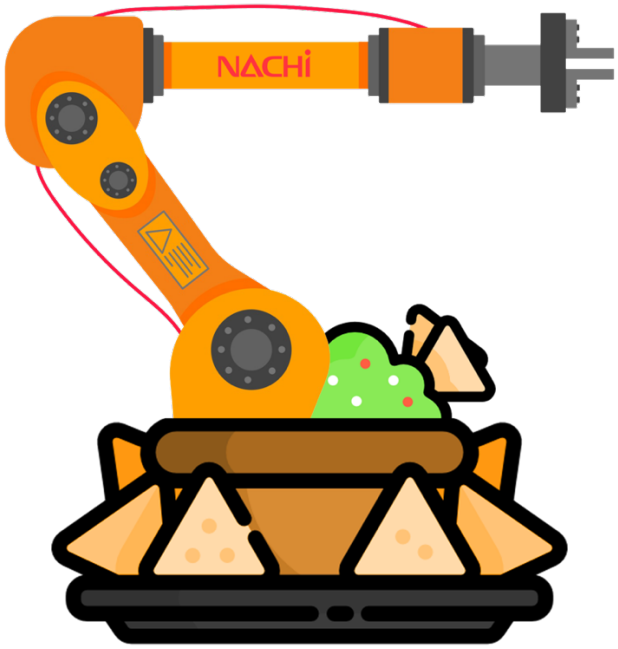

Nachos

An Interface for Nachi Robotics Manipulator Model: ST133TF-01 (AX20 Controller)

(7043 words)

Hadi NAVABI
22228685

Supervisor:
Prof. Thomas Braunl

Table of Content

Abstract

Following the donation of the Nachi robotic manipulator to the University of Western Australia, a great prospect for educational and research activities for students, supervisors and lecturers were created. The Nachi robotics manipulator, with a payload of 133 Kg, creates an esteemed distinction between UWA and competing universities, and if promoted correctly, could generate interest amongst students of Mechatronics, Robotics and Automation.

To provide accessibility and easability to the wide diversity of individuals that may be interested in working with Nachi, and due to their wide range of experiences coming from different scientific fields, industries, and backgrounds, and due to the current complexities surrounding the operation of Nachi, there is a clear need for a simple method of communication. There should be clear and well-defined solution for all individuals to be able complete their tasks easily and independently. Hence the birth of “Nachos”.

Nachos, also known as NachiOS (Nachi Operation System), is a refined and simple to use interface software. Its objectives are to bring together and link Nachi’s internal functionalities into a formal and understandable user interface. It aims to provide assistance to all potential users and individuals of Nachi, regardless of their dissimilar set of expertise.

For its most basic functionality, Nachos uses advance React frameworks to create a single-page multilayered User Interface. The page is divided into a grid of variable and adjustable sections that can dynamically be changed according to the use case and decision of the user. The sections are divided into the following functionalities. First section is an integration of Google’s Blockly, to deliver Block Programming abilities into Nachos. These blocks are analysed using Nachos proprietary internal packages and software and are eventually translated into function calls that is understood by the controller. This Nachi code is then displayed in one of the sections and is ready to be downloaded and copied to the controller.

With the first final release of Nachos, we are aiming to help out all students, lecturers and potential individuals that are interested to work with Nachi, to get a project started quickly and with ease, and to go through and complete the project independently.

https://nachios.vercel.app

1. Introduction

1.1. Nachi System

Nachi is a Japanese industrial robotics manipulator that was donated to the University of Western Australia (UWA), with the intention to create learning and research opportunities for students, lecturers, and supervisors of all fields of study. Nachi robotics manipulator is a unique piece of equipment that creates an esteemed distinction between UWA and competing universities, and if promoted correctly, could generate interest amongst students of Mechatronics, Robotics and Automation, and other engineering fields.

Table 1 Model and Full Name for Nachi robotics manipulator and its controller.

Robot:	**Nachi ST133TF-01 Robotics Manipulator**	**Figure 1**
Controller:	**AX20 Controller with Teach Pendant**	**Figure 2**

Nachi system comprises of two main parts, Nachi ST133TF-01 Robotics Manipulator ("Nachi" henceforth), and the Nachi's AX20 Controller ("Nachi Controller" henceforth). These two parts are wired together to perform top-notch performance with quick movements and a payload of 133 Kg. The entire system runs on 3-phase, 420 volts, and a max current of 30 amps (**Figure 7**).

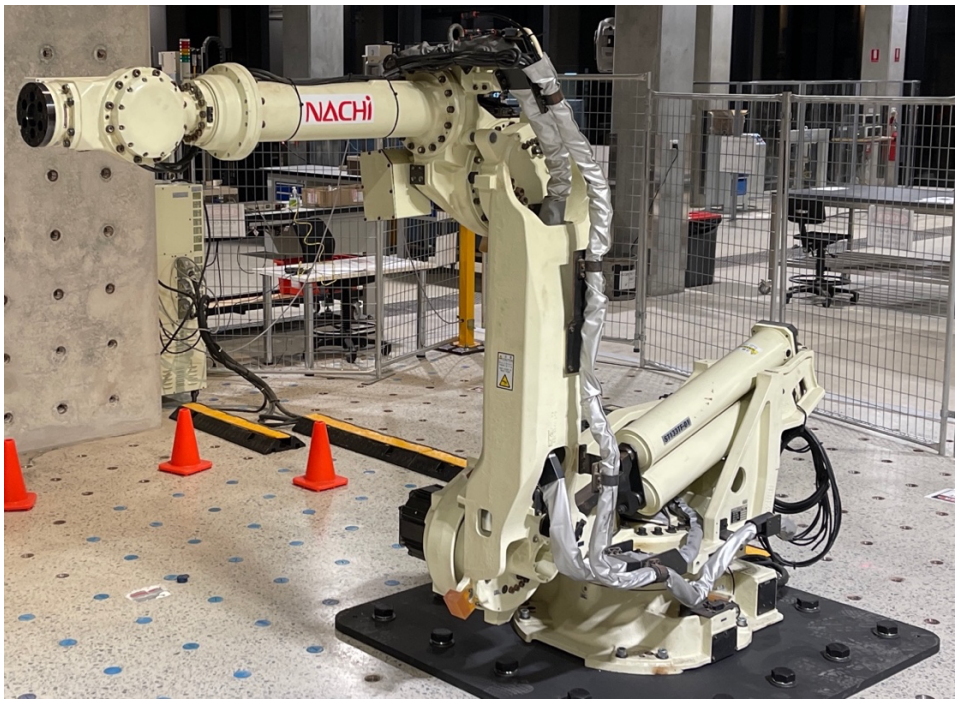

Figure 1 Nachi ST133TF-01 Robotic Manipulator ("Nachi"). Built by Nachi Robotics Systems in 2012. Currently located at Ezone building in UWA.

Figure 2 Nachi AX20 Controller with the Teach Pendant ("Nachi Controller"). Nachi Controller is used to provide power to the robot and to control every aspect of the robotic manipulator. Nachi Controller have the following functions: manual control, software limits, program creation, program conversion, program execution, IO signals, etc…

1.2. Nachi ST133TF-01

Nachi's company, the Nachi Robotics System Inc., has many variants of products in its arsenal. They have been in business since 1928 [1] and have been producing high quality robotics manipulator since 1969 [1].

UWA's robotics manipulator is Nachi ST133TF-01. The model number of Nachi denotes the robot's specifications. **Figure 3** shows the meaning behind each character. Nachi's robot type is "shelf mounting", which means that the robot is intended to be mounted at higher heights on pre-existing shelves, and it can reach lower levels than its base (**Figure 4**). Additionally, the maximum payload authorised is 133 Kg as specified by the model number and the manuals [2].

Schematics and reachability of Nachi are also of great importance. Maximum reachability of the robot is limited by the constraints of the schematics; however, it is also limited by UWA's safety guidelines and the proximity of safety fences to the robot. Hence, reachability is variable from time to time and is due to change. Pure schematics of the robot is provided in **Figure 6** as a reference. [2, p. 1-6]

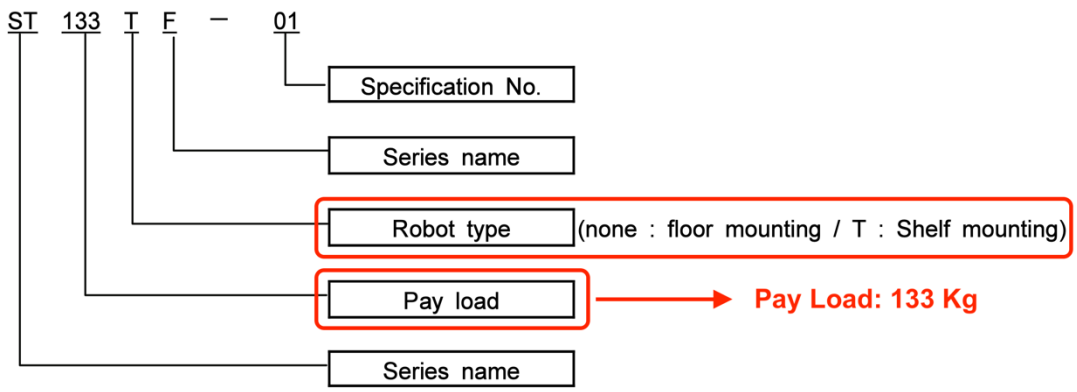

Figure 3 The model number of UWA's Nachi Robotics Manipulator [2, p. 1-1]. The robot is "Shelf mounting" and has a payload of 133 Kg.

