DSQC1030 board and PLC uses I/O buses DI524 and DO524. We have to know for example, an

input of the robot corresponds to what output of the PLC and vice versa, so that we can put in the correct signals in our RAPID code and PLC program. Below are the figures of the electric path circuit of the Input and Output signals between the robot and PLC, note that the only difference is ABB robotics has replaced the DSQC 652 board into the current DSQC 1030, while all the wiring and path remains the same.





VGU
Vietnamese-German Uni

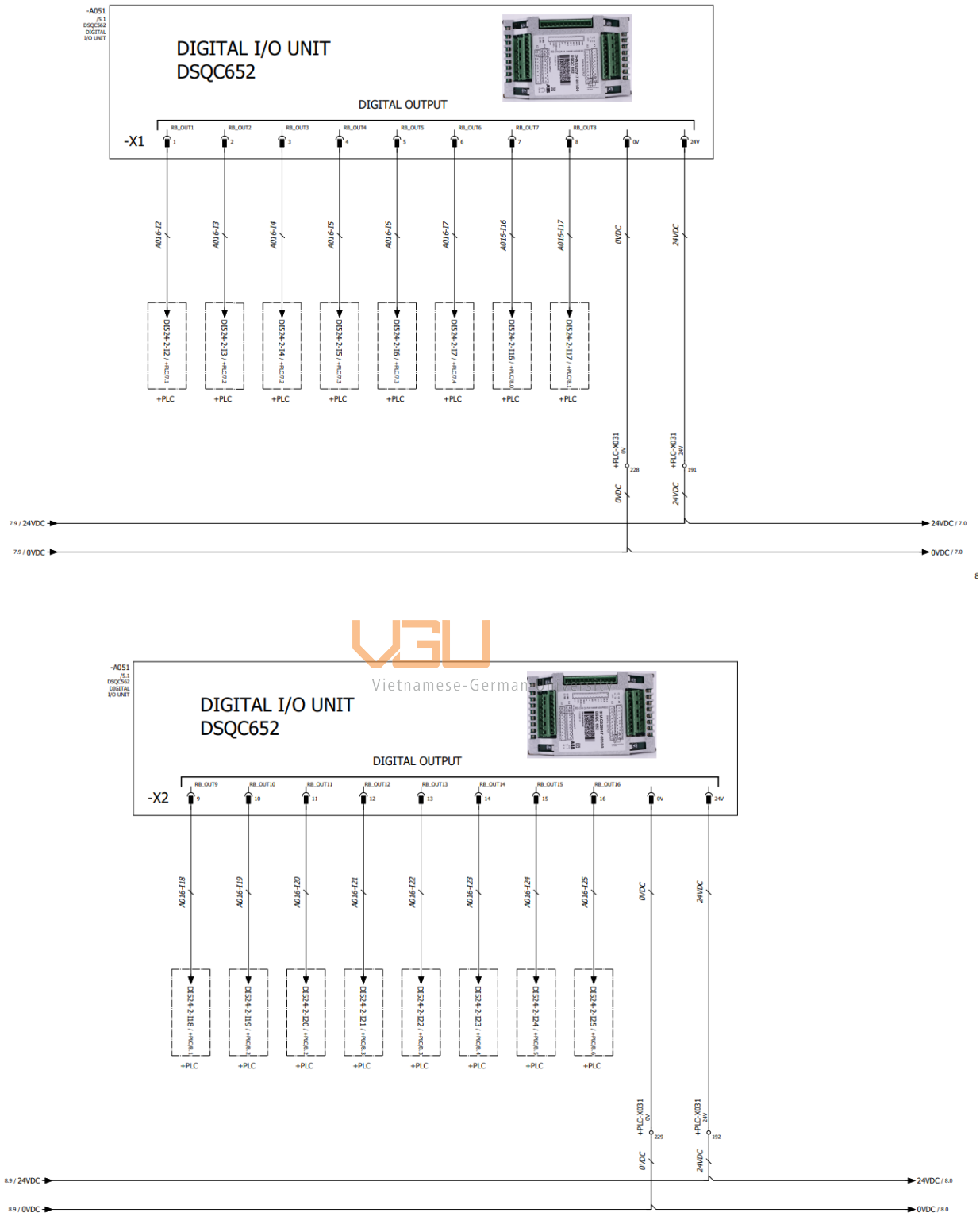


Figure 3.2: Electric path circuit between robot's I/O board and PLC I/O bus.

From the figures, we can easily find out what signal of the DSQC 1030 board connects to what signal on the PLC I/O bus. Note that even though the electric circuit above use the numbers from 1 to 16, the mapping for the DSQC 1030 I/O starts from 0 and ends at 15, which means that the 'RB_OUT1' will be mapping 0 on the DSQC 1030 output list and so on. The mapping process will be as follows: the 'RB_OUT1' is the output of the robot that corresponds to mapping 0 on the DSQC1030, will be connected to the 'DI524-2-I2' of the PLC bus and so on. To be more specific, later on when I program the RAPID to pulse out a signal called 'do08_WorkPosition' which is an output that I put on mapping 8 of the DSQC1030 board, this is the robot's output 'RB_OUT9' and connected to the PLC's input 'DI524-2-I18', and this output will jump into true in the PLC program.

- **Creating new tool in controller**

Each time a different tool is attached to the robot, we need to inform the robot of the exact position of that tool head. Only then can we prevent errors during movement and avoid the risk of the tool head colliding with the workpiece and surrounding components. In the default state, the IRB1100 robot will have tool0, and the position of this tool is located at the center of axis 6. And when connecting the tool to the robot, we want the robot knows the tool's center is not at the bottom of the duck's beak part instead of the original location. The 3d model below shows the position of the tool0 and the position of where we want the robot to acknowledge as the new tool center.

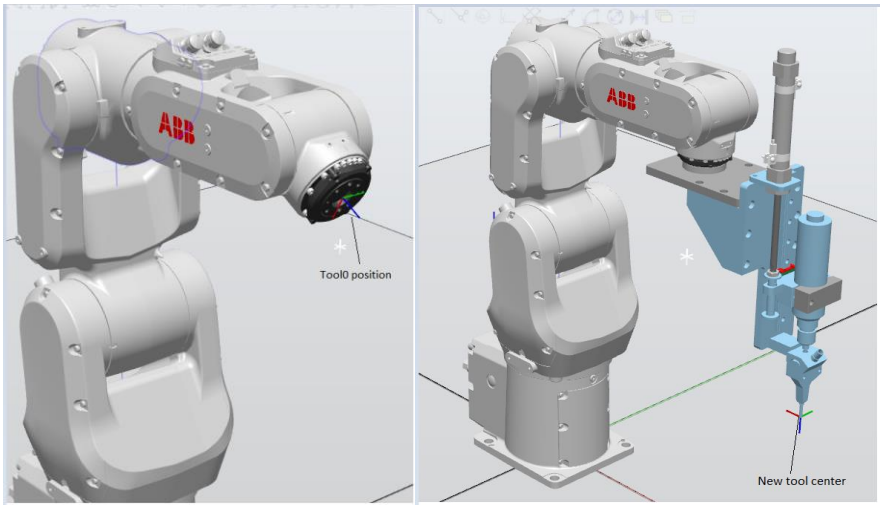
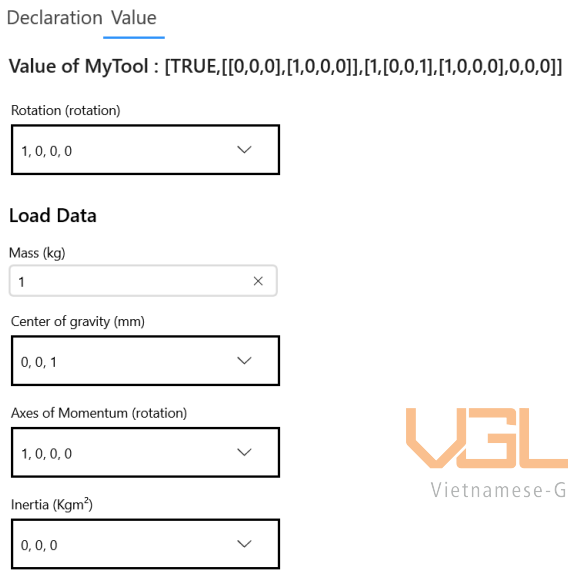


Figure 3.3: tool0 and new tool's position.

In order to do this, I need to use the FlexPendant to teach the robot about the new tool position. This process is called creating new TCP (Tool Center Point). First, from the main interface of the FlexPendant I will go into Calibrate. From there, I tap on ☰ and select Tool. This will show me the list of all tools currently in the controller right now. From then, I tap on + Create New Data and then putting in the name, expected mass and center of gravity. After finishing all the input data, I can hit the 'Apply' for the tool to be shown alongside the tool0. In this station, I named my new tool's TCP is 'MyTool'.



VGU
Vietnamese-German University

Figure 3.4: Inputing data for new tool.

The next step is from the list of tools, I tap on '…' From the newly created tool and tap define, this will lead me into the wizard to define the new TCP.

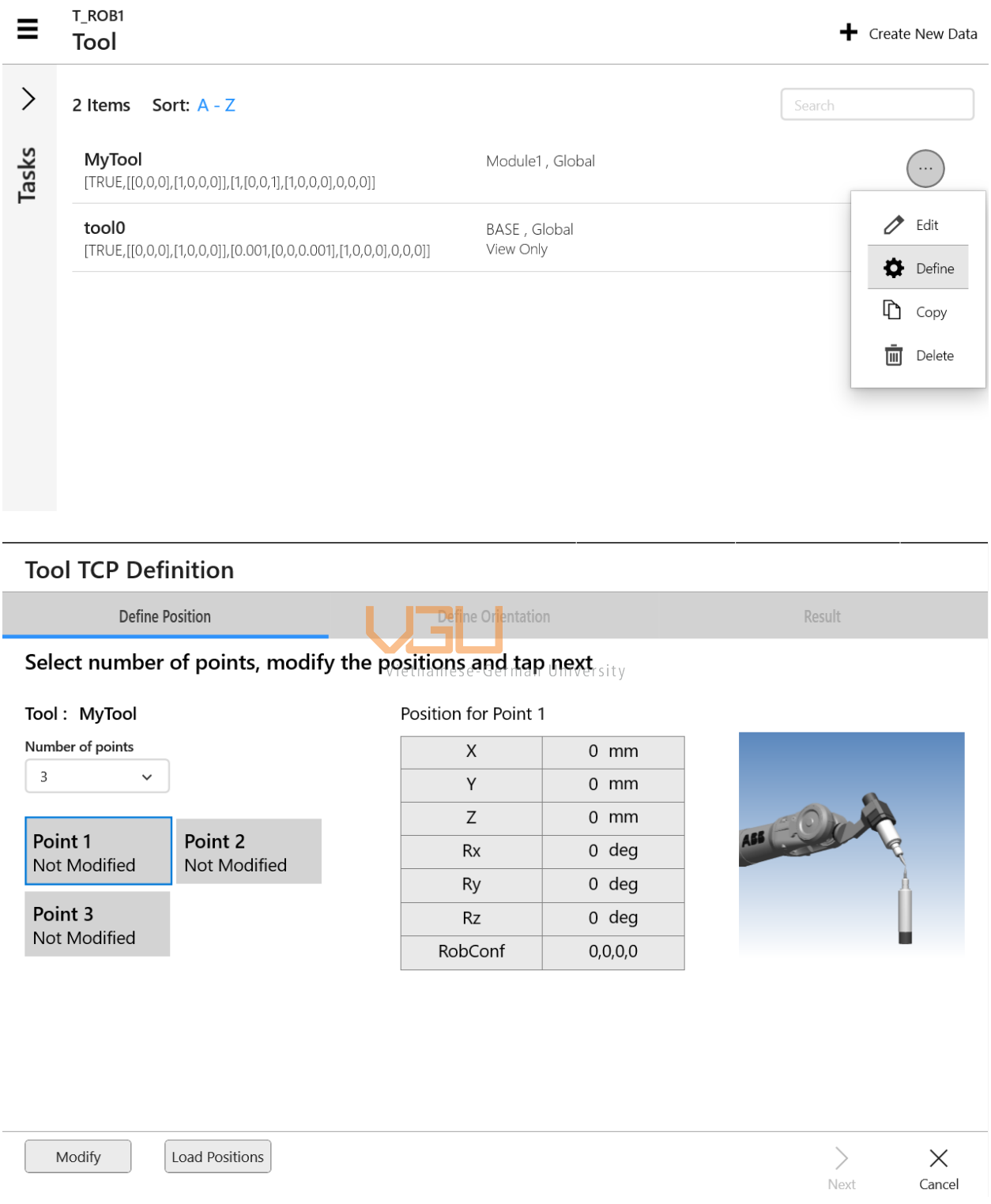


Figure 3.5: Tool TCP Definition wizard.

To define a new tool TCP, I have to locate three or more different point and one point different in Z direction. The requirement for the three points are as follows: All three must have the tip of the duck's beak, where I want my TCP to be, pointed to an exact position, but they have to have different orientation. The point different in Z direction can be from any of those three, it's purpose is to specify the Z direction of the TCP. Once I have finished creating the tool TCP, it's positions will be shown alongside with the tool0. I can double-check if my tool TCP is created correctly or not by jogging the robot in orient mode with the TCP is the one I've just created. Simply goes into the 'Jog' window from the FlexPendant's main interface, selecting Jog Mode: Reorient and select the tool as 'MyTool' and start jogging, if the whole robot's axis move when I jog but the position of my tool TCP stays still, that means I have created the correct TCP.

