EVALUATION OF LABVIEW BASED CONTROL FOR A RECONFIGURABLE MANUFACTURING SUBSYSTEM

by

Darlington M Masendeke

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechanical) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. Anton Basson

March 2015

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification

Date: 2014/12/09

ABSTRACT

Evaluation of LabVIEW based control for a reconfigurable manufacturing subsystem

DM Masendeke

Department of Mechanical and Mechatronic Engineering
Stellenbosch University
Private Bag X1, 7602 Matieland, South Africa
Thesis: MEng (Mechanical)
March 2015

This thesis considers the evaluation of LabVIEW based control for reconfigurable manufacturing systems (RMSs), and in particular for an RMS subsystem. The evaluation used a rivet feeder station as a case study. The architecture of the rivet feeder station included a vibratory bowl feeder, a singulation device, a pick-n-place mechanism and an XYZ positioning table.

The objective of the research was to determine whether LabVIEW is a suitable development environment for implementing holonic control. The motivation for considering LabVIEW in this thesis was that other control approaches, such as IEC 61499 function blocks, agent-based control and object-orientated control, that have been used in most RMS research, have not found favour with industry.

The PROSA holonic reference architecture was adopted here, with the following holons: Coms Holon, Request Manager Holon, Order Holon, Product Holon Manager, Pick-n-Place Holon and XYZ Table Holon. The holons, constituting the controller, were designed based on LabVIEW's producer/consumer and state machine architectures. The controller was aimed at making the rivet feeder station reconfigurable. Furthermore, the controller was interfaced with a cell controller through a TCP/IP connection and an XML format of messaging was adopted for communication.

The main limitations and disadvantages of LabVIEW, for the implementation of holonic control systems, were found to be: the graphical programming which makes the block diagram cumbersome to follow, for complex applications; and that dynamic instantiation of objects or memory cannot be achieved in LabVIEW. However, LabVIEW holds the following advantages: shared variables and TCP/IP components simplify communication over a network; easy construction of the graphical user interface using controls and indicators on LabVIEW front panels; the XML standard format in LabVIEW provides flexibility to create your own unlimited tags; and immediate compilation of LabVIEW programs. Furthermore, LabVIEW easily can be integrated with hardware such as the compactRIO with a CANopen card.

The reconfigurability assessment of the rivet feeder station included three experiments which involved changing the product type, adding new devices and performing station diagnostics. From these experiments, it was concluded that the key characteristics of reconfigurable manufacturing systems were achieved, thereby demonstrating the suitability of LabVIEW for implementing holonic control.

OPSOMMING

Evaluering van LabVIEW-gebaseerde beheer vir 'n herkonfigureerbare vervaardigingsubstelsel

DM Masendeke

Departement Meganiese en Megatroniese Ingenieurswese
Universiteit Stellenbosch
Privaatsak X1, 7602 Matieland, South Africa
Tesis: MIng (Meganies)
Maart 2015

Hierdie tesis beskou die evaluering van LabVIEW-gebaseerde beheer vir herkonfigureerbare vervaardigingstelsels (RMSs), en in besonder vir 'n RMS substelsel. Die evaluering het 'n klinknaelvoerstasie as gevallestudie gebruik. Die voerstasie se argitektuur het 'n vibrerende bakvoerder, 'n singuleringstoestel, 'n optel-en-plaas-robot en 'n XYZ-posisioneringstafel ingesluit.

Die doelwit van die navorsing was om te bepaal of LabVIEW 'n geskikte ontwikkelingsomgewing vir die implementering van holoniese beheer bied. Die motivering vir die oorweging van LabVIEW in hierdie tesis was dat ander beheerbenaderings, soos IEC 61499 funksieblokke, agent-gebaseerde beheer and objek-georiënteerde beheer, wat in die meeste RMS-navorsing gebruik is, nie aanvaarding in die nywerheid gevind het nie.

Die PROSA holoniese verwysingsargitektuur is hier gebruik, met die volgende holone: Coms Holon (kommunikasieholon), Request Manager Holon (versoekbestuurholon), Order Holon (bestellingholon), Product Holon Manager (produkholonbestuurder), Pick-n-Place Holon (optel-en-plaasholon) en XYZ Table Holon (XYZ-tafelholon). Die holone, wat die beheerder uitmaak, se ontwerp is gebou op LabVIEW se vervaardiger/verbruiker- (*producer/consumer*) en toetstandmasjien-argitekture. Die beheerder was daarop gemik om die klinknaelvoerstasie herkonfigureerbaar te maak. Verder, die beheerder het 'n koppelvlak met 'n selbeheerder gehad, waarin 'n TCP/IP-verbinding en 'n XML-formaat vir die kommunikasie-boodskappe gebruik is.

Die belangrikste beperkings en nadele van LabVIEW, vir die implementering van holoniese beheerstelsels, het geblyk te wees: die grafiese programmering wat die blokdiagramme moeilik maak om te volg, vir komplekse toepassings; en dat objekte en geheue nie in LabVIEW dinamies geskep kan word nie. LabVIEW het, egter, die volgende voordele: gedeelde veranderlikes en TCP/IP-komponente vereenvoudig netwerkkommunikasie; gerieflike skepping van die grafiese gebruikerskoppelvlak met behulp van beheerders en meters (*controls and* *indicators*) op LabVIEW se voorpanele (*front panels*); die XML-standaard-formaat in LabVIEW voorsien buigsaamheid om jou eie onbeperkte etikette (*tags*) te skep; en onmiddellike vertaling van LabVIEW programme. Verder, LabVIEW kan maklik geïntegreer word met hardeware soos die "compactRIO" met 'n "CANopen" kaart.