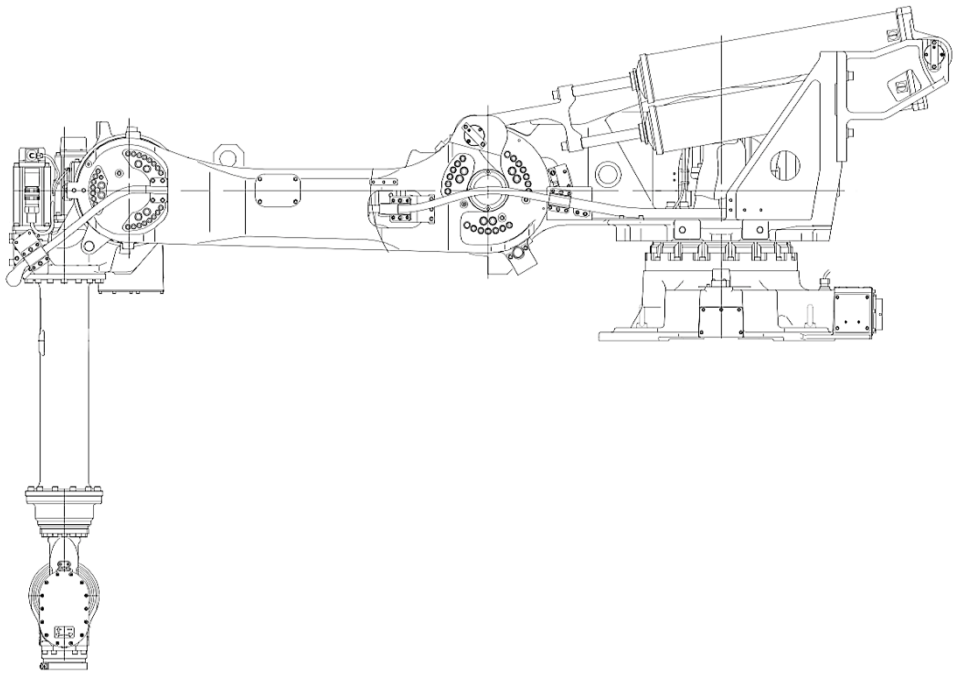

Figure 4 Nachi's home position [2]. As the robot is "Shelf mounting" by design, it needs to be able to reach lower positions, hence the default home position accounts for this.

Nachi comprise of number of joints that enable movement. The number of joints denotes the degrees-of-freedom that the robot has. Most robotics manipulators have six rotational joints, resulting in six degrees-of-freedom. Similarly, Nachi can reach every point in the 3D space that is within its operating range. This reach has been made possible by the first three rotational joints (J1, J2, J3 in **Figure 5**). The ability to move in XYZ-coordinate system provides the robot with three degrees-of-freedom.

However, being able to simply reach every point is not sufficient for most operations, thus, Nachi has a further three rotational joints at its wrist (J4, J5, J6 in **Figure 5**) which provides the remaining three degrees-of-freedom. This enables the reachability at each point at every angle.

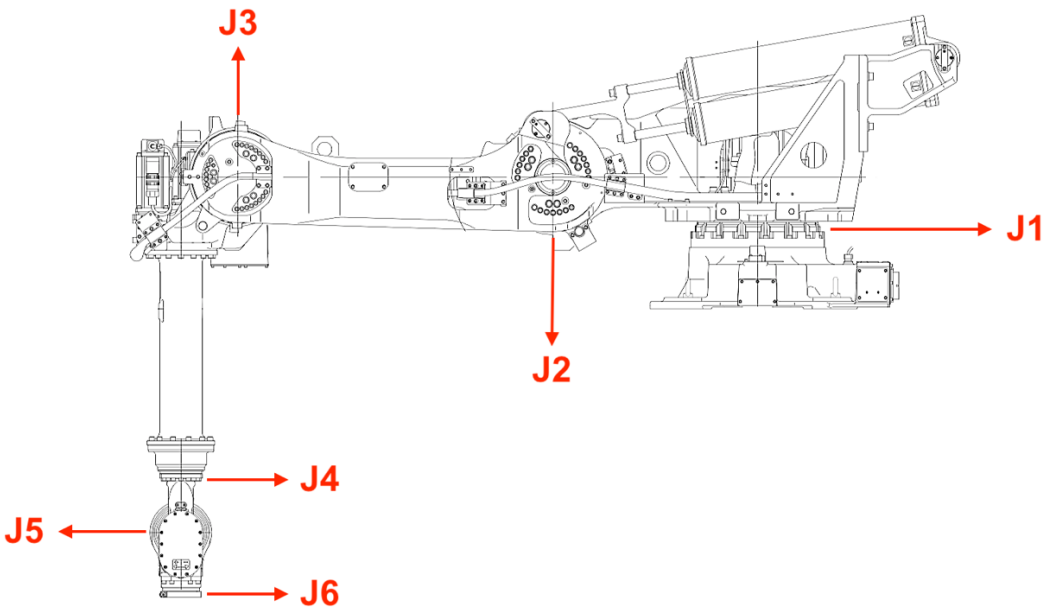

Figure 5 Nachi's six joints. Nachi comprises of six joints which enables the robot with six degrees-of-freedom.

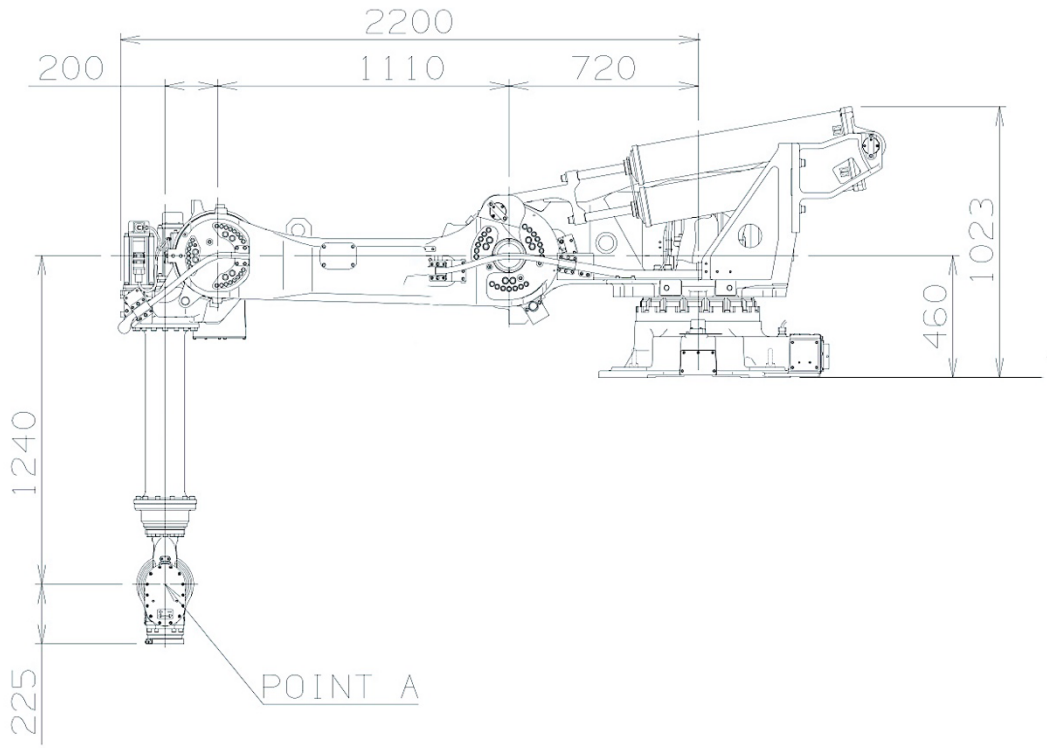

Figure 6 Nachi's schematics. The schematics, dimensions, and measurements of the robot.

1.3. AX20 Controller

Nachi's AX20 Controller (**Figure 2**) is the brain of the system and provides all major and minor functionalities, and commands that Nachi is capable of. Nachi Controller provides power (**Figure 7**) to the robotics manipulator through several wired cables and can controls all aspect of the robot.

Figure 7 Power specification and Nachi Controller's model number. Nachi system (Nachi and Controller) runs on 3-phase, 420 volts and 30 amps of current.

1.4. Teach Pendant

The main point of communication with the Nachi robotics manipulator is through the provided Teach Pendant. The Teach Pendant is the only user interface that is designed for manual and automatic operation of the robot; and with the inclusion of a touch screen, numerous buttons, emergency stop button and break release on the back, all the operations of the robot, is at a touch away.

The first row of buttons on the Teach Pendant provides shortcuts to the most basic functions of the Nachi robotics manipulator. Operations such as changing the coordinate system, changing teach speeds, changing check speeds, toggling between different loop settings, and managing different windows, can be accessed on the Teach Pendant. Additionally, with the press of the "Enabled" button further functionalities, that is represented by the colour green, can be accessed. (**Figure 8**)

The next group of buttons on the Teach Pendant are the manual control and checking controls of the Nachi robotics manipulator. When paired with a light press of the break release buttons on the back of the Teach Pendant, a user can manually take control of the robot, depending on Nachi's motors being turned on and all safety checks passed. According to the selected coordinate system, these group of buttons can behave in different ways. For an instance, when "Joints" coordinate system is selected, "X-" and "X+" buttons will control J1 (**Figure 5**), "Y-" and "Y+" buttons will control J2 (**Figure 5**), "Z-" and "Z+" buttons will control J3 (**Figure 5**), "RX-" and "RX+" buttons will control J4 (**Figure 5**), "RY-" and "RY+" buttons will control J5 (**Figure 5**), and "RZ-" and "RZ+"

buttons will control J6 (**Figure 5**). On the other hand, when “Cartesian” (xyz) coordinate system is selected, the buttons will behave as expected. For example, “X-” and “X+” buttons will change the end effector’s position on the x-axis.