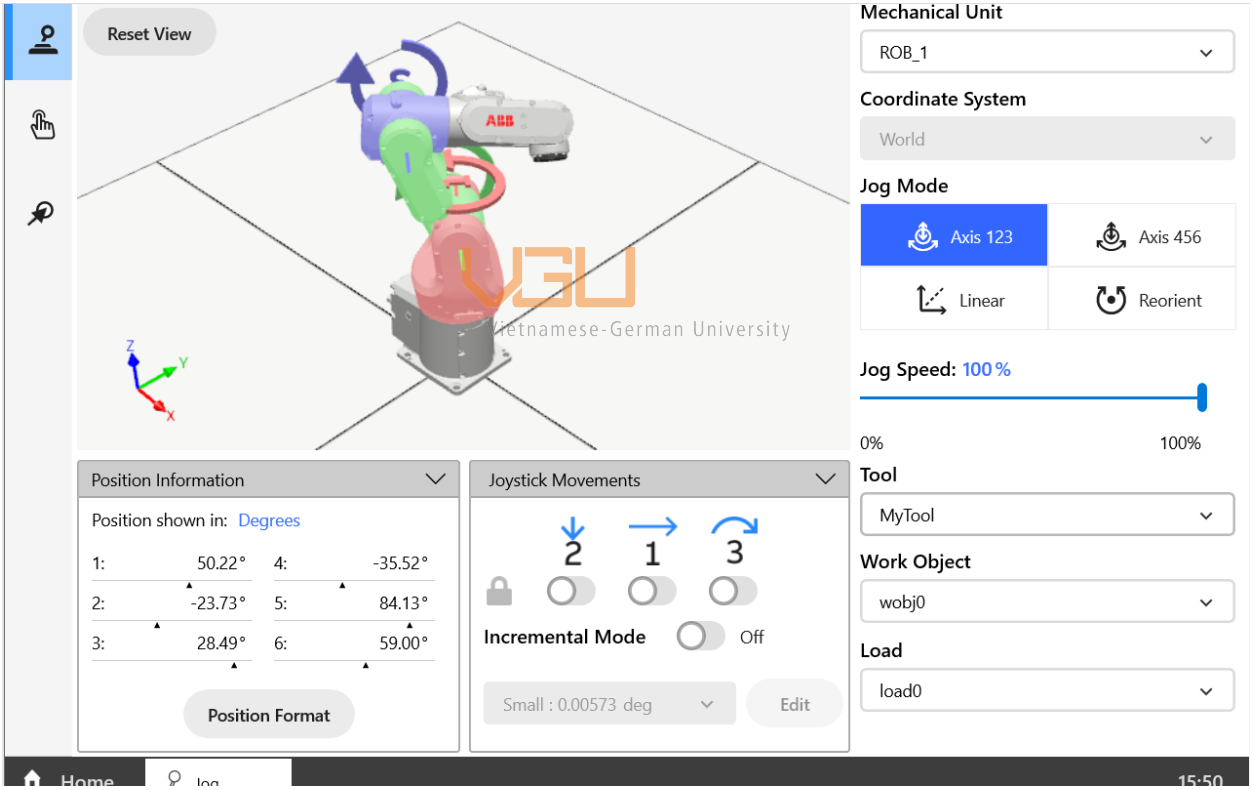


Figure 3.6: Jogging window of FlexPendant.

- **Establish RobotStudio connection.**

My main method of programming the basic's movement is through the software RobotStudio. The first thing I need to do after opening the software RobotStudio is to connect a LAN wire from my laptop's ethernet port toward the Omnicore controller's MGMT port. This will establish a

connection between the software and the robot. Once done that, from the initial window of RobotStudio I will go into the “Online” tab and click on “One Click Connect”.

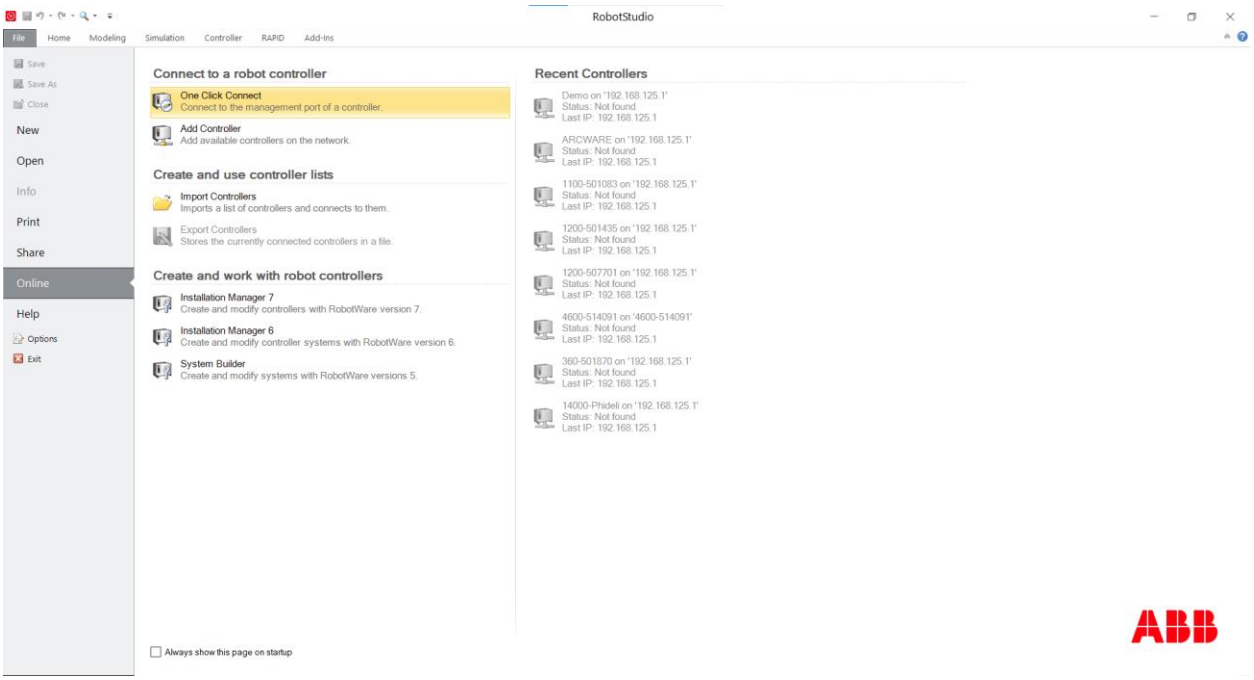


Figure 3.7: Initial interface of RobotStudio

Vietnamese-German University

If the connection is established successfully, there will be a small login box asking to type in the username and password, or choose to log in as a default user. I will choose to log in as a default user then continue to work on programming. If there is an error box saying that: “No controller found on management port”, there are two reason for this. First reason is the LAN wire is being loose from the connecting port, I can simply fix this by making sure the wire is tightly attached. The second reason is my PC is having a different IP address from the Omnicore controller. Normally, these controllers are given the IP address of 192.168.125.1, so what I have to do know is make sure my PC having the address of 192.168.125.x, with x is the different number from 1. To do this, I go into my ‘Control Panel’, find ‘Network and Internet’ then ‘View network status and tasks’, and finally ‘Change adapter settings’. From here, double click on Ethernet and find the TCP/IPv4. Instead of letting it obtain the IP address manually, I will manually input the IP address as follows:

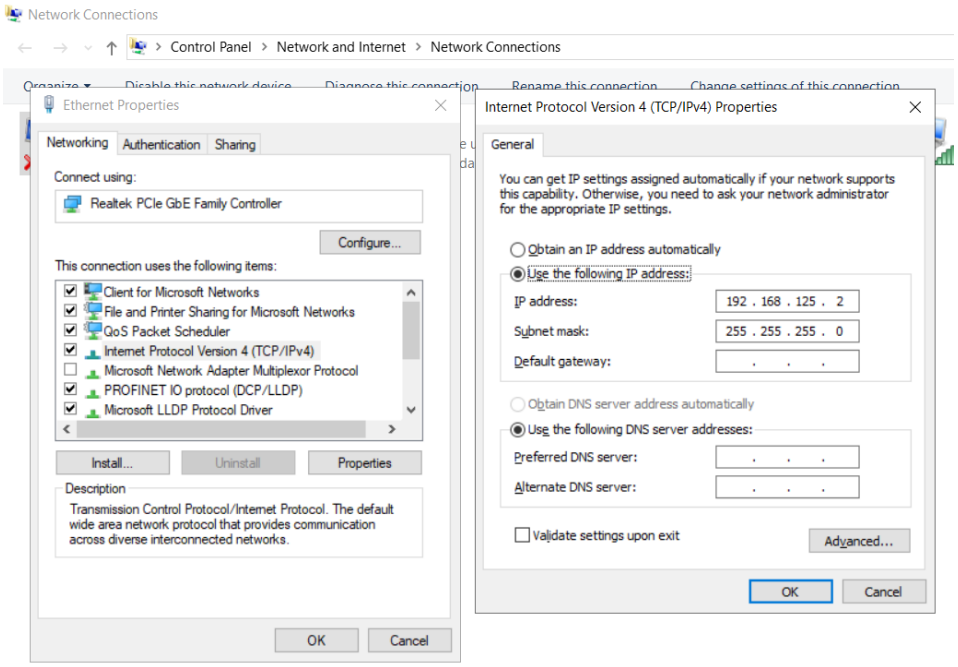


Figure 3.8: Configuring IP address for PC.

Clicking OK after this and click "One Click Connect" on RobotStudio again, this time it should be good to go. Once I'm in the below window, that means the RobotStudio software is ready to work.

VGU
Vietnamese-German University

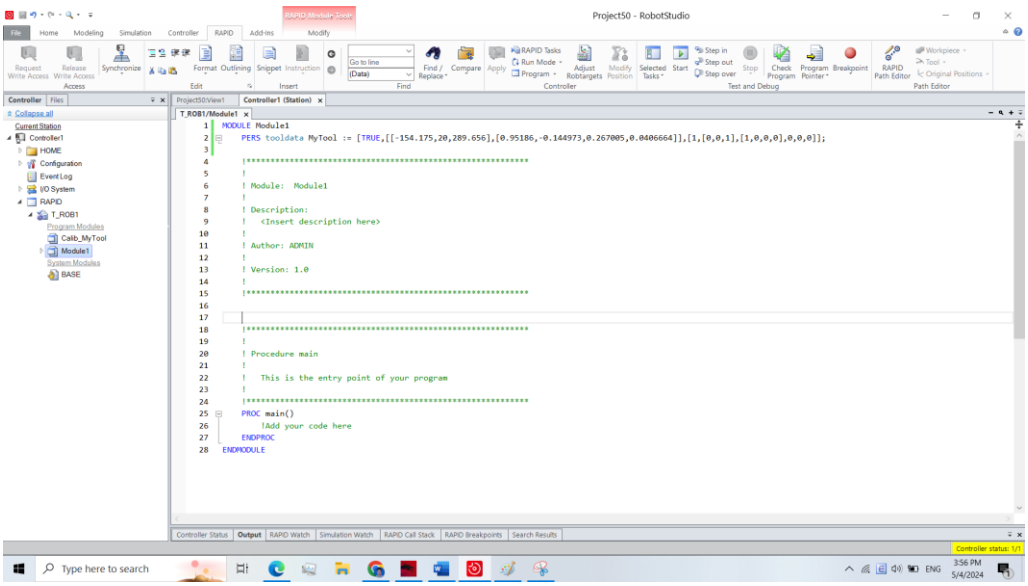


Figure 3.9: Main programming window of RobotStudio.

- **Programming language**

The operating principle of the program in RobotStudio is quite simple and accessible. Robot's movements, signals pulsing and many more can be break down into different processes. Each process can represent one or more fucntions that depends on the user and can be name however we want. Robot will only proceed to run what lies in the 'main' process part of the RAPID. A general structure of a process is as follows:

```
PROC rMove()

MoveJ HomePosition, v200, fine, mytool\WObj:=Wobj0;

ENDPROC
```

The sample code above presents a process with the name of 'rMove', starting from the 'PROC' to annouce the start and naming of a new process, and ended at 'ENDPROC'. The content of this process is to perform an action of moving the robot's tool TCP 'MyTool' to the position defined with the name 'HomePosition'. V200 is speed data and indicates that the speed of this move will be 200mm/s, and fine is the zone data. This data can have different values such as z0, z5, z10, z 20,… and will represent of how smooth the robot will move from one point to another. The lower the value is, the closer the TCP has to be from the actual programmed position before it can continue to move toward another position while the higher value allows robot to fly-by that position.

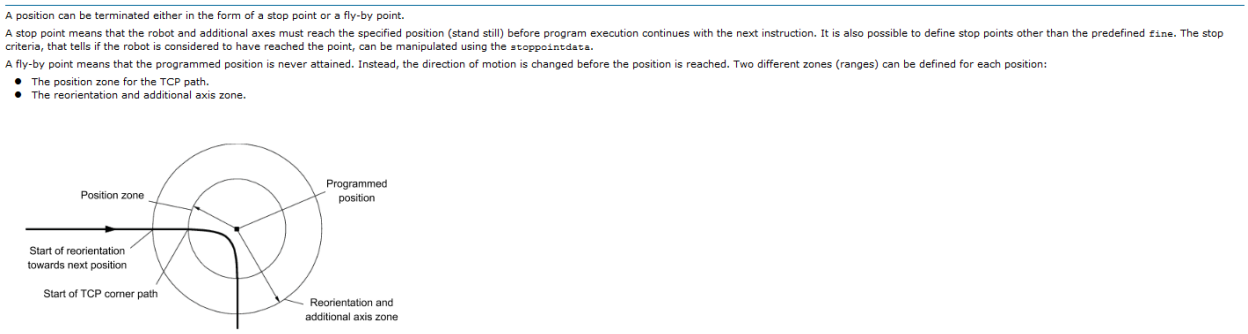


Figure 3.10: Zone-data explanation.

Wobjdata is used to describe the work object that the robot welds, processes, moves within, etc. And in the screwing station, we use the default wobj0. This work object takes the center of the base of IRB1100 as the 0,0,0 point and has the directions as follows:

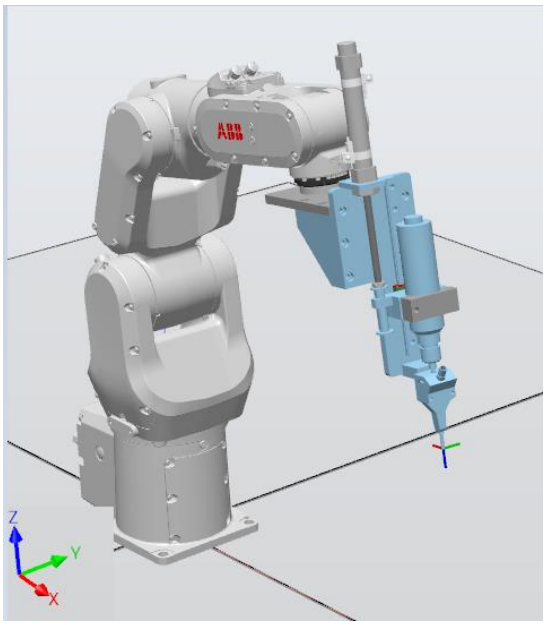


Figure 3.11: Direction of wobj0.

VGU
Vietnamese-German University

Another important thing that I have to take into consideration is the Input and Output signals. This will plays a crucial role in the communication between the robot and PLC. Using signals will have the PLC controlling the current action of the robot, and vice versa, so that the two will be able to work in union. And thankfully, the DSQC1030 is here for us to solve this problem. With up to 16 inputs and 16 outputs, we have plenty of signals to work with. To create signal, I click on the I/O system which can be found on the left side of the screen, and choose ‘Signal’ from the categories.