Die beoordeling van die herkonfigureerbaarheid van die klinknaelvoerstasie het drie eksperimente ingeluit waarin die verandering van die produktipe, die byvoeging van nuwe toestelle en die uitvoering van stasie-diagnoses betrokke was. Uit hierdie eksperimente kan afgelei word dat die sleutel-eienskappe van herkonfigureerbare vervaardigingstelsels behaal is, waardeur aangetoon is dat LabVIEW geskik is om holoniese beheer te implementeer.

ACKNOWLEDGEMENTS

When I reflect on my research work at Stellenbosch University, I am compelled to acknowledge the contributions and help offered to me, without which this thesis would not have been a success.

I feel inadequate of words to express my gratitude to my supervisor, Prof AH Basson, for his guidance, time, patience and support towards my thesis. I thank him for the research experience that I have gained.

To the Copperbelt University council, I am very grateful for awarding me a scholarship to pursue my master's degree studies. This was indeed a rare life-time opportunity for me. Through this scholarship, my academic aspirations have been satisfied.

I am grateful to my beloved family and friends for their support and encouragements. This gave me the morale to work harder in my thesis.

The manufacture of the design components for my thesis could not have been successful, had it not been for the Mechanical workshop staff. I would like to express my gratitude to all of them for their time, efforts and willingness to help.

I wish to thank all the Mechatronics, Automation and Design Research Group members for their advice and help during my research work. I would like to wish them God's blessings for their time with me.

I am grateful to Mr Reynaldo Rodriguez for his expert advice and also for helping with the setting up of the compactRIO system and the NI CANopen interface. I really wish to thank him for his help even when he had busy schedules.

Above all, I am very grateful to God and therefore wish to give Him the glory He deserves for giving me the strength, hope and good health during my research work at Stellenbosch University.

DEDICATION

To

My late Grandmother, Sibukani Dube Masendeke,

You answered the Lord’s call when this thesis was still in progress.

For the cheerful life we lived together, I will miss you.

I will always remember you for your dearest love and care.

Your words of wisdom will forever echo in my mind.

May the almighty God rest your soul in eternal peace.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

API	Application Programming Interface
CAN	Controller Area Network
CC	Cell Controller
CNC	Computer Numerically Controlled
DI	Digital Input
DMS	Dedicated Manufacturing System
DO	Digital Output
FIFO	First In First Out
FMS	Flexible Manufacturing System
IP	Internet Protocol
NC	Normally Closed
OH	Order Holon
PDO	Process Data Object
PH	Product Holon
PHM	Product Holon Manager
RMH	Request Manager Holon
RMS	Reconfigurable Manufacturing System
SDO	Service Data Object
SH	Supervisor Holon
SUH	Singulation Unit Holon
TCP	Transmission Control Protocol
TH	Task Holon
VI	Virtual Instrument
XML	Extensible Markup Language

1. INTRODUCTION

1.1 Background

As noted by Koren (2005), Reconfigurable Manufacturing Systems (RMS), which was invented at the University of Michigan in 1999, is now an established field of research. Koren (2005) defines an RMS as a responsive manufacturing system whose production capacity is adjustable to fluctuations in product demand and whose functionality is adaptable to new products. Furthermore, Malhotra et al (2010) state that the goal in implementing an RMS is to be able to cope with random changes in production requirements through reconfiguration. Reconfiguration, as defined by Malhotra et al (2010), is an iterative process that entails selection of a manufacturing configuration that is optimally fit for purpose. RMSs have modularity, integrability, flexibility, scalability, convertibility, and diagnosability as the key characteristics that help achieve the desired reduction in time and cost (Koren, 2005).

Manufacturing companies in the 21st century face increasingly frequent and unpredictable market changes driven by global competition, including the rapid introduction of new products and constantly varying product demand (Koren et al, 2010). Furthermore, Malhotra et al (2010) stated that the present world is changing thereby giving a turbulent business environment and that the performance of the manufacturing system is largely dependent on the ability to be flexible as well as being able to reconfigure the system for new demands. Therefore, Malhotra et al (2010) identified the following benefits of adopting RMSs:

- Increased product quality
- Reduced time required for product changeover
- Enhanced ease of prototype development
- Reduction of lead-time for launching new manufacturing system
- Rapid upgrading and quick integration of new process technology

From the aforementioned it is the view of the author that the challenges facing the manufacturing companies in South Africa are due to lack of reconfigurability in the design of manufacturing systems. Deloitte (2013) noted that high cost of labour, small size of domestic market, the threat of cheap imports, policy uncertainty, high input costs and limited skills base are some of the challenges facing the manufacturing companies in South Africa. It is therefore recommendable that manufacturing companies in South Africa should adopt the reconfigurable manufacturing systems approach in order to remain competitive and to survive the turbulent business environments.