Figure 8 Nachi Controller's Teach Pendant. It’s the main point of communication for the manual and automatic control of the Nachi robotics manipulator. It enables writing, testing, and execution of all programs. Display shows Program #25 and Step #6 is highlighted (in focus).

The final group of buttons on the Teach Pendant provide other basic functionalities of the Nachi robotics manipulator. The most important set of buttons worth mentioning are the Arrow keys, “REC” button, “FN” button, and “Program/Step” button. The Arrow keys help with navigation through the various menus or lists on the Teach Pendant. They can also enable scrolling through different steps in a program. The “REC” button will simply enable the user to record the current position of the Nachi robotics manipulator

and add it to the current active program. “FN” button will open a list of all the available commands. Users can use this large list of commands to program different functionalities and movements, and in practice, automate their workplace. Finally, with the use of “Program/Step” button, the user can jump to different steps in a program. Alternatively, the user can jump to an entirely other program for further programming of Nachi.

1.5. SLIM Language

SLIM (**S**tandard **L**anguage for **I**ndustrial **M**anipulators) is a language designed by Nachi Robotics Systems Inc. for almost all its robotics manipulators. SLIM complies with the Japanese Industrial Standard (JIS) [3, p. 8] and works with both AX and FD controllers [4]. All Nachi’s programming functionalities and commands are expresses in the SLIM robotics language. A SLIM program comprises of a list of steps (**Figure 8**) that includes movement commands and function commands. There are 3 movement commands and about 420 function commands available in SLIM language. Out of the 420 function commands that is available and can be found in the manuals [5] [6] [7], UWA’s Nachi robotics manipulator has a total of 229 function commands accessible.

Function commands can be specified using the “FN***” format where a 1-3 digit number is input into the “***” part (this is called the function number). For example, the “END” function command is specified as “FN92” and is used to end the execution of a running program.

1.5.1. Movement Commands

The three movement commands are “MOVE”, “MOVEJ”, and “MOVEX”, however, the latter is the main movement command and is the one that is generally used for all cases. “MOVEX” (**Figure 15**) is the movement command that is recommended by the manuals and it’s quite comprehensive due to the many required and optional parameters that it requires. These parameters and their order are as below: (* is required, - is optional)

1.	Acceleration	-	(AC=)	0 to 3
2.	Accuracy	-	(A=)	1 to 8
3.	Smoothness	-	(SM=)	0 to 3
4.	Mechanism	*		M1J or M1X
5.	Interpolation	*		P, L, C1 or C2
6.	Position Values	*		
7.	Speed	*		
8.	Tool	-	(H=)	1 to 32

For detailed explanation and some examples for each of these parameters, refer to [7].

In addition to the above parameters, there are also Synchronisation, Configuration and Speed master definition parameters that can be specified. However, due to the current setup of Nachi, no specific use case has been found for these parameters.

An example of the movement command is outlined below and more examples can be found in **Figure 18** and **Figure 19**.

```
MOVEX A=1, AC=0, SM=0, M1J, P, (*,*,*,*,*,*), S=100, H=1, MS
```

1.5.2. Function Commands

In addition to the movement commands, Nachi Robotics Systems Inc. also provides a full set of high- and low-level control flows, such as logical controls, conditional statements, loop controls, function calls, jump commands, input signal checks, and output signals. All of these provided function commands allows customisable order of execution and enables modern and complex programs.

Table 2 32 major examples of Nachi's function commands. UWA's Nachi robotics manipulator has a total of 229 function commands accessible.

SET	LETX	DELAY	CALLP	RETURN	END	ALLCLR	JMP
RESET	LETY	WAIT	CALLPI	RETI	STOP	REM	JMPI
SETMD	LETZ	TIMER	CALLI	RETN	STOPI	PRINT	JMPN
SETM	INPUT	OUT	CALLN	IF	FOR	NEXT	GOTO

1.5.3. Specialised Commands

Additionally, Nachi Robotics Systems Inc. also provides some specialised function commands. These specialised commands provide functionalities that are commonly used in various industries. Palletisations (pick and place), spot welding, servo gun welding, seam welding, conveyor interactions, roller hem, etc. are some examples of the specialised commands on offer. However, after crosschecking commands found in the manuals with available commands on UWA's Nachi robotics manipulator, only palletising functionalities were found accessible.

Table 3 Some examples of specialised commands. Bold is accessible in Nachi robotics manipulator.

PALLET2	SPOT	AS	ASV	SERVOON
PALLET3	ASM	MPS	SEAM	ROLHEM

2. Problem identification

2.1. Overview

The Nachi robotics manipulator is one of the most valuable and unique equipment that UWA owns. That is in terms of pure dollar value (estimated at about A$50,000), and educational opportunities that it can provide to students, supervisors, and lecturers. If promoted well, it has the potential to attract future students and various interesting projects. Moreover, Nachi is a unique technology that not all competing universities in Perth have the privilege to access. The lack of this sort of technology in other universities, creates a great advantage towards UWA, especially in trending majors such as Mechatronics, and Robotics and Automation. However, currently, Nachi is only being used for simple demonstrations to students and is not being utilised for educational or research purposes. This lack of educational opportunities for students is mainly due to complexities surrounding the operation of Nachi, that will be discussed in depth below.

2.2. Diverse user-base

Due to uniqueness and popularity of this kind of technology, Nachi has already attracted several projects, and individuals. However, there is a huge diversity in the individuals that are interested in working with Nachi, as they may come from different scientific fields, industries, and backgrounds. These individuals should not be required to obtain a complete understanding of Nachi and all its intricate details, features, and functionalities, before being able to complete a specific task, as that would be inefficient, time consuming and discourages motivation, innovation, and originality in the individuals.

There should be clear, easy, and simple solution for the individuals with limited understanding about Nachi, to be able complete their tasks easily, independently, and confidently.

2.3. Lack of Information

Another main problem with Nachi robotics manipulator is the scarcity of information regarding its operation, and numerous uncertainties about its functionalities and capabilities. The ambiguities such as (a) unavailability of operational procedures and instructions, (b) unavailability of programming features and capacities, (c) unknown procedures for compiling and execution of a program, and interacting with IO signals, (d) scarcity of access to manuals and resources due to proprietary nature of the product, (e) lack of support due to end-of-life software, has essentially rendered Nachi unusable.

To be able to come up with a structure of information that satisfies almost all these missing information, an extensive attempt at searching for available materials needs to be conducted. Different manuals, instructions and procedures need to be gathered through exploring different avenues of communication with robotics companies and robotics service providers.

2.4. Limited Efficiency of Nachi's Teach Pendant

As the entire capabilities that Nachi can provide, has been made accessible through the Teach Pendant, theoretically, any program, task, and project can be completed through the Teach Pendant. However, the assumption of the Teach Pendant as the only point of communication with Nachi, creates some challenges and discouragement for the individuals that are interested in working with it.

For example, for the development of any task by a team of student, or group of colleagues, the physical proximity to Nachi robotics manipulator is required. This is because, currently, all development must be done through the Teach Pendant, and the Teach Pendent is physically connected to the controller via a cable. This significantly restricts team working opportunities, collaboration, and development of tasks, to the boundaries of Nachi which reduces efficiency to an absolute minimum.

2.5. Lack of Simulation Software

Lastly, the lack of a local simulation software hinders all development and renders the Nachi almost unusable. This is due to the inability to visualise and test code before its execution on Nachi. In professional software development, a simulation software is used to model the real-world object and see the effect of the code on that object. A simulation software is both essential and necessary for Nachi robotics manipulator, as irreplaceable damages can occur with just smallest mistakes in code.

3. Nachos

Nachos, or NachiOS (Nachi Operating System) is an interface and simulation software for the Nachi robotics manipulator that aims to provide easy accessibility and effortless programming of Nachi for all students, lecturers, and supervisors. Nachos is the proposed application for this project and it's an all-in-one software that intends to solve almost all the previously identified problems. However, to be able to start designing, a full stack of resources and manuals needs to be gathered, analysed, understood, and tested. Additionally, a full design process needs to be conducted.

3.1. Resources

Before the development of an all-in-one software as the solution to all the identified problems, a deep understanding of Nachi robotics manipulator and all its intricate details and functionalities is required. Thus, gathering and elicitation of all the available resources, manuals, and information is one of the most critical tasks in this project. Through communication with various robotics companies and service providers in Australia and abroad, many resources and information were gathered and packaged for future reference.

A relatively comprehensive list of resources is now available for any individuals that may be interested in working with the Nachi robotics manipulator. This package of information consists of numerous and various manuals, software, and marketing resources (photos and videos).

Nachi comprises of many parts and functionalities. The gathered manuals have been organised accordingly into (1) Nachi robotics Manipulator, (2) Nachi's AX20 Controller, (3) Programming functionalities, (4) UWA Safety Instructions and Basic Operations, (5) Other manuals and software. Additionally, all available printed manuals were scanned and prepared for electronic use by converting them into searchable documents.