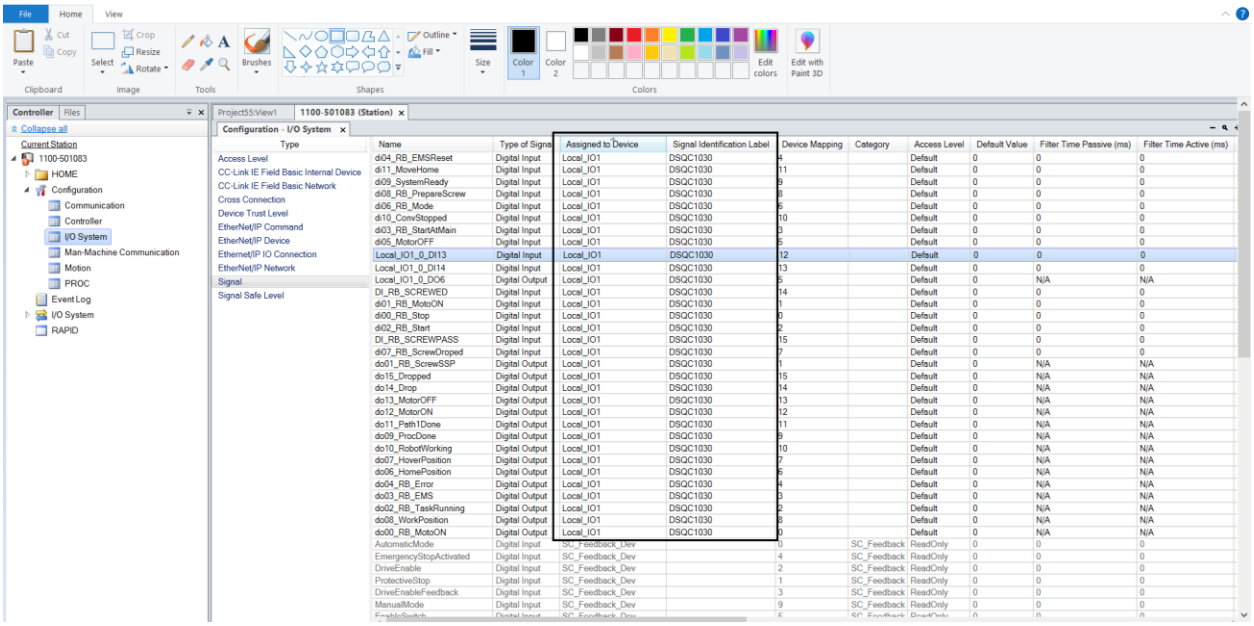


Figure 3.12: Creating signals.

Signals coming from the DSQC 1030 will be displayed with the 'Local_IO1' in their 'Assigned to Device' and 'DSQC 1030' in their identification Label. And I will modify their into my desired names. To modify, right click on the correspond signal I want to change and changing their name and choose 'Edit', a window will pop out for me to type in the name. The figure below shows all the signals that will be used in my program.

VGU
Vietnamese-German University

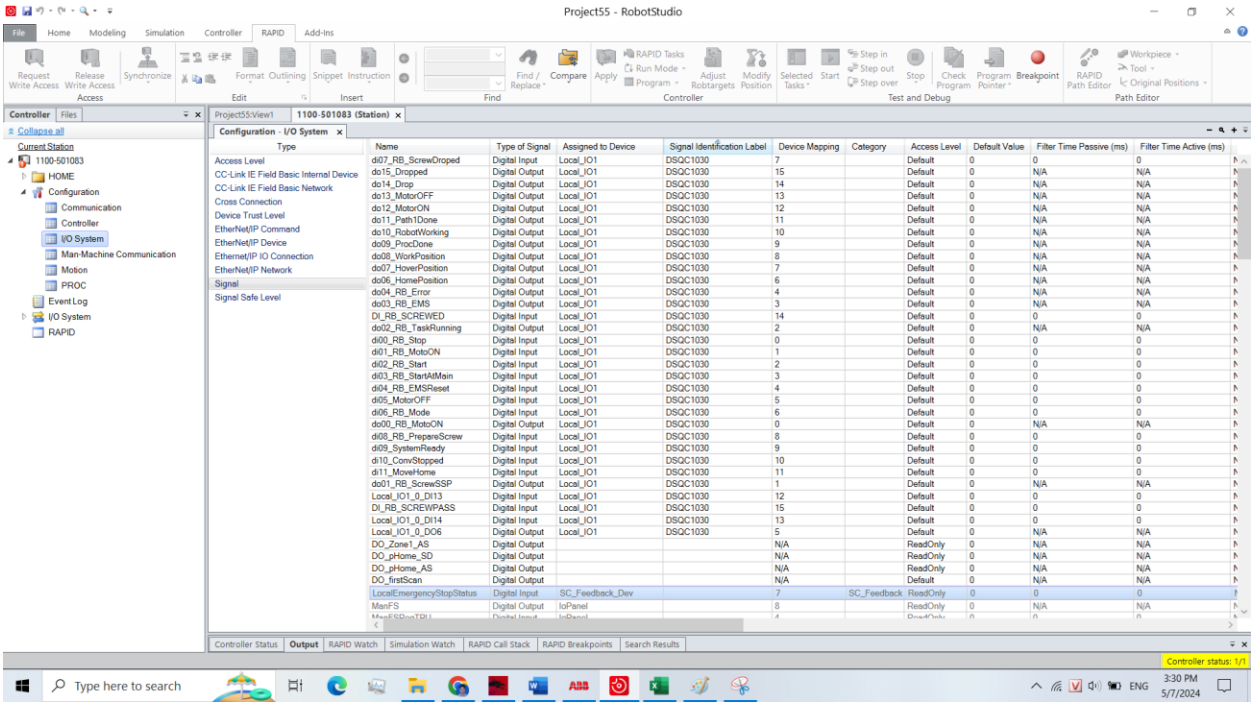


Figure 3.13: All signals created.

The position values in the RobotStudio is called 'robtarget', short for robot targets. And the basic syntax to create a robtarget variable in RobotStudio is:

CONST robtarget p1 := [[x,y,z], [q1, q2, q3, q4], [cf1, cf2, cf3, cf4], [eax_a,eax_b,eax_c,…,eax_f]];

The three x, y and z will be the coordinate of the target in the xyz plane accordingly, while the q1 to q4 will present it's rotation angle through the quaterion angles, and the cf1 to cf4 will represents how the rest of the robot's axes will behave when the tool's TCP moved to that target. For example in the figure below, the axis 4 of the robot can either have the rotation of 0 degree or 180 degree depends on the configuration selected.

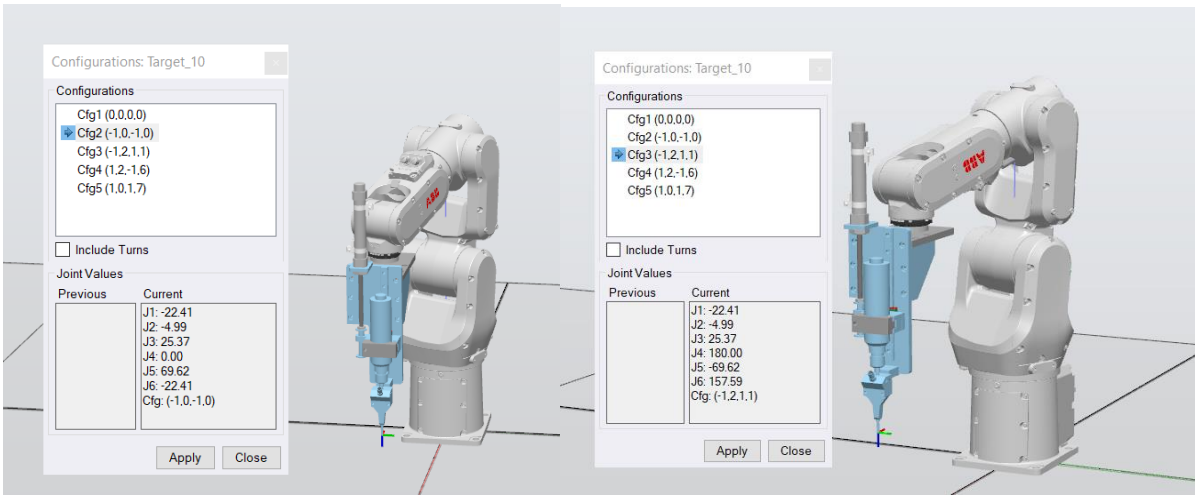


Figure 3.14: different configurations for a target

Though usually, I will not create these target by typing the syntax in like this. Instead, I will just create a random target using the 'Target' function in RobotStudio by my PC first. And then, I will use the FlexPendant to teach the robot where I want my target to be, and all the numbers will be updated into the syntax automatically.

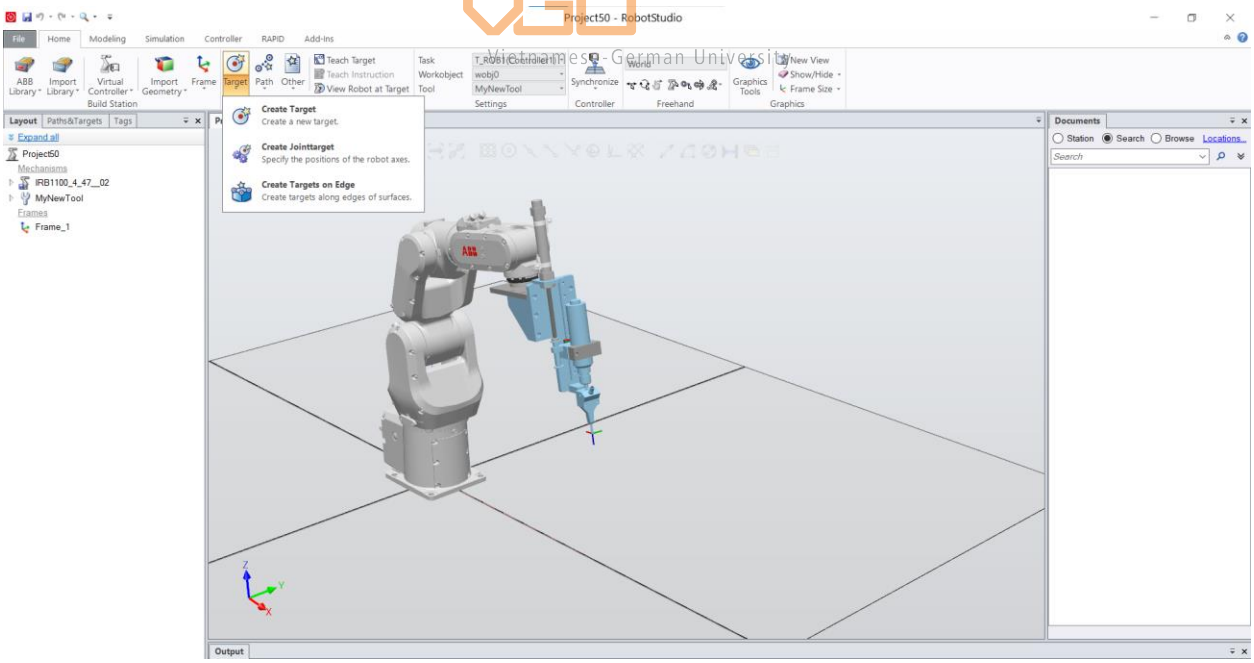
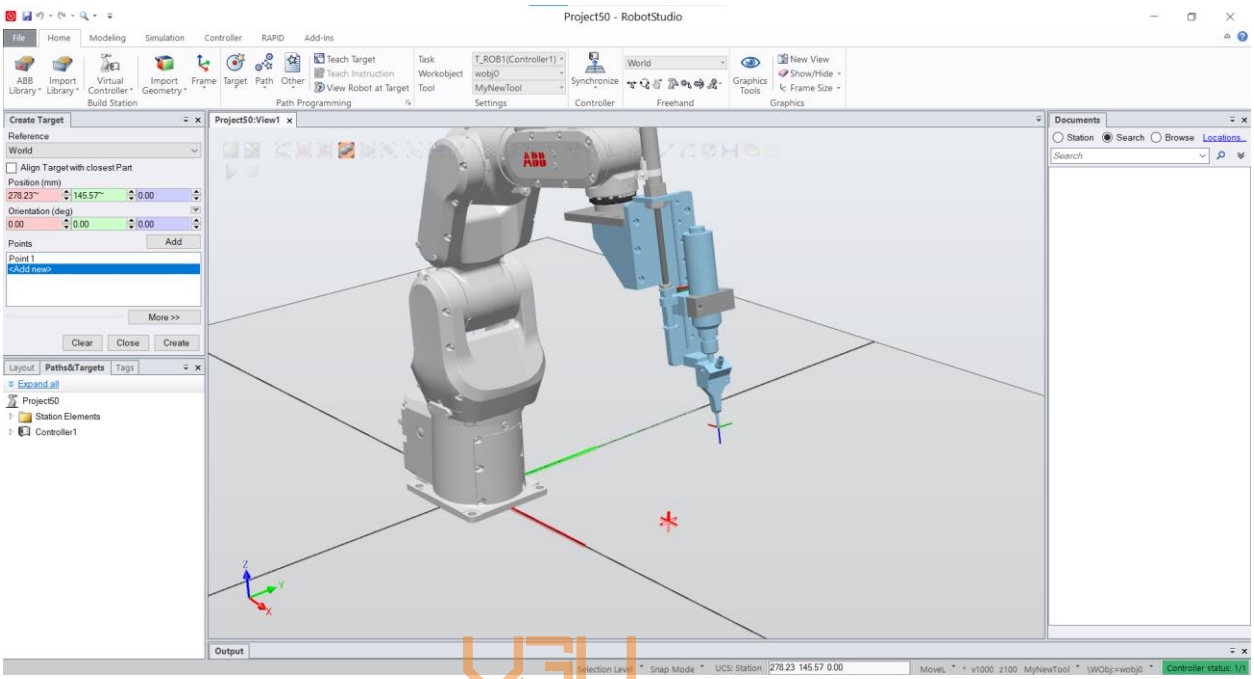


Figure 3.15: Creating target (1).

The ‘Target’ function can easily be spotted from the Home tab. Click on that and choose ‘Create Target’, a window will pop out and I can click on everywhere on the 3D view window, the location will be updated into that window and when I hit create, the target will be created.



VGU
Vietnamese-German University

Figure 3.16: Creating target (2)

The next step is to create a reference path or a process so I can upload the target on it and then load it into the controller, since only until then the RAPID will have the information of the target. To do this, I click on ‘Path’ right next to ‘Target’ and choose ‘New Path’. A path will then be created with the default name of ‘Path 10’. After that, I simply drag the newly created target into the path, and finally using Synchronize to update the target into RAPID.