In spite of the potential advantages of RMSs, industry has been reluctant to adopt these technologies. Lack of adoption of RMSs in industry can be attributed to: expensive controllers, difficult integration of machines and difficult selection of machine modules (Malhotra et al, 2010). Further research is therefore required to develop RMSs that are more acceptable to industry.

This thesis aims to contribute to the aforementioned research by evaluating LabVIEW based control for a reconfigurable manufacturing subsystem. The evaluation is done on a rivet feeder station which is used as a case study for this thesis. The rivet feeder station is part of the RMS cell being developed at the Department of Mechanical and Mechatronic Engineering at Stellenbosch University. Other stations that form part of the RMS cell include the electrical test station, stacking station and the transport sub-system. The RMS cell is aimed at demonstrating the automation requirements for processes at CBI Electric.

CBI Electric is a supplier of quality low voltage electrical distribution, protection and control equipment. Additionally, manufacturing is the core business of CBI Electric. Thus, materials are converted into finished products such as the circuit breaker components, which will be considered in this research.

Various students from the Mechatronics, Automation and Design Research Group have researched RMSs. Therefore, the rivet feeder station will form part of the system that has been developed with other students at the University of Stellenbosch. To elaborate on this, Kruger (2013) developed a control system for the feeder subsystem, Le Roux (2013) developed a control system for the transportation system, Mulubika (2013) evaluated control strategies for a modular Cartesian weld robot and the cell controller for the whole welding assembly cell and Hoffman (2014) worked on creating an RMS using acceptable industrial technologies. Presently, Graefe Rainer is using the control of a stacking and buffering station to evaluate standard OOP languages (C#, in his case) as a platform to implement holonic control; Marius Claassen is investigating the design of an RMS for the manufacturing of thermoplastic fibre-reinforced composite parts; Clint Steed is developing a vision-based reconfigurable feeding cell and he is investigating two types of configurations, namely fixed camera configurations and eye-in hand configurations; and Marcus Kotze is developing a highly distributed holonic controller for a reconfigurable conveyor system. Finally, as stated earlier, the author evaluated LabVIEW based control for a reconfigurable manufacturing subsystem.

1.2 Objective

The objective of this research is to determine whether LabVIEW is a suitable development environment for implementing holonic control.

The objective has been approached firstly by designing a rivet feeder station, which is used as a case study in this thesis, and secondly by implementing holonic control in LabVIEW. Furthermore, reconfigurability assessments are used to evaluate the suitability of a LabVIEW controller to achieve reconfigurability in a rivet feeder station.

1.3 Motivation

Control strategies such as IEC 61499 function blocks, agent-based control and object-oriented control (C++, Java and C#) have been used for reconfigurable manufacturing systems (Brennan et al, 2002, Hussain et al, 2007). Although agent-based control is widely used in RMS research, it has not found favour with industry. The use of object-oriented approaches also is not common in industry. This thesis is therefore aimed at evaluating LabVIEW as a platform for the development of control systems for RMSs.

LabVIEW can be considered the industry standard for test, measurement, automation and control software. It can be used to communicate with hardware such as data acquisition, vision and motion control devices. Elliott et al (2007) pointed out that unlike most programming languages, LabVIEW compiles code as it is created thereby providing immediate syntactic and semantic feedback and reducing the time required for development and testing. What is more, NI LabVIEW supports multithreaded application design and executes code in an inherently parallel, rather than sequential manner. Furthermore, LabVIEW's virtual instruments (VIs) are hierarchical and modular in nature. Therefore, the multithreading capability and the modular nature of LabVIEW make it attractive for implementing holonic control.

Elliott et al (2007) also noted that, similar to most programming languages, LabVIEW supports all common data types such as integers, floats, strings and clusters (structures) and can readily interface with external libraries, ActiveX and the .NET framework. In addition, Shyam (2012) presented a paper on the design of a Robotic Arm for Picking and placing an object controlled using LabVIEW. This demonstrated the automation capability of LabVIEW.

2 LITERATURE REVIEW

This section begins with a discussion of three types of manufacturing systems namely dedicated manufacturing systems (DMSs), flexible manufacturing systems (FMSs) and RMSs. RMSs are discussed in detail with reference to their key characteristics and design issues, since they are the manufacturing systems of focus in this research. Furthermore, the discussion then considers the control architectures used in manufacturing systems. Finally, LabVIEW based control is discussed.

2.1 Types of manufacturing systems

According to Koren (2010), a cost effective response to market changes requires a new manufacturing approach that must not only combine the high throughput of DMSs with the flexibility of FMSs, but also be capable of responding to market changes by adapting the manufacturing system and its elements quickly and efficiently. These capabilities are encompassed in RMSs, whose capacity and functionality can be changed when needed. Koren (2010) further states that capacity, functionality and cost are what differentiate the three manufacturing systems. Thus, DMSs, FMSs and RMSs will be discussed to show that the latter is the most suitable to be considered for this research.

2.1.1 Dedicated manufacturing systems

DMSs produce a company’s core products or parts at high volume using dedicated machines (Koren et al, 2010). Dedicated machines used in DMSs are designed around a specific part that is mass produced. Furthermore, a dedicated machine performs a single operation with high reliability, repeatability and productivity, and is therefore less expensive (Katz, 2007).