A full list of the manuals and elicited resources can be found below:

1. **NACHI ST133TF-01**
    1. NACHI ST133TF-01 Maintenance Service Manual
    2. Model Specification
    3. Schematic - Home Position
    4. Schematic - Joints
    5. Schematic - Measurements
    6. IO Signals

2. **NACHI AX20 Controller**

1. NACHI AX Controller Operating Manual – Basic Operations Manual
2. NACHI AX Controller Operating Manual – Ethernet
3. FD Controller Instruction Manual – Robot Language (SLIM)
4. NACHI AX Controller Operating Manual – Palletize Function
5. NACHI AX20 Controller Maintenance Manual
6. Model Specification on Controller

3. Programming

1. Controller Instruction Manual – Command Reference vol 1 (FN0-356)
2. Controller Instruction Manual – Command Reference vol 2 (FN360-597)
3. Controller Instruction Manual – Command Reference vol 3 (FN598-893)
4. Spreadsheet of All Command

4. UWA Safety Instructions and Basic Operations

1. 2021 Nachi Safety – v1.3
2. Code Conversion and Compile

5. Other Manuals and Software

1. FD on Desk – Windows Software
2. AX on Desk – Operating Manual
3. Lock (TLS1-GD2)
4. OnRobot Grippers for Nachi

3.2. Design Process

3.2.1. Requirements and Specifications

Due to the complete lack of existing software and resources, a comprehensive design process and planning was to be completed for Nachos (NachiOS). Considering the key requirements and constraints of the project and analysing them is always crucial and necessary. These requirements help guild the project in the correct direction and allows for an easier evaluation of the final design.

Through collaboration with the project stakeholders, specifically the project supervisor, numerous design requirements were identified. The identified requirements and constraints for the project are outlined below:

1. Nachos shall use Block Programming methodology as the user interface, so that individuals can easily find and use the functionalities to generate code.

2. Nachos shall be able to convert instructions and generate code that is understood by the Nachi robotics manipulator and its controller.

3. Nachos shall be implemented as a client-side application.

4. Nachos shall be accessible on both Windows and MacOS.

5. The final design shall be implemented in such a way that it contains most of the basic functionalities of Nachi system.

6. The final design shall display the generated code.

7. The final design shall enable downloading of the generated code into an output file.

8. The final design shall output the correct filename.

9. The final design shall output correct file formatting.

10. The final design shall enable saving the current progress and resuming at a later time.

11. The final design shall sufficiently be tested and be robust.

12. The final design shall include an error handling system that throws specific, helpful, and sufficient warnings messages to avoid crashes.

3.2.2. Block Programming Language (Google's Blockly)

Block Programming is a visual-based programming language that is designed for individuals with limited programming knowledge and expertise. It allows for easy visualisation of all the available features of a language and enables easy drag-and-drop interactivity of blocks for fast and easy development of any program. The blocks represent the most fundamental functionality of the language and can be grouped, attached, or nested together to create a larger program or functionality. From a specific arrangement and grouping of blocks a program is created and with a more complex organisation of blocks, programs can get more advanced and complicated. Subsequently, these blocks can be converted to generate a textual representation of all the blocks.

In the preliminary research phase of this project, numerous Block Programming languages that seemed suitable for the requirements of the proposed system, were identified. Blockly, Scratch, Snap, EduBlocks and more are great examples that sounded promising as the base for Nachos. However, upon further investigation and careful examining of their documentations, all of them were either completely based on Google's Blockly or were a close implementation of it. Additionally, Google's Blockly is the most well-designed, advanced, and the most used block programming language that is currently available and accessible for developers. Hence, the decision to use Blockly as the underlying implementation of Nachos was easily reached.

Google's Blockly is a free and open-source JavaScript library for block programming. It comes with extensive documentations, developer tools, forums, examples, and video tutorials for development. The provided examples are very comprehensive and is a great pathway for quick learning of all the features that is implemented and is accessible for use. Additionally, Blockly Factory or Blockly Developer Tool [16], is the perfect tool for the Blockly programming language as it enables developers to automate most of the block creation and code generation for their Blockly implementation.

3.2.3. Framework (ReactJS)

Due to the specific requirements of Nachos as a client-side software, and its requirement for accessibility on both MacOS and Windows, and due to the limitations of Blockly as a JavaScript library, the decision for the Nachos application being implemented as web application, was rather forced.

However, Nachos, as a JavaScript implementation of Blockly, could have been implemented in almost all the popular frameworks such as React, Vue, Angular, and even plain JavaScript. Furthermore, simple examples and implementations for all the

named frameworks above was also provided by Blockly. Therefore, the decision to choose a framework was rather difficult. Following investigation through Blockly's examples and documentation, and research for available resources, and taking into account the compatibility of both Blockly and ReactJS with Node Package Manager (npm), and considering my own experience and expertise in ReactJS, the decision for ReactJS as the main framework for Nachos implementation was attained.

3.2.4. Conclusion

For the implementation of the proposed software for this project, due to the lack of any current software, and due the high capacity for future development of Nachos by other students, the emphasis on commonly used and well-known languages for the implementation of Nachos was quite high. With the decision of Nachos being implemented in JavaScript, Blockly and ReactJS this goal was also achieved. Therefore, any future development and adding of new functionalities to the Nachos system will hopefully have minimum learning curve and easy to start.

3.3. Implementation

3.3.1. Overview

Nachos (**Figure 9**) is the proposed software that is implemented specifically for the Nachi robotics manipulator and all its capabilities. It is a web-based application that uses ReactJS as its main framework and Google's Blockly for its core feature of being a block-based programming application. Nachos consists of a two-page architecture for comfortable navigation and simplicity.

Figure 9 Nachos (NachiOS). It's an interface and simulation software for the Nachi robotics manipulator that aims to provide easy accessibility and effortless programming for Nachi robotics manipulator.

3.3.2. Two-page Architecture

For the ease of navigation and simplicity of the entire system, a simple two-page architecture was implemented for Nachos application. The home page or the welcome page (**Figure 10**) is a well-designed page that welcomes the user to the application and guides them to their destination. From the home page, the user can get to the coding page, access the available documentations, and get help.

After getting redirected to the coding page, the user is introduced to the Nachos core functionality, which contains the Blockly implementation and can generate code that is understood by the Nachi robotics manipulator. The coding page comprises of three main sections. These sections are resizable according to the user's preferences and can be arranged in a split manner to contain both the Blockly workspace and the code output section, or it can be arranged in full screen for larger programs.

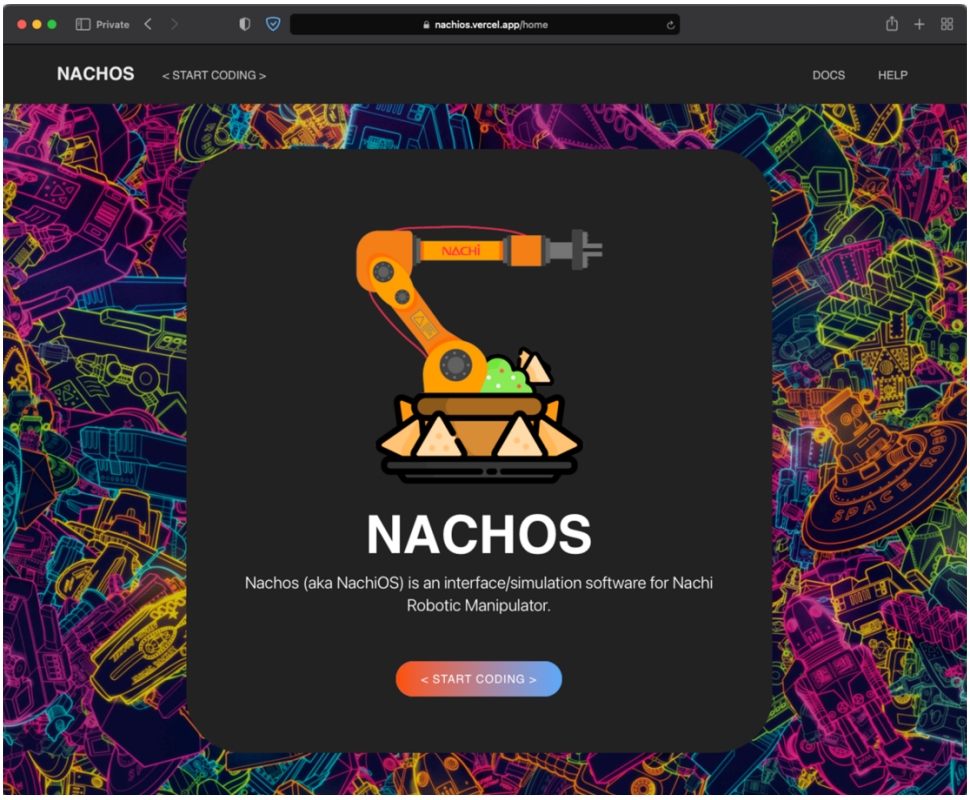

Figure 10 Nachos home page. It has the sole purpose of welcoming the user and guide them in the right direction.