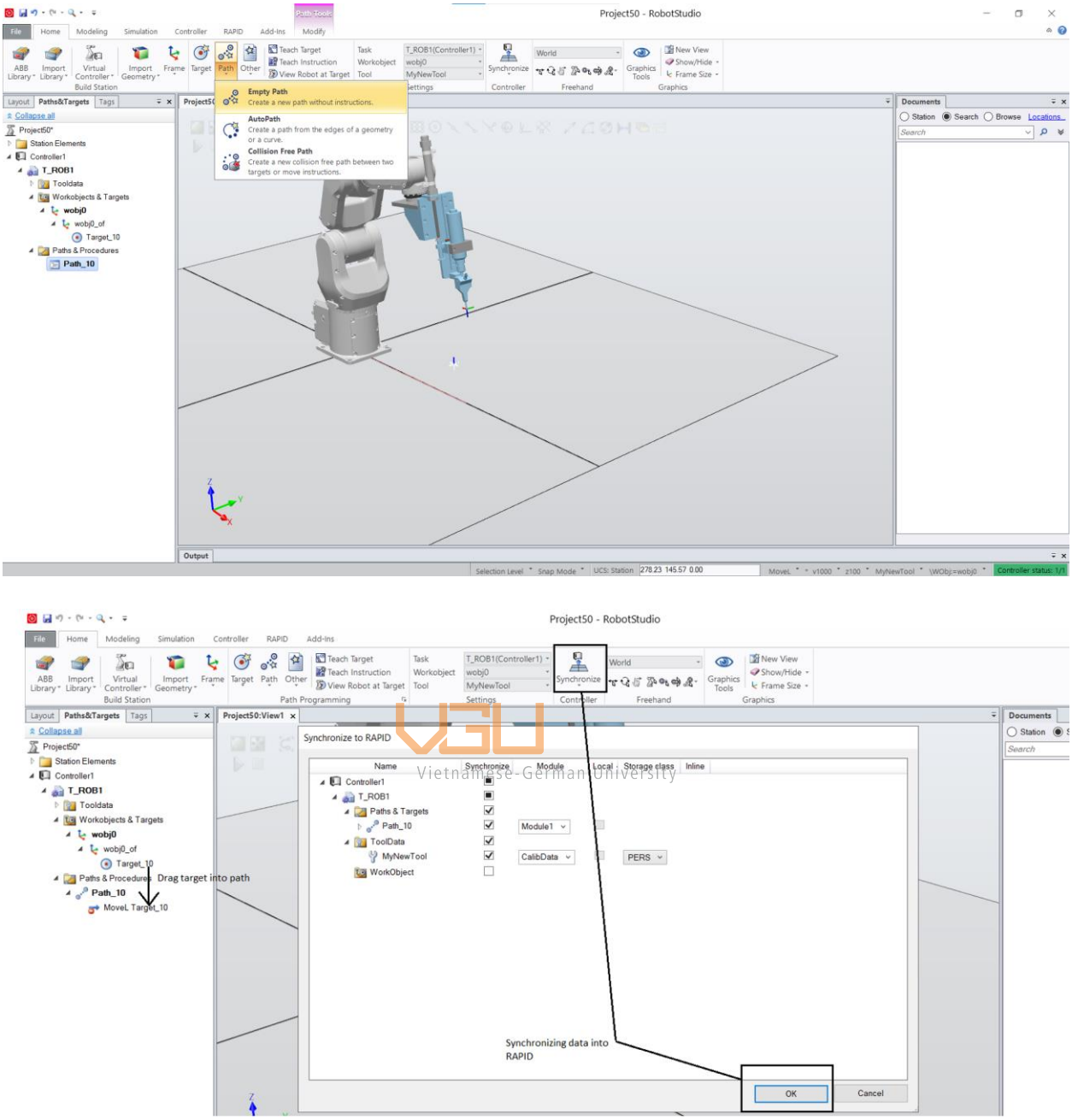


Figure 3.17: Synchronizing targets into controller

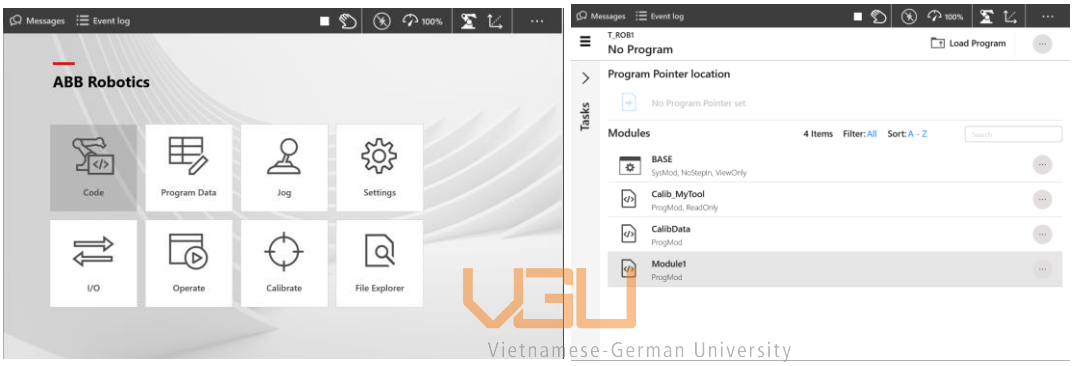
- **Teaching processes and controlling targets**

Once finished all the step, the RAPID program will now has the target data. From now on, whenever I want to create a new target, I can just copy the variable and paste it again while giving it a new name. I will also put all of the created targets into a seperated process called 'Teaching'. For example like this:

```
PROC teaching()
    MoveL Target_10,v1000,z100,MyNewTool\WObj:=wobj0;
ENDPROC
```

Figure 3.18: Process 'teaching'.

Even though this process will contains several moving function only, it's purpose will not be moving robot but instead for me to teach the position of different targets I put in here. To teach a new target, first I jog the robot's TCP to wherever I want the target is at. Then from the FlexPendant, I go into 'Code' and find the process 'Teaching'. For this example, the process 'teaching' is in Module 1, which is a default name of a module in RobotStudio.



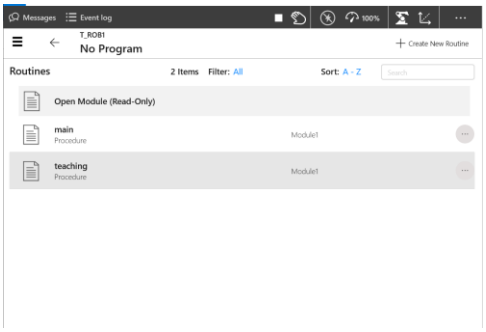


Figure 3.19: Finding process 'teaching' in FlexPendant

Once the tool's TCP is in my desired position, I update it by tapping on the target's name in the move syntax, and tap on 'Update Position'. I can always check the variable in the RAPID code to see if the position is updated or not.

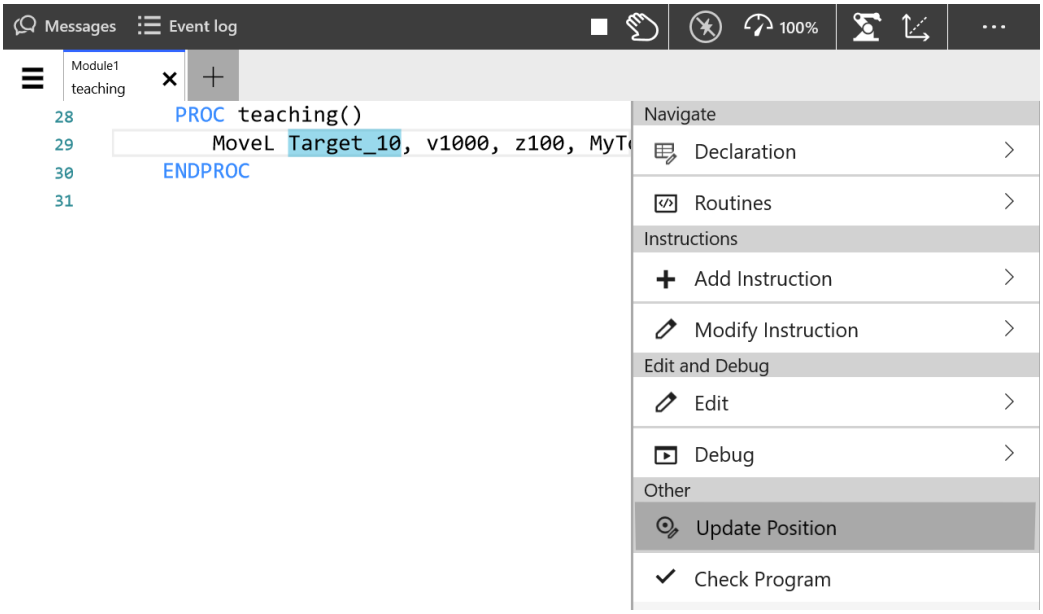


Figure 3.20: Updating position.

According to the programming standard of ABB Robotics, the 'Teaching' process is a very important one that needs to be there for user to control their robot targets. Since there would be a lot of targets that will be made as the programming goes, scatterted in many different modules and sub-processes and it might take a lot of time just to locate where a position is in the program. So, grouping all the targets into one process is an efficient way to control.

To move the robot to a taught position, I tap on 'Debug' and find 'Go To Position' under 'Move Robot' section. Then while keeping the motor on by holding down the three-point enabling device, I tap and hold the 'Press and hold to Go To' to make the robot moves to that position.

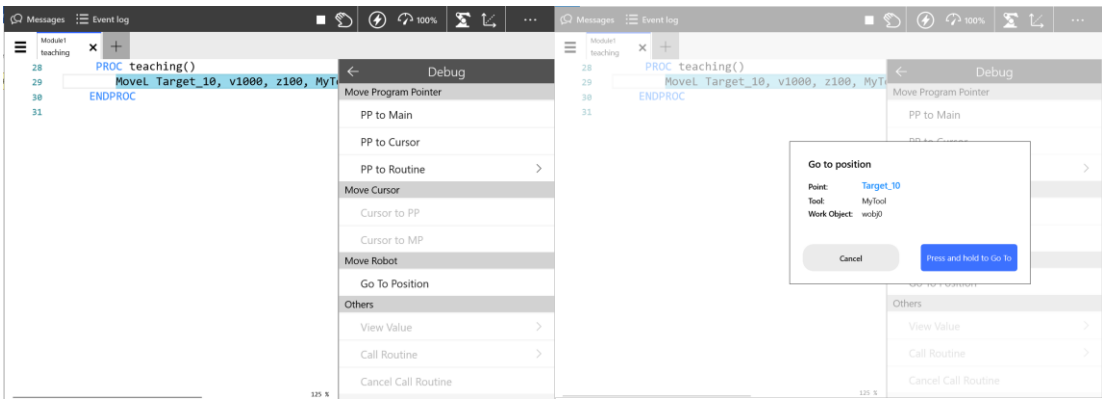


Figure 3.21: Moving robot to a taught position using FlexPendant.

- **Other processes.**

Once everything is ready, it's time for me to start programming the robot's movement. To ensure the program is completely refresh everytime, I have to make the robot to move back to it's 'HomePosition', as well as reseting every signals. This calls for an 'Initial' process. In this 'Initial' process, I will make sure the two signals: 'do10_robotworking' and 'do11_Path1Done' is set back to false as well as creating different possible paths for the robot to move it's TCP back to 'HomePosition'. The 'HomePosition' can be understood as the resting position of the robot before shutting down the power and it also the position at the beginning of the screwing process.

```
7   PROC rInitial()
8       IF DO_firstScan = 0 THEN
9       Zones;
10      SETDO DO_firstScan,1;
11      ENDIF
12      rhome;
13      reset do10_robotworking;
14      reset do11_Path1Done;
15  ENDPROC
```

VGU
Vietnamese-German University

Figure 3.22: process 'Initial'.

The zone process is to create different world zones. For each zone, there will be another path to move the robot's TCP back to HomePosition. This is to simulate the real-life scenario when there might be objects blocking the path in each zone, so if I use the function to move the TCP back to Home by default, it could has some problem like the tool will collide with some object, that's why I have to create zones and different paths. And since the 'zones' process can only run one time for the zones to be established, the IF function is there to make sure the process will not run multiple time since it would cause error.

To create zones, first I need to create some variables. These variables will each store a position in it and will be used to make a zone later on. In RobotStudio, there are two different kind of zones: Sphere and box, and this program will take into account both of them. Sphere zone is created by choosing a position as the center of the sphere and choosing a radius, the zone will then be created with those two information. Box zone on the other hand, is formed by two position indicated Two diagonally opposite corners of a rectangular box. They are presented in the RAPID language as follows: First, I create the variable to store the position in. In this station, I divided the space in a

total of three zones, two box-shaped and one sphere. The two box-shaped zones will be on the front left and right of the robot, while the sphere will obviously be around the 'HomePosition'.

```
VAR pos pHomepos_SD:=[0,0,0];
VAR wzstationary HomeData_SD;
VAR shapedata HomeZoneShape1_SD;
```
→ Variables for create sphere.

```
VAR shapedata Zone1_SD;
VAR wzstationary Zone1Data_SD;
CONST pos Zone1_SD_Low:=[417.73,50.51,95.29];
CONST pos Zone1_SD_High:=[218.77,83.20,132.99];
```
→ Variables for create box zone 1.

```
VAR shapedata Zone2_SD;
VAR wzstationary Zone2Data_SD;
CONST pos Zone2_SD_Low:=[611.89,43.58,-21.42];
CONST pos Zone2_SD_High:=[520.45,-61.19,124.05];
```
→ Variables for create box zone 2.