According to Setch and Lagos (2004), cost in DMSs is low as long as they operate at full capacity, which entails that demand for the products should exceed their supply. This, however, is not usually the case as DMSs rarely operate at full capacity due to increasing pressure from global competition (Koren et al, 2010). Furthermore, DMSs are not scalable because they have fixed cycle times and capacity (Koren et al, 1999). Therefore, DMSs are not suitable to meet the requirements of today’s competitive market which is characterised by fluctuations in the demand for products.

2.1.2 Flexible manufacturing systems

EIMaraghy (2006) defines FMSs as integrated systems of machine modules and material handling equipment under computer control for the automatic random processing of palletized parts. He further states that the objective of FMSs is to cost-effectively manufacture several types of parts, within pre-defined part families that can change over time, with minimum changeover cost, on the same system at the required volume and quality.

FMSs typically consist of general-purpose computer-numerically-controlled (CNC) machines and other programmable forms of automation. The throughput of FMSs is much lower than that of DMSs because CNC machines are characterised by single-tool operation. The combination of high equipment cost and low throughput makes the cost per part using FMSs relatively high. (Koren et al, 2010).

2.1.3 Reconfigurable manufacturing systems

The changing manufacturing environment characterized by aggressive competition on a global scale and rapid changes in process technology requires creating production systems that are themselves easily upgradable and into which new technologies and new functions can be readily integrated (Koren et al, 2000).

An RMS is one designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in market or regulatory requirements (Koren et al, 2010). It can also be defined as a manufacturing systems paradigm that aims at achieving cost-effective and rapid system changes, as needed and when needed, by incorporating principles of modularity, integrability, flexibility, scalability, convertibility and diagnosability (ElMaraghy, 2006). RMSs are designed to rapidly produce different product families in the shortest time and at the lowest cost without sacrificing quality. The major characteristic of such systems is called reconfigurability, which is the ability of rearranging and/or changing manufacturing elements aimed at adjusting to new environmental and technological changes. Manufacturing reconfigurability has become a new economic objective along with classical objectives such as low cost and high quality (Malhotra et al, 2009).

The RMSs are effective paradigms that meet the manufacturing requirements (Wang et al, 2012). Furthermore, the concept of RMSs has been proposed to meet the changes and uncertainties of manufacturing environment, and this objective would be achieved by reconfiguring hardware and/or software resources.

System reconfigurability can be classified in terms of the levels where the reconfigurable actions are taken. As shown in Figure 1, reconfigurability at lower levels is mainly achieved by changing hardware resources, and reconfigurability at higher levels is mainly achieved by changing software resources.

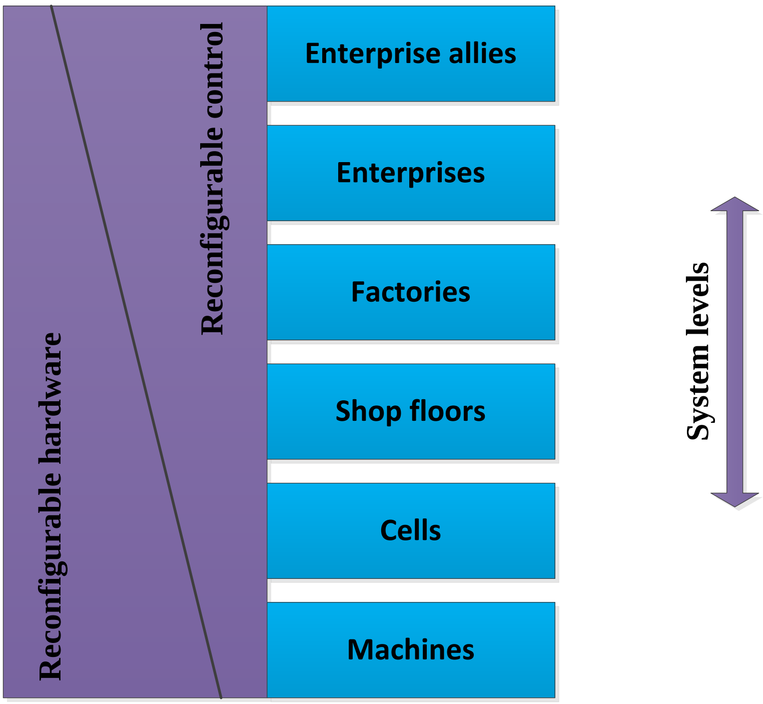


Figure 1: System organisation and reconfigurable resources (adapted from Wang et al, 2012).

There is some disagreement amongst researchers about the definition of RMSs. Some researchers understand RMSs as an intermediate paradigm between DMS and FMS, while some argue that an RMS is an advanced paradigm whose flexibility must be higher than that of FMSs, and others think it is not very meaningful to distinguish RMSs from FMSs. However, Koren et al (2010) describe RMSs as a manufacturing approach that offers cost effective response to market changes. Such an approach does not only combine the high throughput of DMS with the flexibility of FMS but also responds to market changes by adapting the manufacturing system and its elements quickly and efficiently. These capabilities are encompassed in reconfigurable manufacturing system (RMS), whose capacity and functionality can be changed exactly when needed.