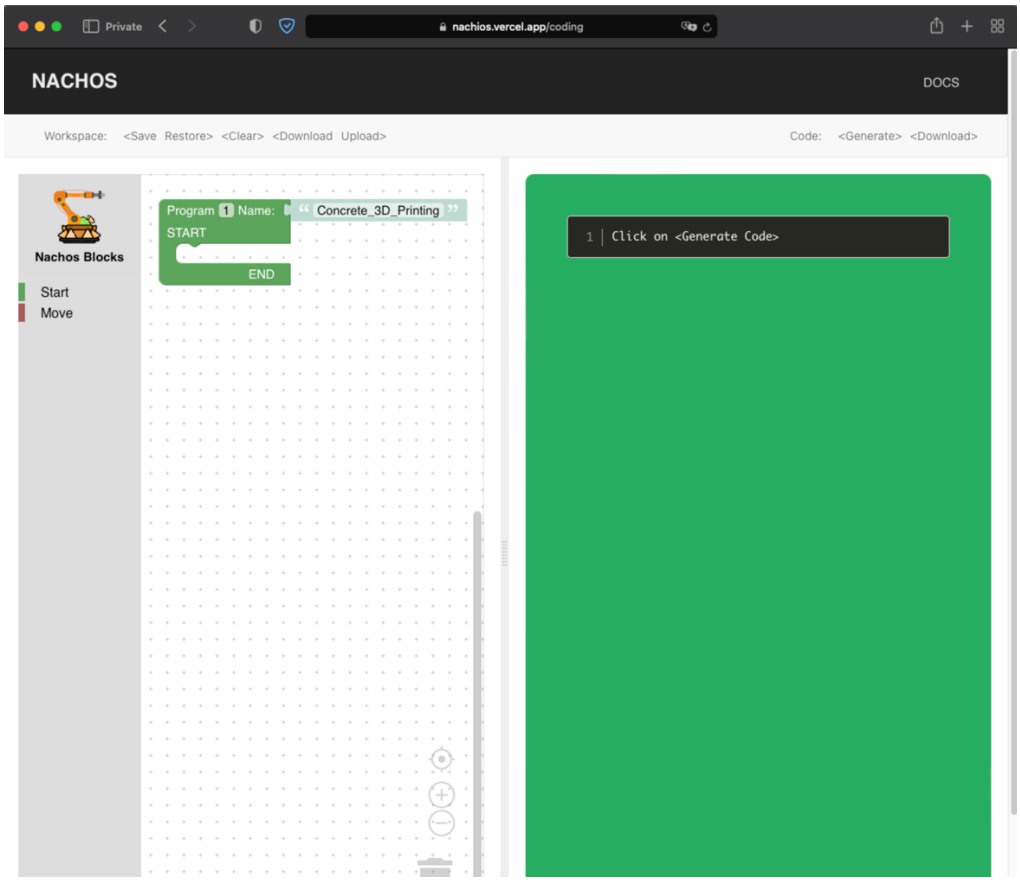

Figure 11 Nachos coding page (split view). It contains the Blockly implementation which is the core functionality of Nachos. The sections are resizable according to the user's preferences.

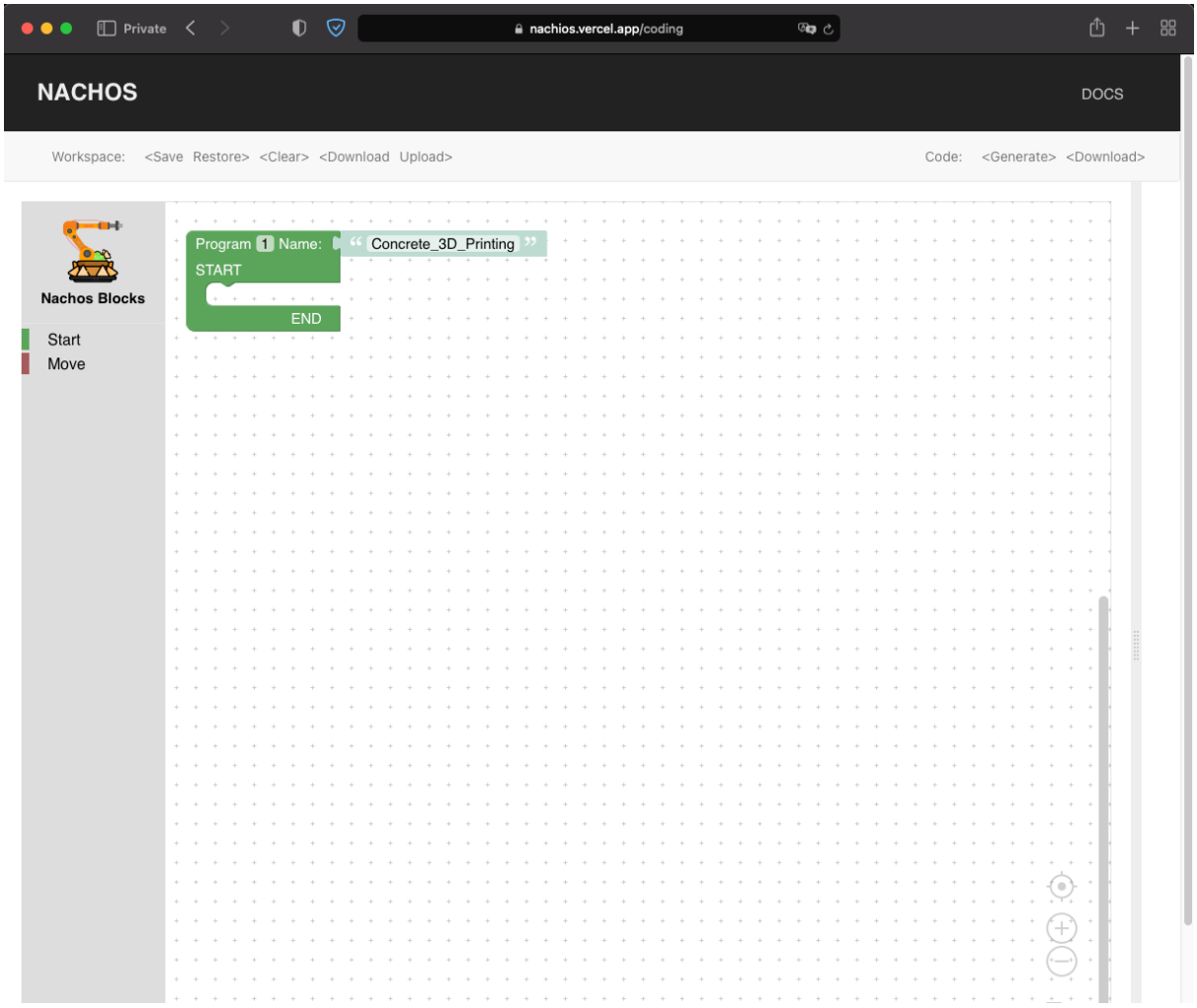

Figure 12 Nachos coding page (full screen view).

The most important section in the coding page, is the workspace, which is the block-based coding area that is powered by Blockly. The Blockly workspace contains two sections, the grey toolbox on the left, which contains all the available blocks that is usable for programming and is sorted into relevant categories, and the dotted white workspace that the blocks can be dragged and dropped into. Any block that is located within the white workspace will be converted into Nachi code and will be displayed on the code output section. The default block in the workspace is the "START" block, which is the main block and encompasses all other blocks (**Figure 13**). Essentially, the "START" block contains an entire program within it and gets terminated by the "END" command. A given workspace can only contain one "START" block and to avoid errors when compiling the code on the Nachi Controller, it is strongly recommended to keep all blocks within the "START" block. Additionally, the "START" command, takes two inputs/parameters, that is, the number of the program, which will be used for the output file during downloading of Nachi code, and the name for the program, that will be used as a comment at the beginning of the output file.

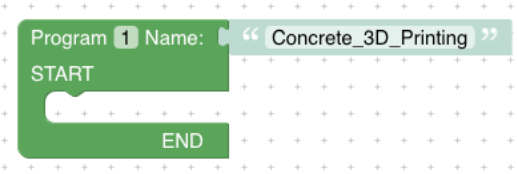

Figure 13 The "START" block. It's the main block that represents a program, encompasses all other blocks, and is terminated by the "END" (FN92) command.

The next section is the code output section, that is represented by the colour green. As explained previously, all the blocks that have been dragged and dropped into the white workspace will be converted by the "**NachosGenerator**" into Nachi code.

The last section is the navbar at the top of the page and contains all the controls and buttons required by the user. For short sessions, there are the "save" and "restore" buttons that enable quick save and quick restore of all the blocks that are located inside the workspace. For longer breaks, there are the "download" and "upload" buttons that will output an XML file containing all the current blocks. This can also enable version control of your code. To upload, click on the "upload" button and paste back the content of the XML file. Finally, to convert and generate Nachi code click on the "generate" button, when the program is finalised, download the generated Nachi code by clicking on the "download" button to get correctly formatted output file with the correct name.

Nachos application, with its block-based programming approach and simple two-page navigation is both easy-to-learn and easy-to-use.

3.3.3. Flow of Execution

Implementation of Nachos with a ReactJS framework follows a specific order of execution. The application starts at "**index.js**", which is then passed onto "**App.js**". "**App.js**" contains the routing of the entire web application, "/home" gets redirected to the home page or "**Home.jsx**" of Nachos, and "/coding" gets redirected to the coding page or "**Coding.jsx**".

As expected, the coding page has a much larger complexity than the home page and will subsequently load "**Blocks.jsx**" which contains all the defined blocks in the Nachos application. The definition of these blocks is either in the Blockly library itself or defined in the "**NachosBlocks.jsx**".