Now that the variables is ready, I will start making the zones. For the sphere zones around the HomePosition, the syntax is as follows:

```
pHomepos_SD:=HomePosition.trans;
WZSphDef\Inside,HomeZoneShape1_SD,pHomepos_SD,100;
WZDOSet\stat,HomeData_SD\Inside,HomeZoneShape1_SD,DO_pHome_SD,1;
```

VGU
Vietnamese-German University

The first line is to transfer the position of my 'HomePosition' into the 'pHomepos_SD' variable, which means the 'pHomepos_SD' now will have the exact same position as my 'HomePosition'. The second line is to define a sphere zone with the name: 'HomeZoneShape1_SD' which takes the 'pHomepos_SD' as the center and the radius up to 100mm from that center. The last line means that whenever the TCP moves into that zone, it will set the signal 'DO_pHome_SD' to 1, else it will stay as zero. Simliar to this, I create two more rectangular zones and put them in a process as follows:

```
PROC Zones()
    pHomepos_SD:=HomePosition.trans;
    WZSphDef\Inside,HomeZoneShape1_SD,pHomepos_SD,100;
    WZDOSet\stat,HomeData_SD\Inside,HomeZoneShape1_SD,DO_pHome_SD,1;
    !Zone1
    WZBoxDef\Inside,Zone1_SD,Zone1_SD_High,Zone1_SD_Low;
    WZDOSet\stat,Zone1Data_SD\Inside,Zone1_SD,DO_Zone1_SD,1;
    !Zone2
    WZBoxDef\Inside,Zone2_SD,Zone2_SD_High,Zone2_SD_Low;
    WZDOSet\stat,Zone2Data_SD\Inside,Zone2_SD,DO_Zone2_SD,1;
endproc
```

Figure 3.23: process 'Zones'

Next up, the process ‘rHome’ will indicates how the robot will move back into the ‘HomePosition’. In this process, I use the IF function which is pretty straightforward. The robot will have three different ways to move back to the ‘HomePosition’ depending on what signal is being active. And that concludes my ‘Initial’ process of the program.

```
PROC rHome()
    IF true THEN
        !SrewDriving
        IF DO_pHome_SD=1 THEN
            MoveJ HomePosition, v200, fine, mytool\WObj:=Wobj0;
        ELSEIF DO_Zone1_SD=1 THEN
            MoveJ pZone1_SD, v200, fine, mytool\WObj:=Wobj0;
            MoveJ HomePosition, v200, fine, mytool\WObj:=Wobj0;
        ELSEIF DO_Zone2_SD=1 THEN
            MoveJ pDrop, v200, fine, mytool\WObj:=Wobj0;
            MoveJ HomePosition, v200, fine, mytool\WObj:=Wobj0;
        ELSEIF DO_pHome_AS=1 or DO_Zone1_AS=1  THEN
            MoveJ pZone3_AS, v200, fine, mytool\WObj:=Wobj0;
            MoveJ pZone3_SD, v200, fine, mytool\WObj:=Wobj0;
            MoveJ HomePosition, v200, fine, mytool\WObj:=Wobj0;
        ELSEIF DO_Zone2_AS=1 THEN
            MoveJ pZone4_AS, v200, fine, mytool\WObj:=Wobj0;
            MoveJ pZone4_SD, v200, fine, mytool\WObj:=Wobj0;
            MoveJ HomePosition, v200, fine, mytool\WObj:=Wobj0;
        ELSE
            TPWrite ("Vui long Jog robot ve vi tri gan Home");
        ENDIF
    ENDIF
endproc
```

VGU
Vietnamese-German University

Figure 3.24: process ‘rHome’.

After the ‘Initial’ process, the robot will start its first movement. As mentioned before, the robot will have to make sure there are no screw stuck inside the tool before starting a new work process. To do this, I will name this process ‘ScrewChecking’. The content of this process is:

```
PROC ScewChecking()
    MoveL HomePosition, v400, z10, mytool;
    MoveL pDrop, v400, z10, mytool;
    MoveL Drop, v400, z10, mytool;
    WaitTime\inpos, 0;
    PulseDO do14_Drop;
    WaitDI di07_RB_ScrewDroped, 1;
    MoveL pDrop, v400, z10, mytool;
    MoveL HomePosition, v400, z10, mytool;
    PulseDO do15_Dropped;
ENDPROC
```

Figure 3.25: process ‘ScrewChecking’.

It can easily be seen that the process is a group of basic move function along with controlling the signals, with the ‘pDrop’ is the position lies above the ‘Drop’ or ‘pDrop’ is the offset position of

‘Drop’. The robot will moves its TCP from ‘HomePosition’ to ‘pDrop’ and then ‘Drop’, the ‘waittime\Inpos, 0;’ means that the system will have to wait until the TCP reached ‘Drop’ and not any moment before, to pulse out the signal ‘do14_drop;’ toward PLC. PLC then will make the screwdriver tool to pushes down, releasing excess screw (if any) and then pulse out a signal to robot. Robot will wait for this signal to know that the screwdriver has finished the process of releasing excess screw and then move back to ‘HomePosition’, which concludes the process ‘ScrewChecking’.

The output signal ‘do15_Dropped’ is to let the PLC knows the ‘ScrewChecking’ process is done. Once receiving this signal, the workobject will then be delivered from the starting location toward the working position. Once the object arrived, the stopper as well as the fixture mechanism will move up, stopping the object from continue moving as well as fixing it to place. Another signal will also be sent back to the robot, informing that the object is now fixed and robot can start working on it. The object has a total of 28 screw points and located as follows:

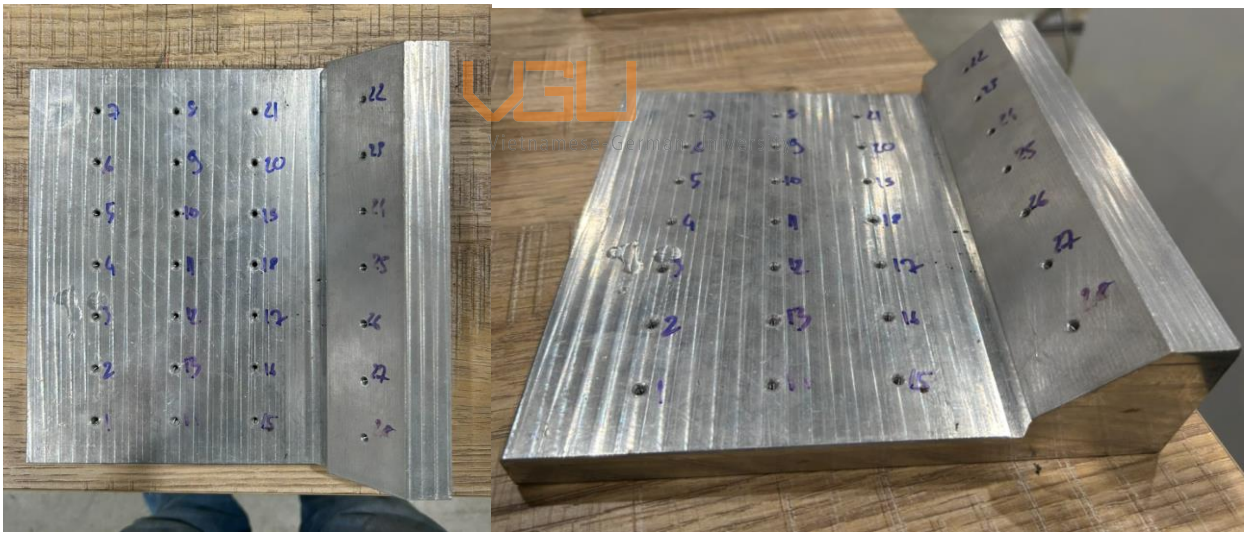


Figure 3.26: Workobject screw’s location.

There are 21 points lie on the horizontal plane 7 points lie on the inclined plane. They will be divided into 7 processes in which I will name them from ‘rPath1’ to ‘rPath7’, and each process will include three points from the horizontal plane and one from the inclined plane:

- Path 7: 7 8 21 22
- Path 6: 6 9 20 23
- Path 5: 5 10 19 24
- Path 4: 4 11 18 25
- Path 3: 3 12 17 26
- Path 2: 2 13 16 27
- Path 1: 1 14 15 28

For path 1, my process will be as follows:

```
PROC rPath1()
    !Chờ tín hiệu con hàng đã tới vị trí làm việc
    WaitDI di10_ConvStopped,1;
    waittime 2;
    !Tín hiệu robot bắt đầu làm việc
    set do10_RobotWorking;
    !Di chuyển tới vị trí tiếp cận lỗ vít 1
    movej offs(Work_1,0,0,5), v400, fine, mytool\WObj:=wobj0;
    !Đưa tín hiệu về plc để cấp vít vào tool
    PulseDO do07_HoverPosition;PulseDO do01_RB_ScrewSSP;
    !Chờ tín hiệu sensor vít đã được cấp vào tool
    WaitDI DI_RB_SCREWPASS,1;
    !Di chuyển đến vị trí lỗ 1
    Movel Work_1, v150, fine, MyTool\WObj:=Wobj0;
    PulseDO do08_WorkPosition;
    !Gửi tín hiệu về PLC để đẩy xi lang đưa tool xuống và bật motor
    WaitDI DI_RB_SCREWed,1;
    !Di về vị trí tiếp cận
    movel offs(Work_1,0,0,5),v150,fine,mytool\wobj:=wobj0;
    !Di chuyển đến vị trí tiếp cận lỗ 2--->


    !MoveL work_14, v400, fine, MyTool\WObj:=Wobj0;
    movel offs(Work_14,0,0,5),v400,fine,mytool\wobj:=wobj0;
    PulseDO do07_HoverPosition;PulseDO do01_RB_ScrewSSP;
    WaitDI DI_RB_SCREWPASS,1;
    MoveL Work_14, v150, fine, MyTool\WObj:=Wobj0;
    PulseDO do08_workPosition;
    WaitDI DI_RB_SCREWed,1;
    movel offs(Work_14,0,0,5),v150,fine,mytool\wobj:=wobj0;
    !MoveL work_14, v20, fine, MyTool\WObj:=Wobj0;
    !PulseDO do07_HoverPosition;PulseDO do01_RB_ScrewSSP;
```

VGU Vietnamese-German University

```
34
35          !MoveJ work_15, v400, fine, MyTool\WObj:=Wobj0;
36          movej offs(Work_15,0,0,5),v400,fine,mytool\wobj:=wobj0;
37          waittime\inpos,0;
38          PulseDO do07_HoverPosition;PulseDO do01_RB_ScrewSSP;
39          WaitDI DI_RB_SCREWPASS,1;
40          MoveL Work_15, v150, fine, MyTool\WObj:=Wobj0;
41          PulseDO do08_WorkPosition;
42          WaitDI DI_RB_SCREWed,1;
43          !moveL work_15, v20, fine, MyTool\WObj:=Wobj0;
44          movel offs(Work_15,0,0,5),v150,fine,mytool\wobj:=wobj0;
45          !PulseDO do07_HoverPosition;PulseDO do01_RB_ScrewSSP;
46
47          !MoveL work_28, v200, fine, MyTool\WObj:=Wobj0;
48          movel offs(Work_28,0,0,5),v400,fine,mytool\wobj:=wobj0;
49          waittime\inpos,0;
50          PulseDO do07_HoverPosition;PulseDO do01_RB_ScrewSSP;
51          WaitDI DI_RB_SCREWPASS,1;
52          MoveL Work_28, v150, fine, MyTool\WObj:=Wobj0;
53          PulseDO do08_WorkPosition;
54          WaitDI DI_RB_SCREWed,1;
55          !MoveL work_28, v20, fine, MyTool\WObj:=Wobj0;
56          moveL offs(Work_28,0,0,5),v150,fine,mytool\wobj:=wobj0;
57          reset do10_RobotWorking;
58
59          !waittime 1;
60          !pulsedo do11_Path1Done;
61      ENDPROC
```

Figure 3.27: process 'rPath1'

Vietnamese-German University

As mentioned above, when the stopper and fixture mechanism fix the workobject in place, it will send out a signal which the robot will receive as 'di10_ConvStopped'. When receive this signal, the robot itself will also set out a signal 'do10_RobotWorking' to the PLC, annoucing that it will start its process now. After that, robot will start moving the TCP to the first location of path 1: point 1. But it will first stopped at the offset position of point 1, which is 5mm above the z-axis. This is displayed in the RAPID as the 'Offs(Work_1,0,0,5)'. Another set of signals: 'do07_HoverPosition' and 'do01_RB_ScrewSSP' will be pulsed to annouce PLC that the tool is in the waiting position and ready to be supply with a screw now. After a screw from the supplier is supplied to the tool's head, PLC will give out a signal 'DI_RB_SCREWPASS' to let the robot now it has finished supplying, and the robot will continue moving downward the work location (line 15). Once the robot's TCP is in the work_1 position, it will pulse a signal 'do08_WorkPosition' back to the PLC so the screwdriver head will start driving the screw into location. And once the screw is tightly fixated into place and the screwdriver is forced to stop by the force sensor at the head of the screwdriver, PLC will sent the signal 'DI_RB_SCREWed' back to the robot, annoucing it to move back upto the offset position and finished the sub-process of screwing one location. I will then

repeat the code with three more location: 11, 15 and 28 and at the end of point 28, I will reset the signal ‘do10_RobotWorking’ again so the PLC knows the robot has finished the ‘rPath1’ process and allows the workpiece to go for another turn on the conveyor before continue ‘rPath2’. This action is as I mentioned before, to simulate the real-life production line when many objects will be deliver by the conveyor to the robot.

I will also create the rest of ‘rPath2’ to ‘rPath7’ with the same syntax, while at the very end of ‘rPath7’ when the robot finished position 22 and the signal ‘do10_RobotWorking’ is reset, I will also pulse out the signal ‘do11_Path1Done’ to let the plc knows the robot has fully completed the whole process.

```
!MoveL work_22, v200, fine, MyTool\WObj:=Wobj0;
movel offs(work_22,0,-5,5),v400,fine,mytool\wobj:=wobj0;
PulseDO do07_HoverPosition;
PulseDO do01_RB_ScrewSSP;
WaitDI DI_RB_SCREWPASS,1;
MoveL Work_22, v150, fine, MyTool\WObj:=Wobj0;
PulseDO do08_WorkPosition;
WaitDI DI_RB_SCREWed,1;
!MoveL work_22, v20, fine, MyTool\WObj:=Wobj0;
!movel offs(work_22,0,-5,5),v150,fine,mytool\wobj:=wobj0;
movel offs(work_22,0,-5,5),v150,fine,mytool\wobj:=wobj0;
reset do10_RobotWorking;
```

VGU
Vietnamese-German University

Figure 3.28: Ending of ‘rPath7’

Now that all the sub-processes has been made, the next thing I need to do is to put them all together and forming a complete process serving screw-driving purpose. I will name this process ‘ScrewDriving’ and create a variable called ‘sum’. ‘Sum’ will take the value of zero at the beginning and the ‘ScrewDriving’ process will be as follows:

```
64    PROC ScrewDriving()
65        WaitUntil di09_SystemReady=1;
66            ScewChecking;
67        IF sum=0 THEN
68           rPath1;
69        sum:=sum+1;
70        ENDIF
71        !waittime 2;
72        IF sum=1 THEN
73            rpath2;
74            sum:=sum+1;
75        ENDIF
76        !waittime 2;
77        IF sum=2 THEN
78            rpath3;
79            sum:=sum+1;
80        ENDIF
81        !waittime 2;
82        IF sum=3 THEN
83            rpath4;
84            sum:=sum+1;
85        ENDIF
86        !waittime 2;
87        IF sum=4 THEN
88            rpath5;
89            sum:=sum+1;
```

VGU
Vietnamese-German University

```
90        ENDIF
91        !waittime 2;
92        IF sum=5 THEN
93            rpath6;
94            sum:=sum+1;
95        ENDIF
96        !waittime 2;
97        IF sum=6 THEN
98            rpath7;
99            sum:=sum+1;
100       ENDIF
101       !waittime 2;
102       IF sum=7 THEN
103         PulseDO do09_ProcDone;
104           sum:=0;
105       ENDIF
106       !waittime 2;
107      ! endwhile
108       ENDPROC
```

Figure 3.29: process 'ScrewDriving'

Simply, the robot will start running ‘ScrewChecking’ the ‘rPath1’ process first and once it’s done the value of ‘sum’ will be plus 1, and then the sum will be one which will make the robot to start ‘rPath2’ and so on until the ‘rPath7’ is finished. And as stated above, RobotStudio will only runs the processes that are lie in the main part of the program. So once I finished with all my process, I have to put them all into the main part of the program, this one contains the ‘rInitial’ process and the main process ‘ScrewDriving’:

```
15      PROC main()
16          rInitial;
17          ScrewDriving;
18
19      ENDPROC
```

Figure 3.30: process ‘main’.