Reconfigurable manufacturing systems are designed to rapidly produce different product families. The objective of an RMS is to provide the functionality and capacity that is needed, when it is needed. Thus, a given RMS configuration can be dedicated or flexible, or in between, and can change as needed. An RMS goes beyond the economic objectives of FMS by permitting (Koren et al, 2000):

- Reduction of lead time for launching new systems and reconfiguring existing systems.
- The rapid manufacturing modification and quick integration of new technology and/or new functions into existing systems.

The reconfigurable manufacturing system have some merits, such as increased product quality, increased product variety, increased uptime, reduced ramp-up time for new products, enhanced ease of prototype development, reduced maintenance and reduced in floor space (Malhotra et al, 2009).

Koren (2005) state that during the lifetime of the RMS, its functionality is reconfigured several times to fit new products designed to be produced on the RMS, and its production capacity changes according to market demand. A ramp-up period to re-calibrate the machines must follow each reconfiguration. Achieving a short ramp-up period is very critical with RMS since there are many ramp-up periods during the lifetime of the system, as shown in Figure 2.

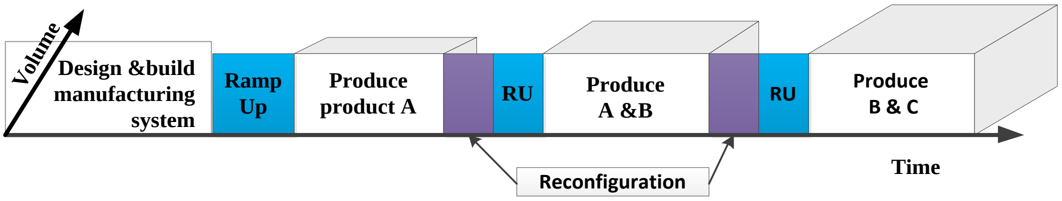


Figure 2: RMS functionality reconfiguration (adapted from Koren, 2005).

During its lifetime the RMS will be reconfigured many times to adapt to the market in terms of production volume (capacity) and type of goods produced (changed functionality). Furthermore, for a manufacturing system to be readily reconfigurable, the system must possess the following six core reconfigurable characteristics (Koren et al, 2010):

i. **Customization:** System or machine flexibility limited to a single product family, thereby obtaining customised flexibility.

ii. **Convertibility:** The ability to easily transform the functionality of existing systems and machines to suit new production requirements.

iii. **Scalability:** The ability to easily modify production capacity by adding or removing manufacturing resources (e.g. machines) and/or changing components of the system.

iv. **Modularity:** The compartmentalization of operational functions into units that can be manipulated between alternate productions schemes for optimal arrangement.

v. **Integrability:** The ability to integrate modules rapidly and precisely by a set of mechanical, informational, and control interfaces that facilitate integration and communication.

vi. **Diagnosability:** The ability to automatically read the current state of a system to detect and diagnose the root causes of output product defects, and quickly correct operational defects.

RMSs constitute a new class of systems characterized by adjustable structure and design focus, as shown in Table 1.

Table 1: Comparison of manufacturing systems (Koren et al, 2010)

System property	DMSs	RMSs	FMSs
System structure	Fixed	Changeable	Changeable
Machine structure	Fixed	Changeable	Fixed
System focus	Part	Part family	Machine
Scalability	No	Yes	Yes
Flexibility	No	Customized	General
Simultaneous operating tools	Yes	Possible	No
Productivity	Very high	High	Low
Cost per part	Low	Medium	Reasonable

For any type of RMSs, critical issues will be involved. These issues include architecture design, configuration design and control design (Wang et al, 2012).

Architecture design determines system components and their interactions. System components are encapsulated modules. Interactions are the options when the modules are assembled. Wang et al (2012) further state that RMS architecture has to be designed to produce as many system variants as possible, so that the system can deal with changes and uncertainties cost-effectively. Architecture design is involved at the phase of system design.

Configuration design determines system configuration under given system architecture for a specific task. A configuration is an assembly of the selected modules. Configuration design is involved at the phase of system application.

Control design determines appropriate process variables so that a configuration can be operated to fulfil the task satisfactorily. Control design is involved at the phase of system operation.

2.2 Control of manufacturing systems

2.2.1 Main types of control architectures

Meng et al (2006) discuss three kinds of control architectures namely, centralised, hierarchical and distributed, as shown in Figure 3. Centralized control architecture is one in which all the subsystems (machine components) are controlled by a central controller which carries out all the automation processes. In hierarchical control multiple controllers are arranged in a hierarchy form. According to Van Brussel et al (1998), commands flow top-down and feedback information flows bottom-up in hierarchical control. All hierarchical control architectures require a fixed structuring while the system is running and assume deterministic behaviour of the components (Van Brussel et al, 1998). Distributed control involves the use of independent and interconnected controllers.