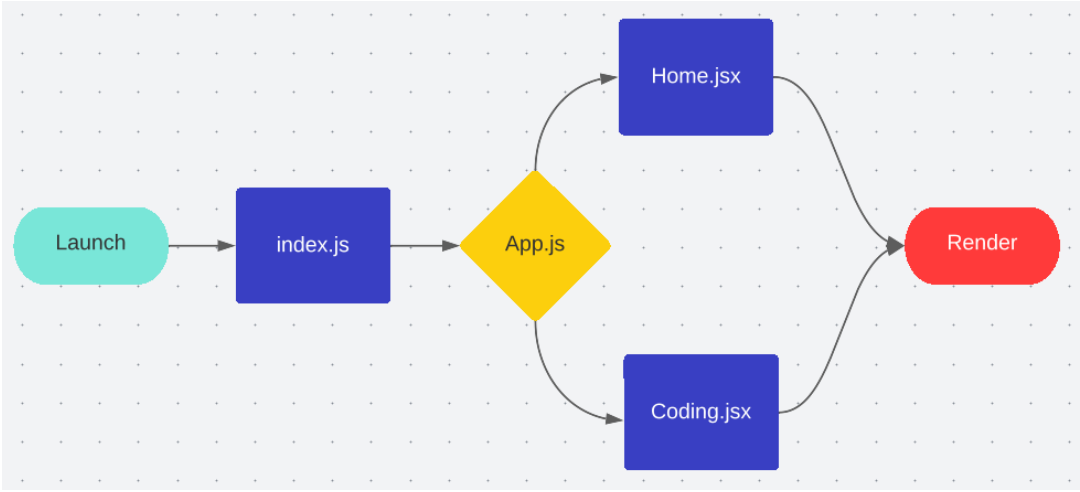

Figure 14 Flow of execution of Nachos.

3.3.4. NachosBlocks

To be able to implement the features and capabilities of the Nachi robotics manipulator, a customised blocks needs to be defined for each and all of movement and function commands that can be found in the manuals and the Nachi itself. Not only this requires an intricate knowledge and understanding for each command, but also requires an exhaustive testing regime for the command that is going to be implemented. Let's assume the "MOVEX" movement command that was explained previously (**Figure 15**). The "MOVEX" command comprises of many parameters as inputs. For a robust system such as Nachos, a full combination of the parameters was tested, compiled, and verified through the Nachi Controller. This process is very time-consuming and specific, each command will eventually have a list of requirements and specifications that needs to be followed to avoid syntax errors during compiling and runtime errors during runtime.

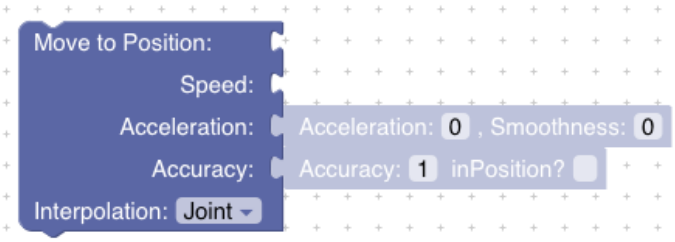

Figure 15 The "MOVEX" block. It's the main movement command and comprises of many parameters for input.

Continuing with the "MOVEX" block, it contains four value inputs and one dummy inputs [17]. "Position" value input can only accept two other block types, "JOINT" block (**Figure 16**), and "POSE" block (**Figure 17**). The "JOINT" block requires six joint parameters (**Figure 5**) as inputs, and the "POSE" block requires three values for translation and three values for rotation in XYZ coordinate system.

Additionally, the "MOVEX" command has "Speed" value input that can enable attachment of three speed type blocks. According to the manuals, the speed can be specified in (a) TCP linear speed in "mm/s" unit, (b) time to reach next position in "seconds", and (c) a percentage of maximum power "%".

For detailed explanation and more examples for the other parameters, refer to [7]. Furthermore, the code definition for each block can be found in "**NachosBlocks.jsx**".

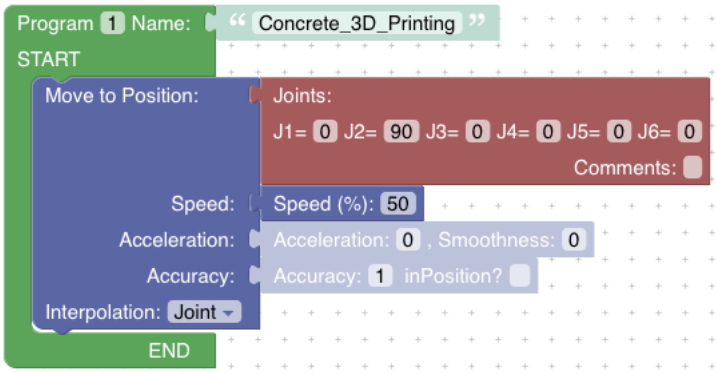

Figure 16 "MOVEX" block with "JOINT" block attached as an input to the "Position" value input. "JOINT" block requires six joint parameters (**Figure 5**).

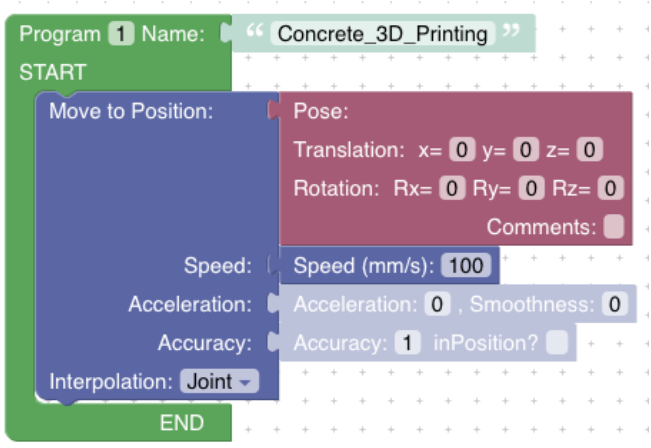

Figure 17 "MOVEX" block with "POSE" block attached as an input to the "Position" value input. "POSE block requires three values for x, y, z coordinate system, and three other values for Rx, Ry, and Rz.

3.3.5. NachosGenerator

Defining the block according to the manual is not sufficient for the conversion and generation of Nachi code. Each Blockly block needs to have specific rules and instructions to be able to convert the block and all its inputs to a specific language, in this case Nachi code. "**NachosGenerator**" is the name dubbed for these rules and instructions that have been implemented for the Nachos application.

Similar to all other blocks, the “MOVEX” block requires specific NachosGenerator rules and instruction to be able to convert and generate the correct Nachi code. The general method is to first convert and generate the code for the input blocks. The input blocks are the “Position” block, “Speed” block, “Acceleration” block, “Accuracy” block, and “Interpolation” value. After the conversion of these blocks into Nachi code, the outputs are passed to the “MOVEX” block, then an overall Nachi code is generated as the output for the “MOVEX” block.

Specific implementation for the NachosGenerator rules and instructions for the “MOVEX” command can be found in “**NachosBlocks.jsx**”.

3.3.6. Generated Code

After the completion of editing and rearranging of blocks in the Blockly workspace, the user may want to generate the code. By clicking on the “generate” button, the NachosGenerator gets triggered, and the conversion and code generation begins in the background of the Nachos application. After the a few milliseconds, depending on the size of the program, the generated code is displayed in the code output section.

```
1 ' Concrete_3D_Printing
2 MOVEX A=1, AC=0, SM=0, M1J, P, (0, 90, 0, 0, 0, 0), R=50, H=1, MS
3 END
```

Figure 18 The generated Nachi code for Figure 16. The generated code can be downloaded locally by clicking on the “download” button in the navbar.

```
1 ' Concrete_3D_Printing
2 MOVEX A=1, AC=0, SM=0, M1X, P, (0, 0, 0, 0, 0, 0), S=100, H=1, MS
3 END
```

Figure 19 The generated Nachi code for Figure 17.

3.3.7. Conclusion

The implementation of Nachos application can now enable easy accessibility and effortless programming of the Nachi robotics manipulator for all students, lecturers, and supervisors. Nachos uses widely accessible JavaScript and ReactJS, so that the development of Nachi programs can be done remotely and offsite. This can significantly help with increase productivity and teamworking opportunities. Furthermore, the Teach Pendant is no longer the only point of development, and individuals can more easily start and finish a project independently. All the core functionality of the Nachi robotics manipulator is now readily available and visualisable through the implemented Blockly blocks, and the intricate knowledge and a full understanding of the Nachi robotics

manipulator is no longer necessary. Even for complex programs and functionalities, with the elicitation of the many manuals and resources, Nachi is no longer a black box and can easily be understood through a light reading of the materials that is provided.