One advantage of creating different processes is that the main program will be very neat and tidy, making the entire program more organized. I also can easily see where changes need to be made if I want to modify or adjust the program.

After finishing the programing process, the last thing I need to do is to teaching the exact screwing positions for the robot. To do this, I first go into the teaching process I named ‘teachpoint’ where I already put in there 28 moving function for 28 targets. Then, I will perform the same steps stated in the “Teaching Process” section to teach my work positions. All of the work positions will be taught 5mm above the threaded-hole to ensure the no-collision between the tool’s duck-beak and workpiece, thus avoiding damage to both components.

```
proc teachpoint()

 Movel Work_1, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_2, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_3, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_4, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_5, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_6, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_7, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_8, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_9, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_10, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_11, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_12, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_13, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_14, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_15, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_16, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_17, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_18, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_19, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_20, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_21, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_22, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_23, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_24, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_25, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_26, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_27, v20, fine, MyTool\WObj:=Wobj0;
 Movel Work_28, v5, fine, MyTool\WObj:=Wobj0;
endproc
```

Figure 3.31: Process 'teachpoint'

VGU
Vietnamese-German University

3.3 PLC programming

ABB Automation Builder is the main PLC software for our demo station. Steps to operate the software are also simple. At the very beginning after launching the software, I will see the home screen as follows:

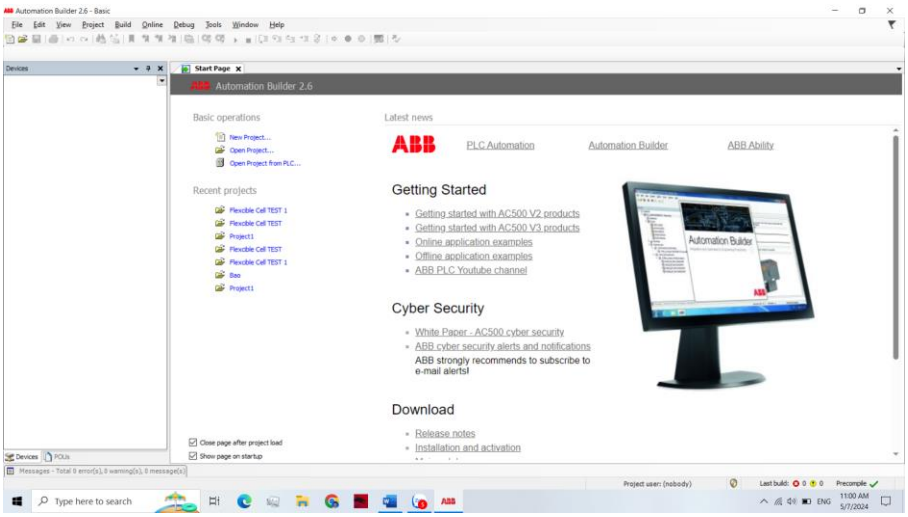
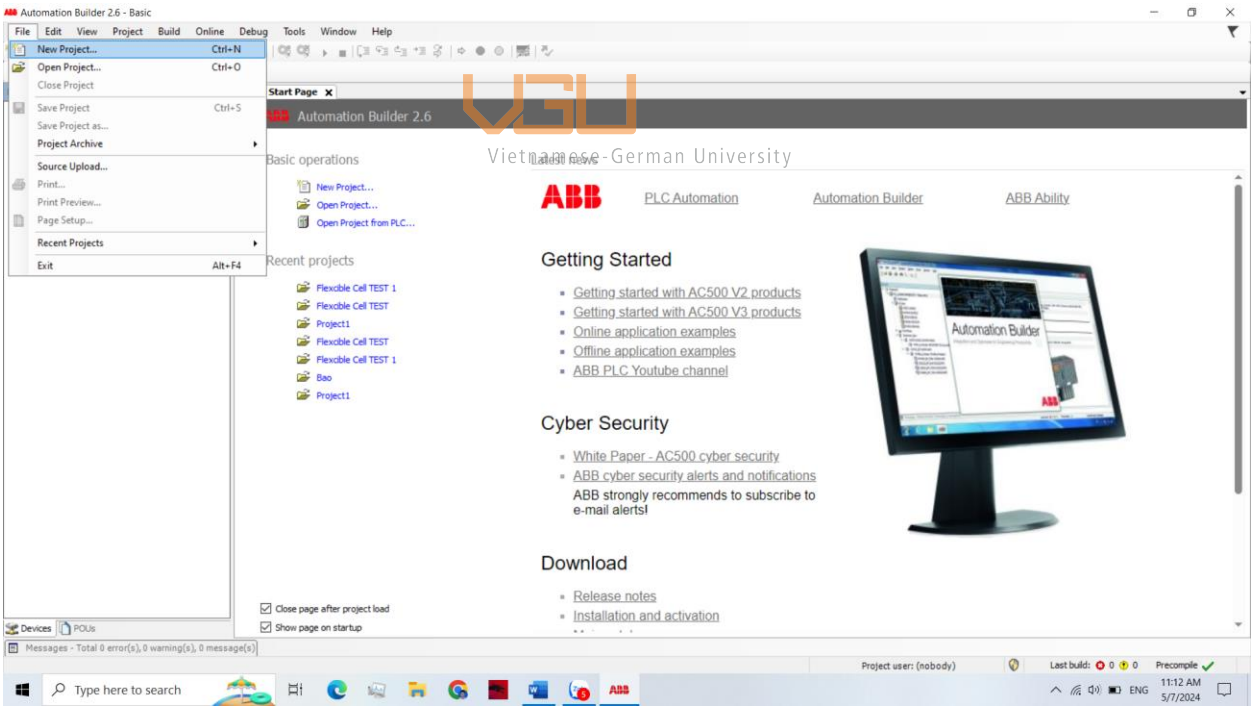


Figure 3.32: ABB automation Builder home screen.

Click on 'File' → 'New Project' on the tool bar or simply 'New Project' from the 'Basic operations' section to start a new project. A small window will pop up for me to select the save location, name of the project as well as the hardware selection.



VGU
Vietnamese-German University

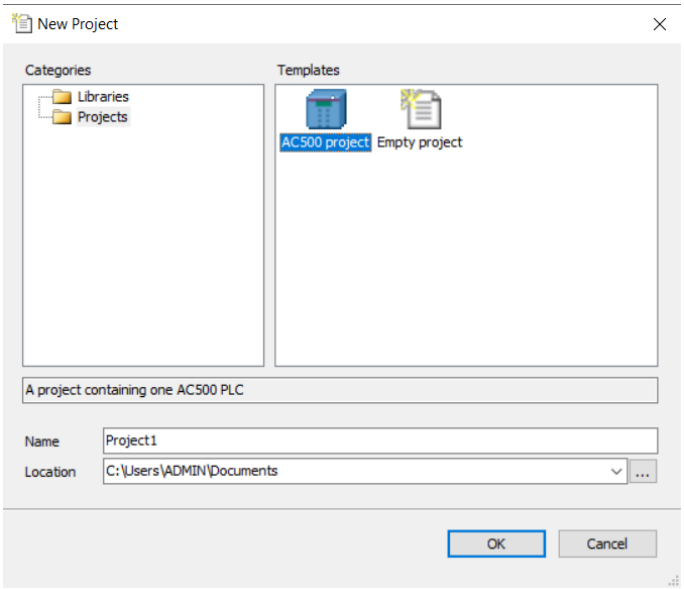


Figure 3.33: Configure ABB's Automation Builder.

Since our station uses the PLC PM 583-ETH which belongs to the AC500 series, I will choose AC 500 project in the Templates, then select the AC500 PM583-ETH which is in the PLC – AC 500 V2 categories. Double-click on 'Application' insde the project window after that will open the CoDeSys – the main interface to program. I can also right-click on the PLC_PRG to add more programming window with different language such as ladder (LD) or function blocks (FBD). The descriptive programming of the PLC will be carried out by my thesis partner, Mr Doan Minh Khang.

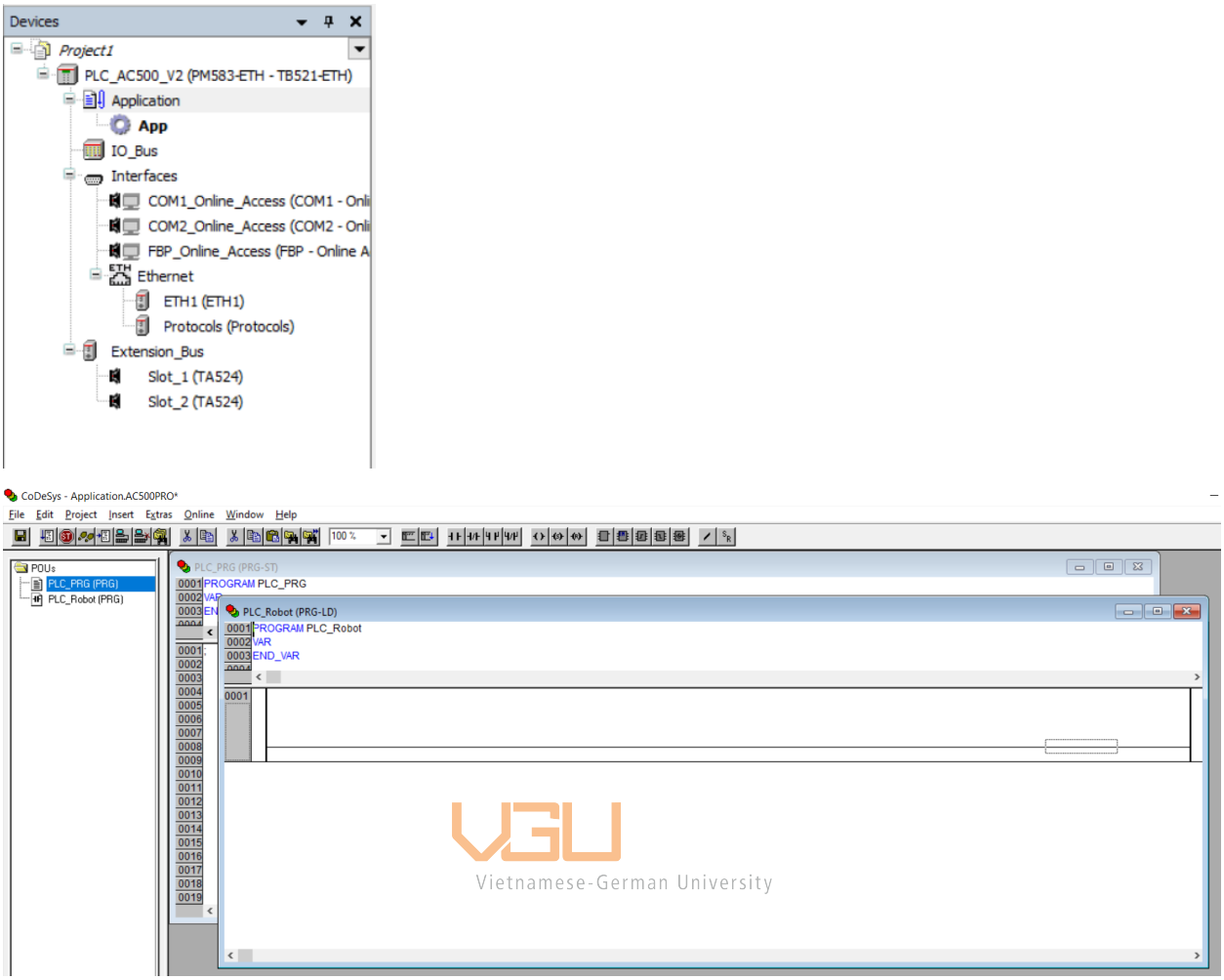


Figure 3.34: Main programming interface of ABB Automation Builder.

Chapter 4. EVALUATING AND OPTIMIZING

4.1 Evaluating and optimizing the demo station's process

- **Finding the approriate speed value**

After completing the robot path programming and ensuring all signals operate as intended, the first thing I need to do is to find out what is the fastest speed possible for the station to operate smoothly and not having problems. To do this, I will simply try running the program with different speed data. To do this, I will keep on executing the process but putting in different value of speed in the proccess 'rPath1' to 'rPath7'.

```
Movel Work_1, v150, fine, MyTool\WObj:=Wobj0;   →   Movel Work_1, v200, fine, MyTool\WObj:=Wobj0;
```

Figure 4.1: Example on changing speed data using RAPID code.

I will start from the low speed value v100 which means 100mm/s for all of the movements, then continue increasing the speed until the problems start to show up. After several trial run, I have sucessfully found out the set of value which will also make the station runs fine while keeping it stable: The robot speed to move from one an offset of a target to an another is 400mm/s, moving speed between the offset position and the work position of one target is 150mm/s. Any higher speed compared to this would cause the robot shaking, affecting the effiecieny and it's life span. And these numbers are also used in the program above.

- **First evaluating and optimizing**

a. Evaluating

My final and most important task is to evaluate the current performance of it and find solutions to optimize the efficiency if the current performance does not meet the requirement. The target set by ABB Robotics is above 99%, meaning that out of every 100 screws, a maximum of 1 screws may be missed. With that number in mind, I will start making a checklist showing the hit and miss rate of as many screws as I can. After the first evaliation, the result came back as follows:

Bảng testing trạm bắn vít số lượng 1000

turn/ thứ tự	Path1				Path2				Path3				Path4				Path5				Path6				Path7			
lần 0	1	14	15	28	2	13	16	27	3	12	17	26	4	11	18	25	5	10	19	24	6	9	20	23	7	8	21	22
lần 1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	x Lệch tool														
lần 2	✓	✓	✓	✓	✓	✓	✓	✓	✓	vít hỏng																		
lần 3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	Lỗi như lần 2																	
lần 4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren																
lần 5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren									
lần 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren														
lần 7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	vít hỏng														
lần 8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	vít hỏng																
lần 9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
lần 10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren														
lần 11	✓	✓	✓	✓	✓	✓	lệch ren																					
lần 12	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren					
lần 13	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren																		
lần 14	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren														

Total vít bắn được	Total vít	tỉ lệ total (%)
189	392	48.21

Figure 4.2: First evaluation checklist.