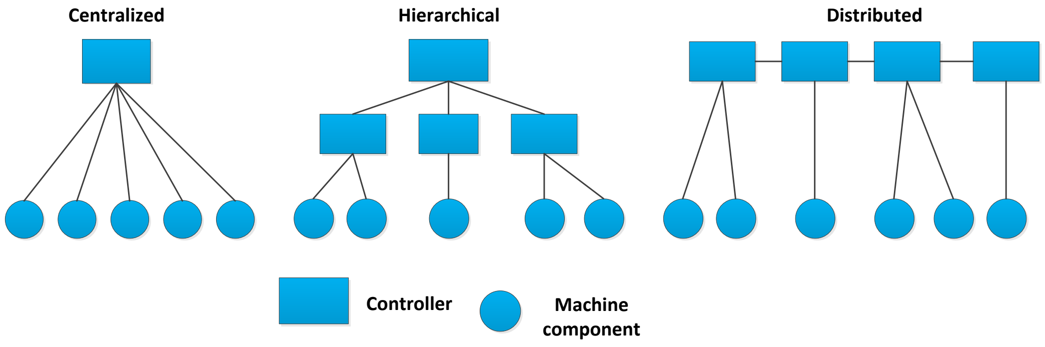


Figure 3: Three types of control architectures (Meng et al, 2006)

Both centralized and hierarchical control architectures are not reconfigurable-friendly owing to the shortcomings which they have such as structural rigidity, difficulty of control system design, lack of flexibility. Furthermore, to modify structure, the system is required to be shut down and all data structures of higher levels need to be updated (Meng et al, 2006).

Distributed control is implemented in systems that have heterarchical architecture, such as holonic manufacturing systems. Heterarchical control architectures have a flat structure and are composed of independent entities sometimes called agents. Unlike hierarchical control which has a master-slave relationship, heterarchical control uses a negotiation protocol for the exchange of information and commands (Van Brussel et al, 1998).

Duffie et al (1996) explain that since the 1990s there has been a growing interest in heterarchical control architecture because of the attractiveness of the architecture as an alternative to conventional hierarchical control architecture.

According to Scholz and Freitag (2007) heterarchical control architecture has advantages over hierarchical control architecture as it leads to:

- Reduced complexity by localizing information and control,
- Reduced software development costs by eliminating supervisory levels,
- Higher maintainability and modifiability due to improved modularity and self-configurability,
- Reliability by taking a fault-tolerant approach rather than a fault-free approach

2.2.2 Holonic control

Van Brussel et al (1998) indicate that future manufacturing systems need to cope with frequent changes and disturbances. Holonic manufacturing is a highly distributed control paradigm that promises to handle these problems successfully. It is based on the concept of autonomous co-operating agents, called ‘holons’.

The word ‘holon’ was proposed by Arthur Koestler. It is a combination of the Greek holos which means whole, with the suffix –on which, as in proton or neutron, suggests a particle or part. Koestler proposed the word holon to describe the hybrid nature of sub-wholes/parts in real-life systems; holons simultaneously are self-contained wholes to their subordinated parts, and dependent parts when seen from the inverse direction.

The Holonic Manufacturing Systems Consortium developed the following list of definitions to help understand and guide the translation of holonic concepts into a manufacturing setting (Van Brussel et al, 1998):

Holon: An autonomous and co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be part of another holon.

Autonomy: The capability of an entity to create and control the execution of its own plans and/or strategies.

Co-operation: A process whereby a set of entities develops mutually acceptable plans and executes these plans.

Holarchy: A system of holons that can co-operate to achieve a goal or objective. The holarchy defines the basic rules for co-operation of the holons and thereby limits their autonomy.

2.2.2.1 PROSA reference architecture

PROSA is one of the best known reference architectures for holonic control. The acronym PROSA stands for Product-Resource-Order-Staff Architecture.

Therefore, this architecture consists of four holons, namely product, resource, order and staff.

Van Brussel et al (1998) note that, in the research community as well as in manufacturing companies, three relatively independent manufacturing concerns do exist:

- Resource aspects such as driving the machine at optimal speed and maximising its capacity.
- Product and process related technological aspects such as which operations need to be performed to achieve a good quality product.
- Logistical concerns about customer demands and due dates.

Thus, from this analysis, Van Brussel et al (1998) made a conclusion that there are three types of basic holons: product holons, resource holons and order holons. These are then the basic holons to PROSA.

A resource holon contains a physical part (production resource) of the manufacturing system and an information processing part that controls the resource. A resource holon offers production capacity and functionality to the surrounding holons. Van Brussel et al (1998) state that a resource holon is an abstraction of the production means such as a factory, a shop, machines, furnaces, conveyors, pipelines, pallets etc.

A product holon holds the process and product knowledge to assure the correct making of the product with sufficient quality. Contained in a product holon is consistent and up-to-date information on the product life cycle, user requirements, design, process plans, bill of materials, quality assurance procedures, etc. This being the case, Van Brussel et al (1998) point out that a product holon contains the ‘product model’ of the product type, not the ‘product state model’ of one physical product instance being produced. The product holon acts as an information server to the other holons in the Holonic Manufacturing System (Van Brussel et al, 1998).

An order holon represents a task in the manufacturing system. It is responsible for performing the assigned work correctly and on time. It manages the physical product being produced, the product state model and all logistical information processing related to the job. Often, the order holon can be regarded as the workpiece with a certain control behaviour to manage it to go through the factory, for instance, to negotiate with other parts and resources to get produced (Van Brussel et al, 1998).

Figure 4 illustrates that the basic holons exchange knowledge information about the manufacturing system.