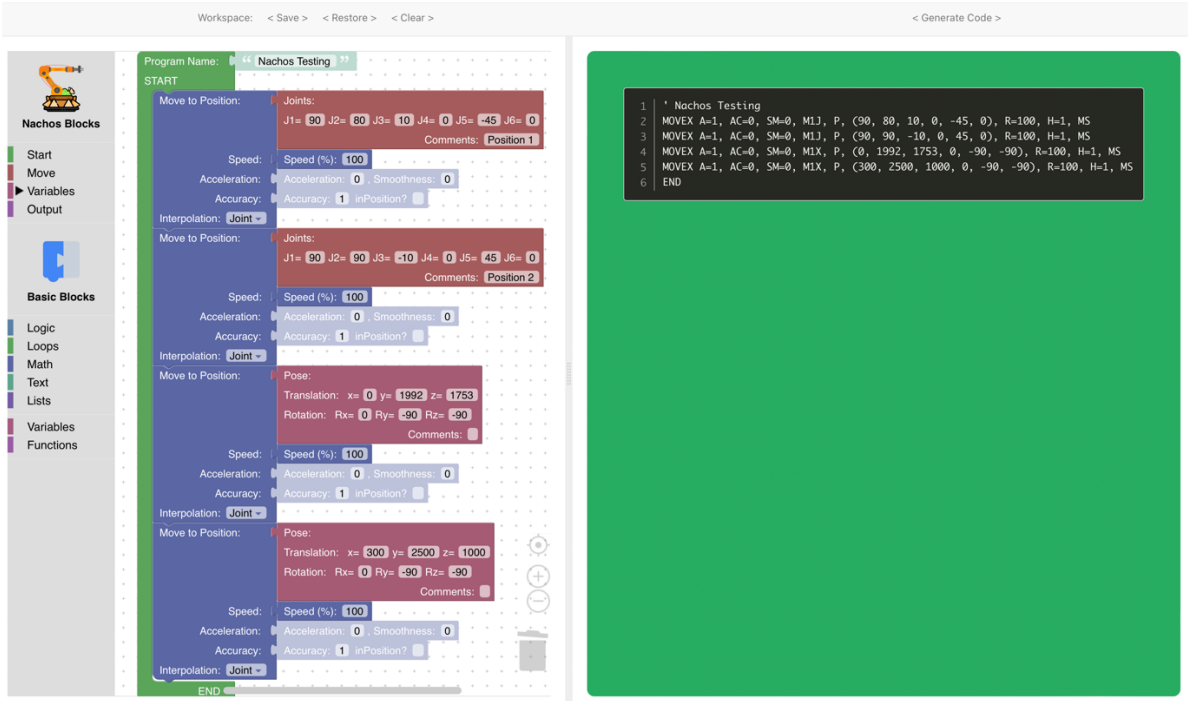

Figure 20 Full example of a simple program with its generated code. Program was coded through the Blockly workspace. The code was generated via the NachosGenerator. The generated code was compiled on the Nachi Controller without any syntax errors. The compiled code was then executed on Nachi robotics manipulator successfully.

4. Projects

4.1. Previous Projects

The extension of the completed projects and previous works on the Nachi robotics manipulator is heavily limited by the problems that are identified previously. Up until the project handover, only the basic setup and instructions were established, and limited functionalities, such as manual movement of the robot, creating small programs and its automatic execution, was achieved using the Teach Pendant.

4.2. Concurrent Projects

4.2.1. Concrete 3D Printing

With the handover of the Nachi robotics manipulator as a research project, some on-hold projects reignited and were in the works again. A major example of this, is the “Concrete 3D Printing” project. A group of postgraduate civil engineering students were attempting to use a robotics manipulator to 3D Print a concrete barrier. In collaboration with me, they used various different programs, such as Slic3r [8] and

RoboDK [9], to generate a specific 3D Printing program for Nachi robotics manipulator. We effectively managed to accomplish their goal, after hours of collaboration, learning different Nachi's procedures and instructions, and teamwork across different UWA departments and management.

Figure 21 Final Concrete 3D Print with the Nachi robotics manipulator. After many attempts and overcoming challenges, the "Concrete 3D Printing" project were successful.

4.2.2. Relocation of Nachi

Another project that required my involvement was the planned temporary relocation of Nachi robotics manipulator. The newly found manuals [2] were used to anchor the robot in the correct way so that the weight was evenly distributed.

4.3. Future Projects

4.3.1. UWA's new robotics unit

The splitting of UWA Robotics unit (GENG5508) into two is a potential prospect that was inspired due to the massive educational opportunities that Nachi can provide to students. Currently, the Robotics unit's curriculum contains material from Mobile Robots, that is thought through EyeSim simulation software [11], and Robot Manipulators, that is thought through UR5 robots.

The addition of Nachi to this curriculum would enable the unit to be split into two units of "Robotics: Mobile Robots" and "Robotics: Manipulators". The new unit will introduce the students to Nachi robotics manipulator, and the possibilities of true automation and

large-scale assembly lines. The unit will be thought through Nachi and Nachos software, which will be an ideal resource for students to get insight into the world of manufacturing and industrial robotics manipulators.

4.3.2. Foam Design by UWA Design

With collaboration with UWA Design and Robotics, a new project was emerged. This project aims to use the newly designed software (Nachos) and all the new information that has been found, to generate a program to move pieces of foam timber and to create an artform.

The project aims to utilise a unique vacuum gripper by UniGripper for pick and place of the timbers [10]. However, for the gripper to be integrated with the program the I/O signal commands need to be tested and implemented into the Nachos application.

5. Future Work

5.1. I/O Functionality

The IO (Input and Output) functionality of Nachi enables a massive opportunity for the integration of various physical buttons and real-world sensors into the Nachos application, as well Nachi programs. Currently, specific functionalities of the Nachi robotics manipulator are limited by the lack of physical inputs and sensors from the real-world and conditional statements cannot be influenced by the outside world which limits the functionality significantly.

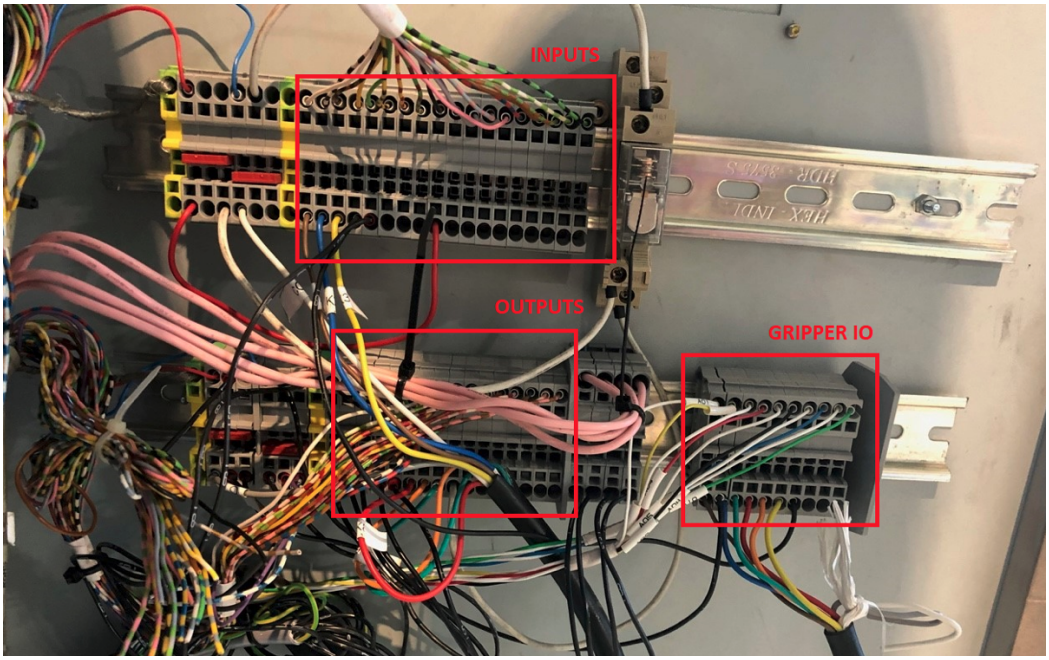

Figure 22 I/O signal arrangement. 5-volt input and output signals can used for use via sensors and other functionalities (theoretically).

As a future work, the input and output signals into the controller needs to be analysed and exhaustively tested. An electrical engineering student is recommended for this research project due to highly complex wiring within the Nachi Controller and high voltage and current for operation.

This future work is absolutely crucial to enable any possibility of a fully functional gripper or readings from a sensor or registering of a click of a button.

5.2. FTP implementation for Nachos

For the execution of the generated code by the Nachos application, it requires to be transferred to the Nachi Controller. The full instructions and procedures for the transfer of files are outlines in the "Code Conversion and Compile" document in the elicited resources package, but as a general guideline, the downloaded output file from the Nachos application needs to be transferred via FTP to the "**program**" folder on the Nachi Controller. The FileZilla application is recommended for the FTP transfers.

However, as FTP is a well-known technology and is widely used, the transfer of the textual representation of the generated code can theoretically be streamlined further with a better integration with Nachos. The proposed project is the implementation of the FTP protocol on the Nachos web application. The requirements of this project are outlined below:

1. Nachos shall be able to connect to the Nachi Controller via the FTP protocol with a click on the "Connect to Robot" button.
   (Given that the computer is within the Nachi Controller's proximity and connected to the router)

2. Nachos shall output the correct file containing the generated Nachi code (by NachosGenerator) to the correct file path on the Nachi Controller (the "Program" folder).

3. Nachos shall keep the FTP connection open until the "Disconnect" button is clicked.

4. (Optional) Nachos shall be able to display all the already available programs on Nachi Controller.

5. (Optional) Nachos shall be able to download or delete available programs on the Nachi Controller.

6. (Optional) Nachos shall only have access to the "Program" folder on Nachi Controller due to security concerns.

5.3. "AX on Desk" Software

Even though, it seems like the execution of all programs will always be reliant on the Nachi Controller and the FTP transfer, during the elicitation of the resources, there were reports for a remote access of Nachi robotics manipulator through the "AX on Desk" software. In fact, a manual specifically for this software was found with interesting capabilities that have great potential and opportunities. Even though, the software was never found, the opportunity to be able to control Nachi remotely is too considerable to pass on. Therefore, this research project is strongly recommended and could result in great findings.