It can easily be seen that of all 15 routines, there is only one time the robot can screw all 28 target without having a problem. Not mentioning about ABB Robotic's requirement just yet, only 3 out of 15 routine has a sucessful rate above 50% which is generally a very poor performance so far. But these numbers are not surprising considering that this is ABB Robotics' first attempt at automating the screwing process. It's certain that perfection from the start would be almost impossible to achieve.

b. Optimizing

It can easily be seen the main cause of missed screwing is the screws getting misalinged from the threads, in which I noted in the checklist as "lệch ren". The detail explaination for this problem is that even though the tip of the screw has been inserted into the correct position of the threaded hole, the rest of it is misaligned and not perpendicular to the threaded hole. And it will resulted in the screwdriver will not be able to twist it into place when being pushed down, and it can even destroy the threaded hole. The below photo shows the problem occurred in one of the inclined positions.

Figure 4.3: Misalinged screw.

And judging from the checklist, I can see that the problem can occur everywhere on the workobject and not repeatedly at any specific position. This means that the problem doesn't come from the threaded hole itself but may come from the screwdriver tool, more specific is the teaching positions aren't accurate enough. This made me realized a mistake I made in the process of teaching the screwing position, which is not checking every position manually after teaching it. Knowing this, I started optimizing the demo the following day. Firstly, I double-check all the positions by using the FlexPendant to jog the robot to each work position, and manually pulsing the signal to make the screwdriver twists the screw into the threaded hole, and proceed to re-teach all the positions that have the problem.

Once that is finished, I continue to address the problem of broken screw, which I noted as "vít hỏng". This problem occurs simply because among the hundreds of screws used for screwing into the workpiece, some of them have damaged heads and are stripped. This problem also related directly from the misalignment problem mentioned earlier. Specifically, when a screw is misaligned from the threaded hole, the screwdriver tool presses and rotates downwards, but it still can't enter the hole and instead gets stuck. This leads to strong collision and friction between the screwdriver tool and the head of the screw and will destroy the screw's head and lead to broken screw. To address this issue, I simply take out all of the screws in the supplier and then proceed to check one by one. I check them by pushing the screwdriver's head into every screw's head and twist the screw by hand. If any screw slipped from the screwdriver's head, it means that the screw is broken and I will discard it.

VSU
Vinh University

Figure 4.4: Checking screw manually.

- **Second evaluating and optimizing**

a. <u>Evaluating</u>

After the first optimizing process, the result for the next evaluation came in as follows:

lần 15	✓	✓	✓	✓	✓	✓	lệch ren																				
lần 16	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren								
lần 17	lệch ren	✓	✓	✓	✓	✓	lệch ren	✓	lệch ren	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren								
lần 18	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	vít hỏng								
lần 19	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	lệch ren	✓	✓	lệch ren	✓	lệch ren	✓	✓	✓	✓	✓	✓	lệch ren						
lần 20	✓	✓	✓	✓	✓	✓	lệch ren	✓	lệch ren	✓	✓	✓	lệch ren	✓	✓	✓	✓	✓	lệch ren	hết vít								
lần 21	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	lệch ren	✓	✓	✓	✓	✓	✓
lần 22	✓	✓	✓	✓	✓	✓	✓	vít hỏng																				
lần 23	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	✓	lệch ren															
lần 24	✓	lệch ren	✓	lệch ren	lệch ren	lệch ren	✓	✓	✓	✓	lệch ren	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	lệch ren	✓	✓	✓	✓	✓	✓
lần 25	✓	✓	✓	lệch ren	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	lệch ren	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	✓	✓
lần 26	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	✓	lệch ren	✓
lần 27	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	✓	✓	✓	lệch ren	✓	lệch ren	✓	lệch ren	✓	✓	✓	✓	✓	✓	lệch ren	lệch ren			
lần 28	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	lệch ren	lệch ren	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren				
lần 29	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓	lệch ren	✓	✓	✓	✓	✓	✓	lệch ren	✓	✓		
lần 30																												
lần 31																												
lần 32																												
lần 33																												
lần 34																												

Total vít bắn được	Total vít	tỉ lệ total (%)
265	420	63.10

Figure 4.5: Second evaluation checklist

Although the accuracy rate of screwing has increased from 48% to 63%, it's still far from our goal and it's also evident that misalignment issue remains the primary cause of errors. This means that double-checking the positions of the work positions may not be the most accurate solution, as the underlying cause may still come from another factor. And since I'm not sure exactly what that factor is, the only thing I can do is to run more test cycles combined with observation and recording slow-motion videos. And after several test runs and study my slow-motion videos, I finally identified a serious issues: the fixture mechanism is not working as intended as the workpiece is still shifting and isn't securely fixed on it.

The fixture mechanism of our the demo station is a metal plane fixed on an air cylinder and on that plane will be two locating pin placed diagonally to eachother. Initially, It will lay beneath the conveyor and will be pushed up when receive signals. And simultaneously, on the underside of the workpiece, there will also be two corresponding holes for the two pins to go into and secure it. And that when the problem comes in, there were an error in the machining process of the workpiece leading to one of the two holes on the workpiece bigger than the other and the pin and as a result causing the workpiece shifting everytime the screwdriver pressed down the threaded-holes.

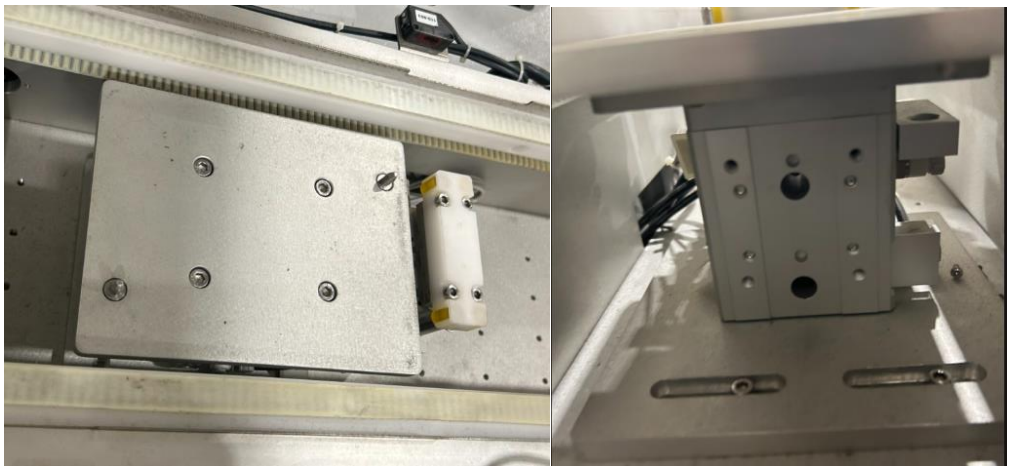


Figure 4.6: Fixture mechanism.

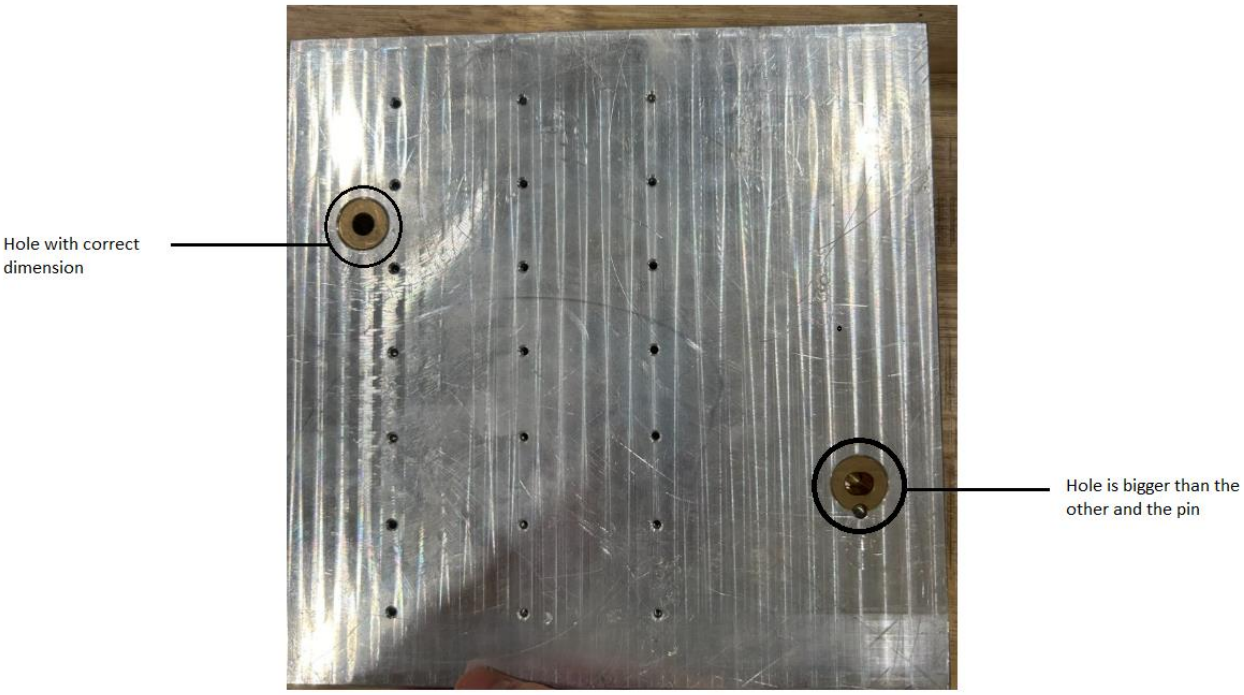


Figure 4.7: Locating holes on workpiece with problems.

VGU
Vietnamese-German University

b. Optimizing

Since this workpiece has already been machined beforehand, I have no way to intervene and make direct adjustments on it. Instead, I will have to adjust the position of the stopper to fix this problem. Like the fixture mechanism, the stopper contains a rectangle metal piece installed above the air cylinder, that piece will be pushed up to block the workpiece as it's moving on the conveyor. Luckily, the stopper is also mounted on it's own locating system, which I can manually shifting the position by unloading the two screws holding it and moving the stopper between an allowed distance.

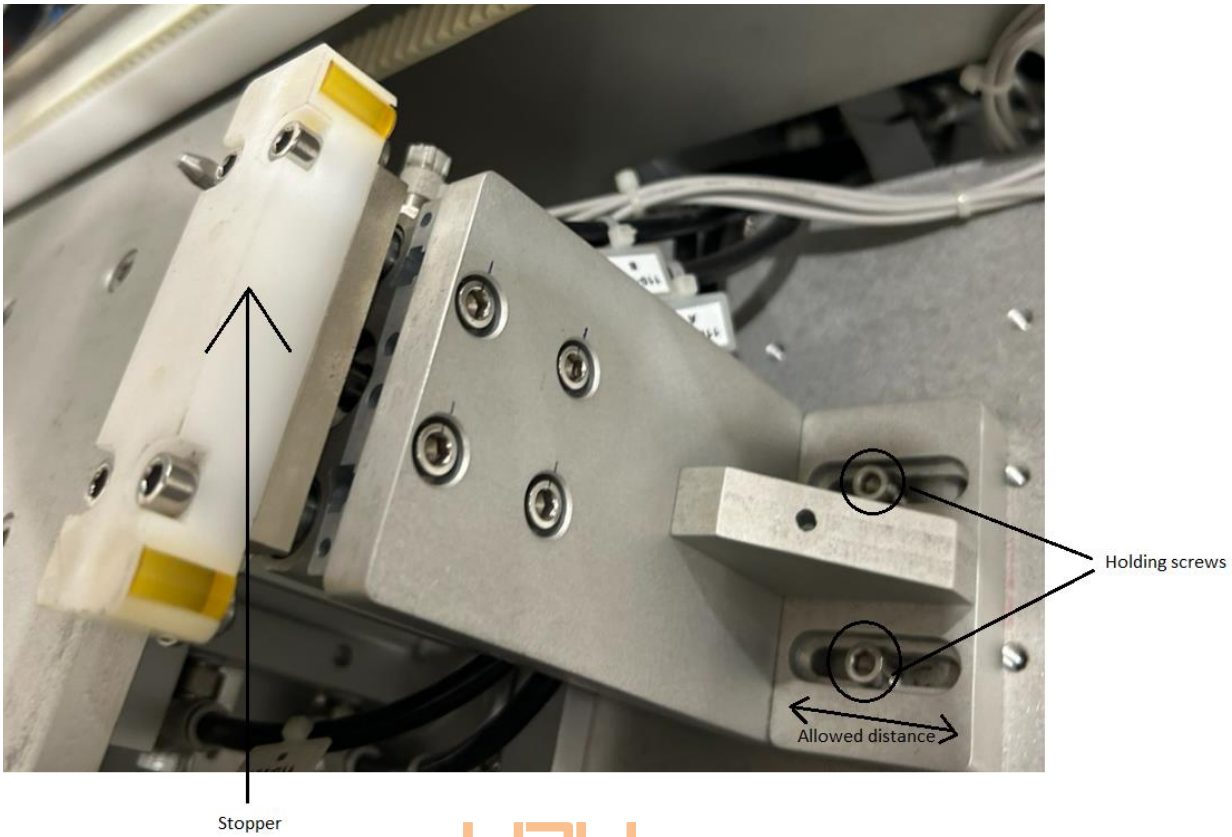


Figure 4.8. Stopper.