- Product holons and resource holons communicate process knowledge. Process knowledge contains the information and methods on how to perform a certain process on a certain resource.
- Product holons and order holons exchange production knowledge, which is the knowledge that represents the information and methods on how to produce a certain product using certain resources.
- Resource holons and order holons share process execution knowledge, which is the knowledge containing the information and methods regarding the progress of executing processes on resources.

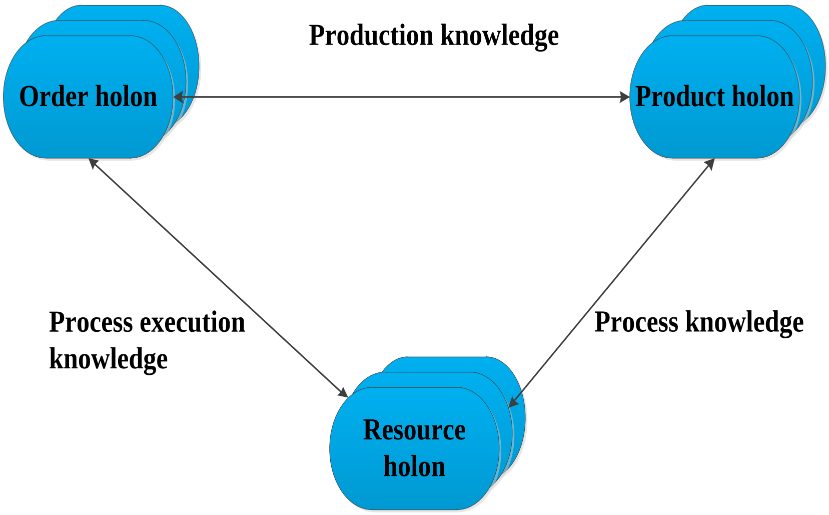


Figure 4: Basic building blocks of a HMS and their relations (Van Brussel et al, 1998)

The staff holons assist the basic holons in performing their work. In addition, the staff holons consider some facets of problems facing the basic holons and provide sufficient information so that correct decisions are made to solve the problems. The introduction of staff holons reduces the work load and work complexity of the basic holons, by proving them with expert knowledge.

Comparing PROSA with existing manufacturing control approaches, Van Brussel et al (1998) conclude that PROSA covers all aspects of both hierarchical and heterarchical control architectures. Van Brussel et al (1998) also point out that PROSA introduces two significant innovations. The system structure is decoupled from the control algorithm, logistical aspects can be decoupled from technical ones and PROSA allows incorporating more advanced hybrid-control algorithms.

2.2.2.2 ADACOR reference architecture

ADACOR is also one of the best known reference architectures for holonic control. The acronym ADACOR stands for ADAptive holonic Control aRchitecture. ADACOR intends to contribute to the improvement of the manufacturing control systems performance in terms of the agile reaction to emergence and change, by increasing the agility and flexibility of the enterprise when it works in volatile environments, characterised by the frequent occurrence of disturbances (Lietao and Restivo, 2006).

Lietao and Restivo (2006) point out that ADACOR architecture is built upon a set of autonomous and cooperative holons, to support the distribution of skills and knowledge and to improve the capability of adaptation to environment changes. Each holon is a representation of a manufacturing component that can be either a physical resource or a logic entity (Lietao and Restivo, 2006).

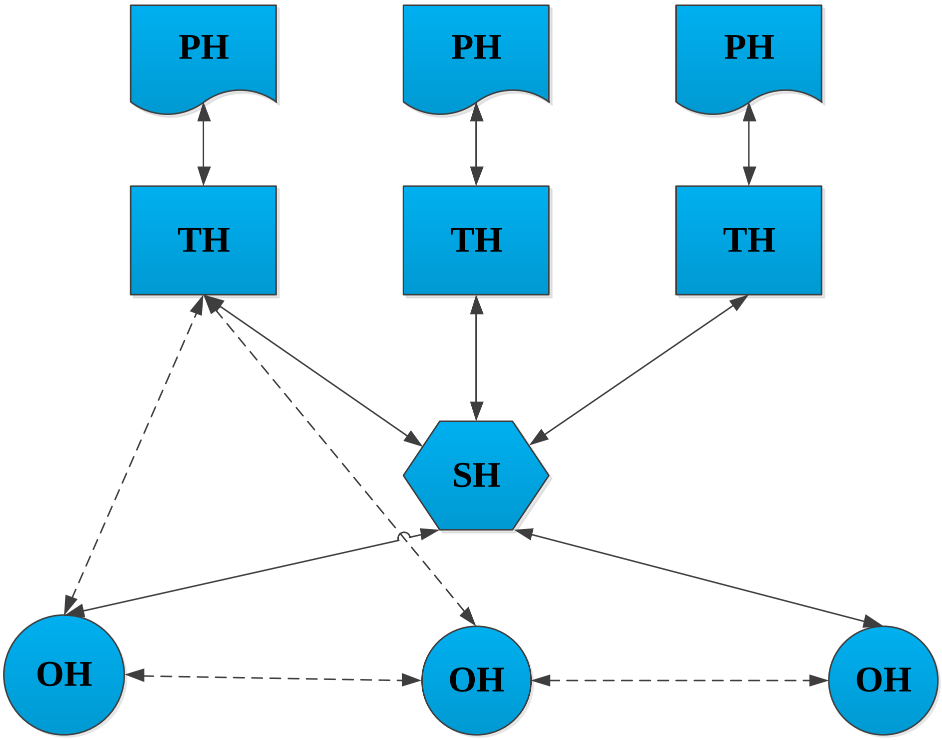


Figure 5: ADACOR Holon classes, with interactions (adapted from Lietao and Restivo, 2006)).