5.4. Integration with third-party applications

Professionals nowadays use a variety of software to accelerate their workload. There are many examples of such software, but some specific examples are RoboDK, Grasshopper, Rhinoceros 3D and Slic3r.

During the work for the concurrent projects (4.2. and 4.2.1), some stakeholders expressed their interest for the integration of Nachi with other available robotics software, such as Grasshopper, RoboDK and others. Although, this was never fully researched in this project (due to its out-of-scope nature), it could result in promising findings and opportunities. Alternatively, some guidelines and instruction for these third-part application, would render very useful for these professionals.

The focus are professionals that will be utilising Nachi as part of a larger project. These individuals are already using these third-party applications and would like to import and test their already generated Nachi code via Nachos.

5.5. Simulation

One of the main features of NachiOS that was proposed and was unable to be satisfied due to its complexity, is the simulation capabilities within Nachos. "Nachos Simulation Engine" shall model a six degrees-of-freedom robotic manipulator, that resembles Nachi to some extent. The vision for this simulation will be taken mostly from RoboDK (**Figure 23**), but also some inspirations from "Simulator for Articulate Robots" paper (**Figure 24**) (by V. Oloworaran, 2020) [11] and "RoboSim" (**Figure 25**) which is a freely accessible software [12].

The Nachos Simulation Engine shall display Nachi's position, Nachi's position during the execution of a program, Nachi's joint angle values, and Nachi's pose values (translation and rotation) in at least the XYZ coordinate system. Furthermore, the following information and values shall be able to be manipulated through the simulation.

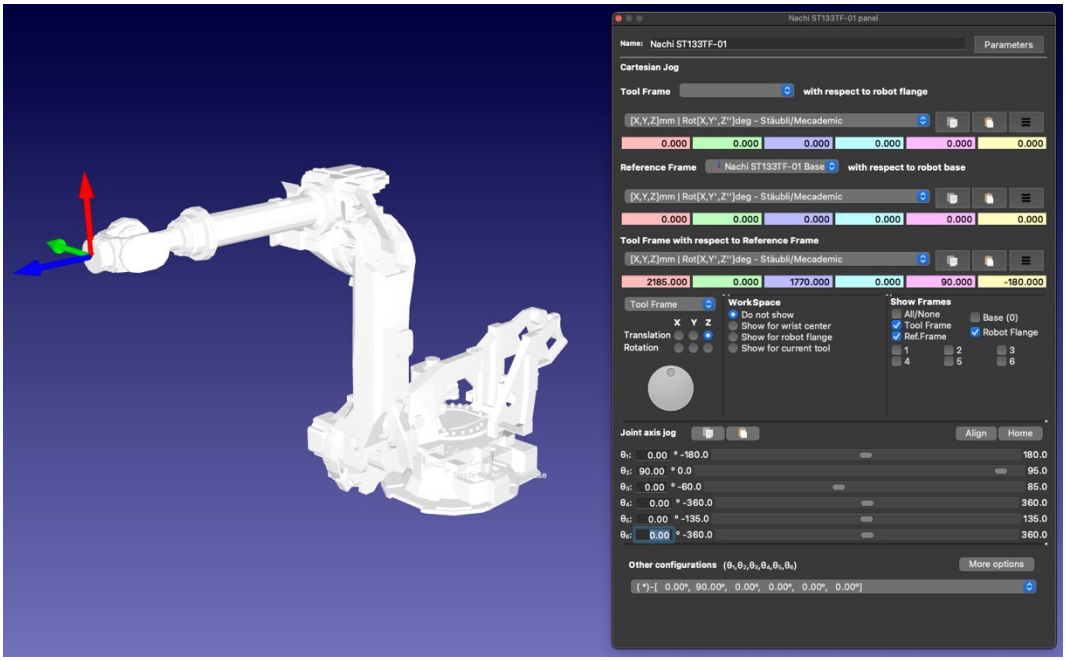

Figure 23 RoboDK Simulation Software. RoboDK is a widely used and popular software for the simulation of Robotics Manipulators. It displays most of the information that is required in Nachos.

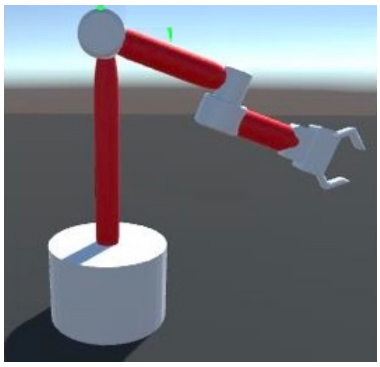

Figure 24 Simulator for Articulate Robots [11].

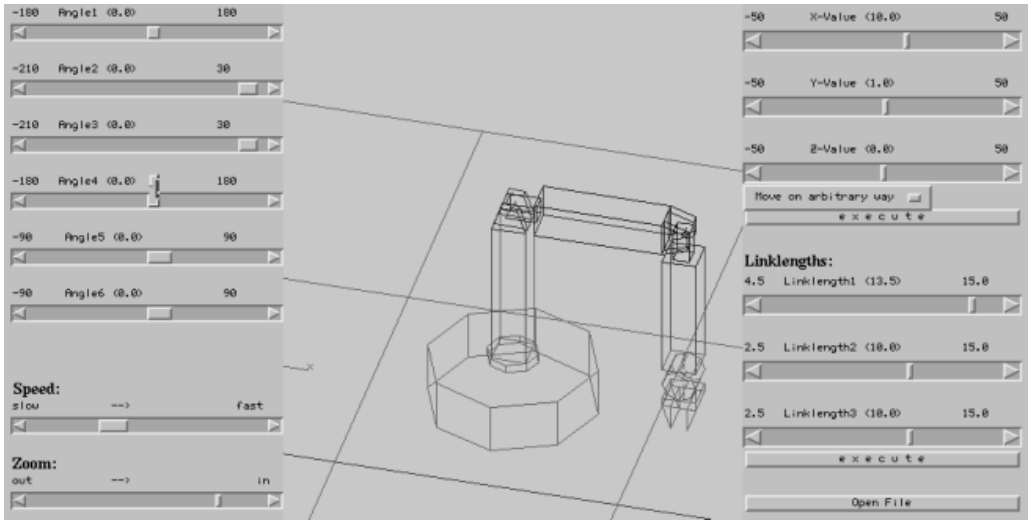

Figure 25 RoboSim Software [12]. Originally written by Rainer Pollak in C, then adapted by Johannes Schutzner in Java.

6. References

[1] *History of Nachi Robotic Systems*. Nachi Robotics Systems, Inc. *Retrieved October 5, 2021, from* https://www.nachirobotics.com/company-information/nachi-history/

[2] *Nachi Maintenance Service Manual – ST-F-01 series [AX20] (1108, MSTEN-208-008, 001)*, 8th ed. Nachi Robotics System Inc, Japan.

[3] *FD Controller Instructions Manual – Robot Language (1112, TFDEN-012-001, 001)*, 1st ed. Nachi Robotics System Inc, Japan.

[4] *Robot Software*. Nachi Robotics Systems, Inc. *Retrieved July 10, 2022, from* https://www.nachirobotics.com/fd-controller/software/

[5] *FD Controller Instruction Manual - Command Reference – Vol. 1 (FN0–365)*, 5th ed. Nachi Robotics System Inc, Japan.

[6] *FD Controller Instruction Manual - Command Reference – Vol. 2 (FN360-597)*, 5th ed. Nachi Robotics System Inc, Japan.

[7] *FD Controller Instruction Manual - Command Reference – Vol. 3 (FN598-end)*, 5th ed. Nachi Robotics System Inc, Japan.

[8] *Open-source 3D printing toolbox*, *Slic3r*. *Retrieved July 10, 2022,* from https://slic3r.org.

[9] *Simulator for industrial robots and offline programming,* RoboDK. *Retrieved July 10, 2022,* from https://robodk.com/.

[10] J. Showell, "Timber Gripping and Handling with Unigripper - Romheld Automation Pty Ltd", *Romheld Automation Pty Ltd*, 2022. [Online]. Available: https://romheld.com.au/timber-gripping-with-unigripper. [Accessed: 23-Jul-2022].

[11] Oloworaran, V. (2020). *Simulator for Articulate Robots* [Master's thesis]. https://robotics.ee.uwa.edu.au/theses/2020-RobotManipulator-Oloworaran.pdf

[12] RoboSim [Computer Software (Java)]. Retrieved from https://robotics.ee.uwa.edu.au/robosim/

[13] Scratch for Developers. Scratch. Retrieved October 5, 2021, from *https://scratch.mit.edu/developers*

[14] Pasternak, E., Fenichel, R., & Marshall, A. N. (2017). Tips for creating a block language with blockly. *2017 IEEE Blocks and Beyond Workshop (B&B)*. https://developers.google.com/blockly/publications/papers/TipsForCreatingABlockLanguage.pdf

[15] *Google/blockly: The web-based visual programming editor*. GitHub. Retrieved October 5, 2021, from https://github.com/google/blockly

[16] *Blockly Developer Tool*, *Google's Blockly*. *Retrieved July 10, 2022,* from https://blockly-demo.appspot.com/static/demos/blockfactory/index.html.

[17] *Define Blocks*, *Google's Blockly Documentation*. *Retrieved July 10, 2022,* from https://developers.google.com/blockly/guides/create-custom-blocks/define-blocks.