So far, there's still a small distance between the workpiece's fixture mechanism and the stopper. The reason for this is I want to make sure there would be no collision between those two in the working process. But after seeing the serious problem mentioned above, I decided to shift the stopper closer to the fixture mechanism and somewhat pressing into it. The idea is to also turning the stopper into a thrid 'locating pin' and stop the workpiece from shifting in the screwing process. The result is as follows:

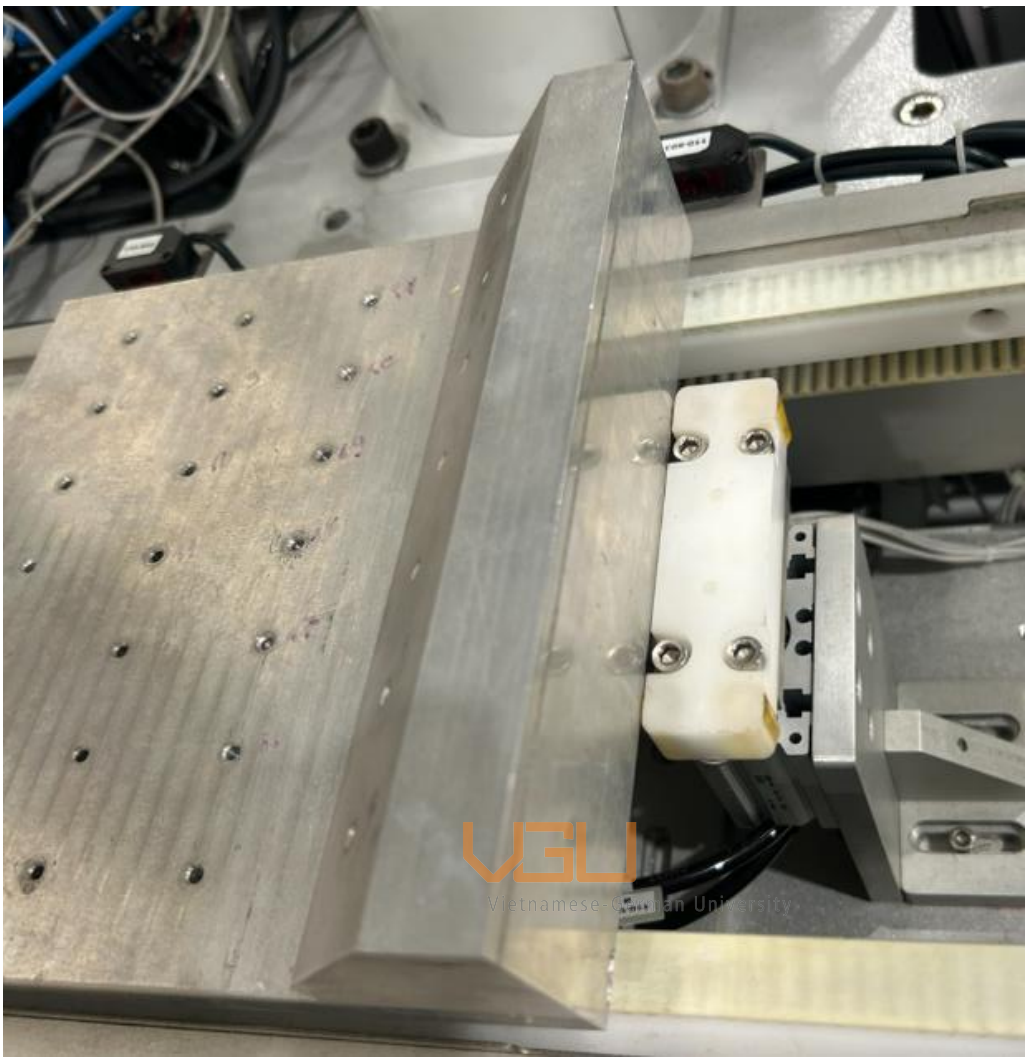


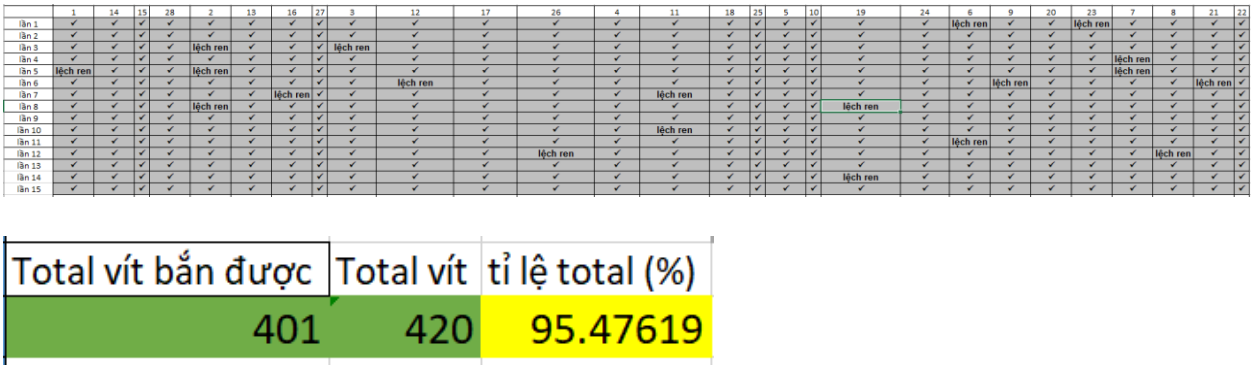
Figure 4.9: Fixture and stopper mechanism after adjustment.

It can be seen that now, the stopper is also securely fixed against the workpiece. The little yellow pieces on the four corners of the stopper is made of rubber and will ensure elasticity to prevent damage from collisions between it and the workpiece.

- **Third evaluation and optimizing**

a. Evaluating

After fixing the problem of the workpiece shifting, the result of the following evaluation came in as follows:



Total vít bắn được	Total vít	tỉ lệ total (%)
401	420	95.47619

Figure 4.10: Third evaluation checklist.

Compared to the last evaluation, this time there has been a tremendous increase in the hit rate of our screw-driving demo, which means I have found and succesfully fixed what could be seen at the most serious issue so far. However, this rate is still not perfect as it's lacking about 4% compared to the requirement set by ABB Robotics and again, the remaining issue is still misalinged screws, though significantly lesser than before.

b. Optimizing

VGU
Vietnamese-German University

Instead of addressing the issue like how I did in the first evaluation, I decided to approach it in a different view. This time by manually inspecting the screwdriver's movement combining with slow-motion videos, I noticed a detail that is most likely the root cause for the problem. Right at the moment the screwdriver's head pressed down and the duck's beak part opened, the screw itself is shaking due to eccentricity. This happens because the screwdriver's head itself is also eccentric when rotating.

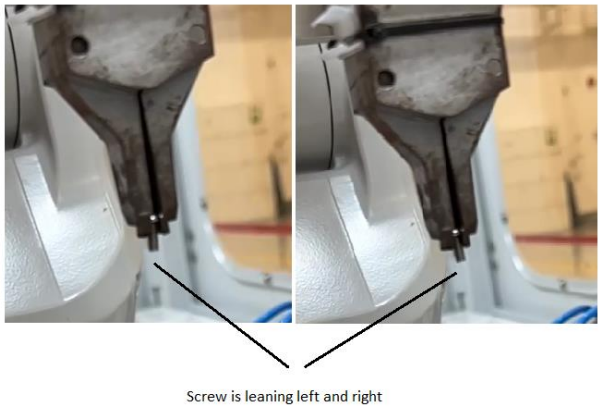


Figure 4.11: Screw shaking due to eccentricity.

To solve this problem, I will adjust the screwing positions so that they are no longer 5mm above the workpiece but instead touching it, which means the duck's beak part will now touch the workpiece. Since when the beak opens, the screw starts to wobble and if it remains 5mm away from the workpiece, the screw will continue to wobble within that range. And after further discussion with Mr. Nguyen Minh Toan, head of this station, I decided to adjust the tool further. The new one will have the duck's beak part connected with a spring system, ensuring elasticity between the collision of it and the work-piece and prevent those two from getting broken. After designing and machining, the result for the new tool head is as follows:

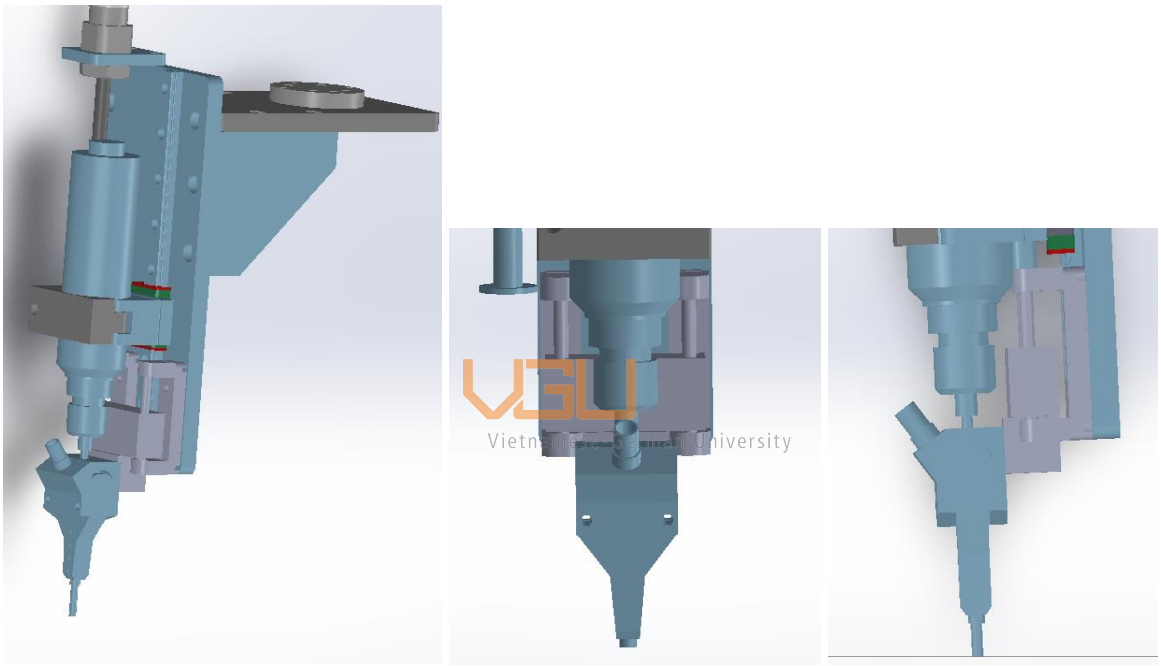


Figure 4.12: Designing of the new tool head using SolidWorks.

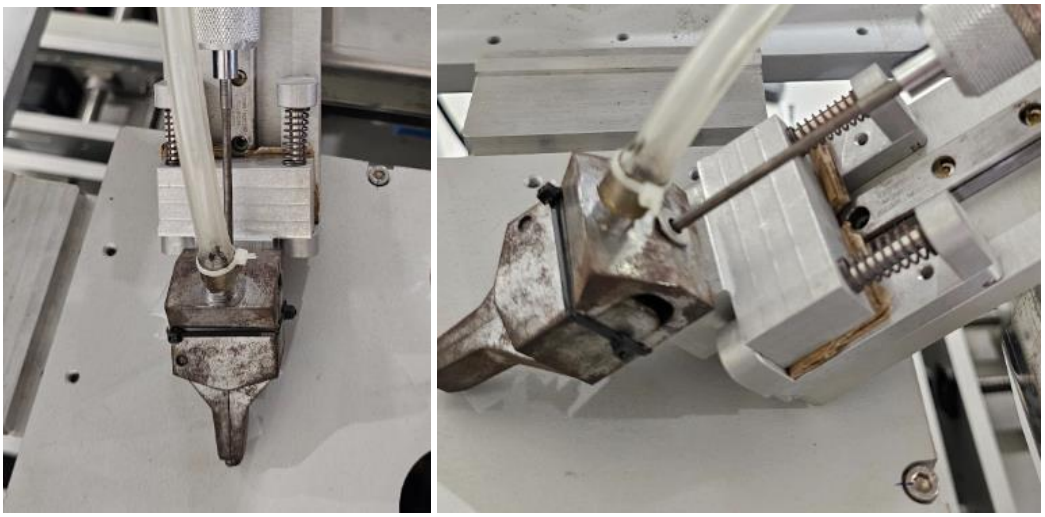


Figure 4.13: Real product after machining.

After finishing setting the new tool on the robot, I once again teaching the new work positions using the FlexPendant and after that running the evalution. This time, the result came out almost perfect as in the next 15 turns, only happen two miss positions and from the error of "broken screw". And to fully get rid of this simple problem, I decided to replace all the current screws into completely new ones. The result later on has been accepted by Mr Nguyen Minh Toan, head of the project.

VGU
Vietnamese-German University

4.2 Results before and after the evaluation and implementation

It can be seen that the results I obtained after evaluating and optimizing the station were beyond expectations. This means that I have correctly assessed and identified the issues and provided reasonable solutions. And if these solutions are not sufficient in the long term, at least I have identified the correct direction to pursue, which will make long-term optimization easier to implement. The figure below shows the difference before and after optimization.

Evaluation No	Actions taken	Succesfully driven screws	Unsuccesfully driven screws	Ratio
Initial result	None	189	392	48.21%
First evaluation	1. Checking every screwing position and screw's condition	265	420	63.10%
Second evaluation	2. Fixing fixture's mechanism	401	420	95.47%
Thrid evaluation	3. Manufacture new tool's head	418	420	99.52%

Figure 4.14: Result before and after optimizing processes

Chapter 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

After a long period of research, practice, and applying the skills I learned, the project 'Operating and Optimizing the Screw-Driving Efficiency for ABB Robotics' Demo Automatic Screw-Driving System' has completed. Although there are still some shortcomings and a few things that can be improved, the project has fundamentally achieved the initial objectives. And above all, during the implementation of this project, I had the opportunity to learn many new things that can benefit me greatly in the future.

Besides reinforcing my knowledge about devices like PLCs, sensors, and programming languages, I also learned a lot about the structure and operation of industrial robots in general, and ABB robots in particular. This knowledge is essential and extremely important as we are in the era of Industry 4.0 and looking towards the future, where automation will become highly popular and widely applied.

5.2 Future work

Since this is a demo station, the potential for further improvement of this project is still very significant. In the future, instead of using a system of fixed positions like we currently do, we could implement a vision system to optimize the station for various types of workpieces. Moreover, a full re-design of the tool can also be considered in order to decrease the size and weight, and fixing minor caussing the tool head’s to rotating eccentric. But overall, the project has meet it’s basic requirement.

REFERENCES

[1] David Peterson, 2023, "*Origin Story: Meet Unimate, the First Industrial Robot".*

Meet Unimate, the First Industrial Robot

[2] Abby, 2023, "*Shakey the Robot Explained: Everything You Need to Know".*

Shakey the Robot Explained: Everything you need to know

[3] Alex Misiti, 2020, "*A History of Industrial Robot*.".

History of Industrial Robots

[4] ABB Robotics, 2024, " *Product Manual: IRB 1100"*

Product Manual: IRB 1100

[5] ABB Robotics, 2024, " *Product Specification: IRB 1100"*

Product Specifications: IRB1100

[5] ABB Robotics, 2023, " *Application Manual: Scalable I/O"*

Application Manual: Scalable I/O

[6] ABB Robotics, 2020, " *Operating Manual: OmniCore*"

Application Manual: OmniCore

[7] ABB Robotics, 2022, " *Product Manual: OmniCore E10"*

Operating Manual: OmniCore E10

[8] Nguyen Luu Minh, 2022, "*Similarities and differences between Profinet and Ethernet."*

Profinet vs Ethernet

[9] Duy Nhat, 2024, "*Ladder Logic/Ladder Diagram.*"

Ladder Logic/Ladder Diagram

[10] ABB Robotics, 2024, "*Product: PM583 – ETH"*

PM583-ETH | ABB

Vietnamese-German University