As can be seen from Figure 5, ADACOR architecture defines four manufacturing holon classes, namely product (PH), task (TH), operational (OH) and supervisor holon (SH), according to their functions and objectives (Lietao and Restivo, 2006). Lietao and Restivo (2006) point out that the product, task and operational holons are quite similar to the product, order and resource holons defined in PROSA. However, the supervisor holon presents characteristics not found in the PROSA staff holon.

In ADACOR, a product holon represents each product available to be produced in the factory plant. The product holon contains all information related to the product and being responsible for the short-term process planning. Each production order launched to the shop floor (to execute a product or sub-product) is represented by a task holon. Operational holons represent the physical resources available in the shop floor. The supervisor holons introduces coordination and global optimisation in decentralised control and is responsible for the formation and coordination of groups of holons.

2.3 LabVIEW based control

Halvorsen (2012) defines LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) as a platform and development environment for a visual programming language from National instruments. The graphical language is named “G”. Originally released for the Apple Macintosh in 1986, LabVIEW is commonly used for data acquisition, instrument control and industrial automation on a variety of platforms including Microsoft Windows, various flavours of UNIX, Linux, and Mac OS X. The version of LabVIEW used in this thesis was LabVIEW 2013.

The code files have the extension “.VI”, which is an abbreviation for “Virtual Instrument”. LabVIEW offers a lot of additional Add-Ons and Toolkits (Halvorsen, 2012).

2.3.1 Data flow programming

The programming language used in LabVIEW, also referred to as G, is a dataflow programming language. Execution is determined by the structure of a graphical block diagram on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available. Since this might be the case for multiple nodes simultaneously, G is inherently capable of parallel execution. Multi-processing and multi-threading hardware is automatically exploited by the built-in scheduler, which multiplexes multiple OS threads over the nodes ready for execution (Halvorsen, 2012).

2.3.2 Graphical programming

Halvorsen (2012) and National Instruments (2003) state that LabVIEW programs or subroutines are called virtual instruments (VIs), because their appearance and operation imitate physical instruments, such as oscilloscopes and multimeters. Each VI has three components: a block diagram, a front panel and a connector pane. The connector pane is used to represent the VI in the block diagrams of other, calling VIs. Controls and indicators on the front panel allow an operator to input data into or extract data from a running virtual instrument. It is worthwhile to note that the front panel can also serve as a programmatic interface. Therefore, a virtual instrument can either be run as a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the given node through the connector pane. This implies that each VI can be easily tested before being embedded as a subroutine into a larger program.

Examples of the front panel and the block diagram are shown in Figure 6 and Figure 7, respectively.

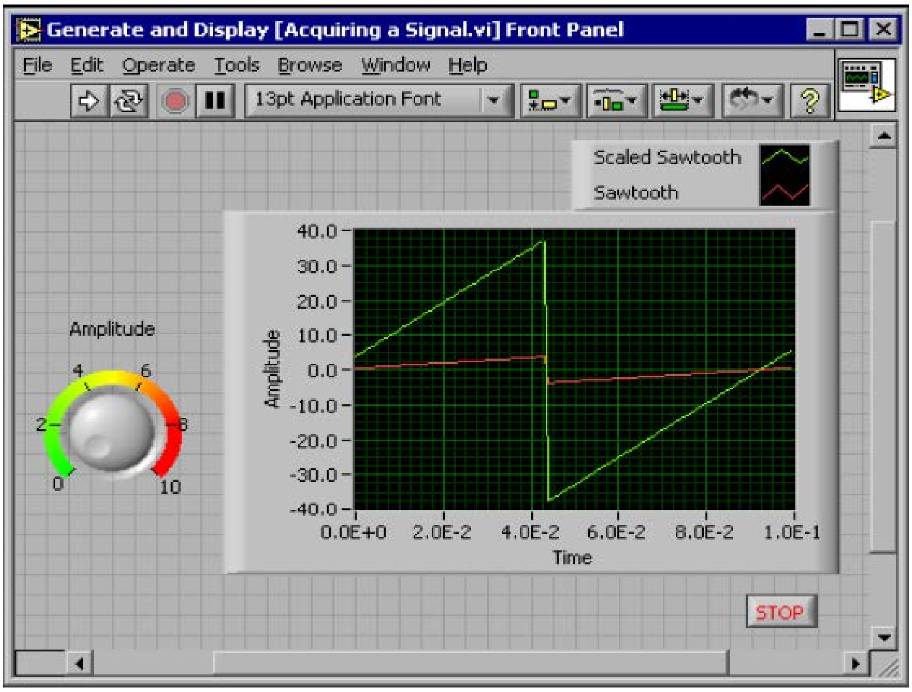


Figure 6: Example of a front panel (National Instruments, 2003)

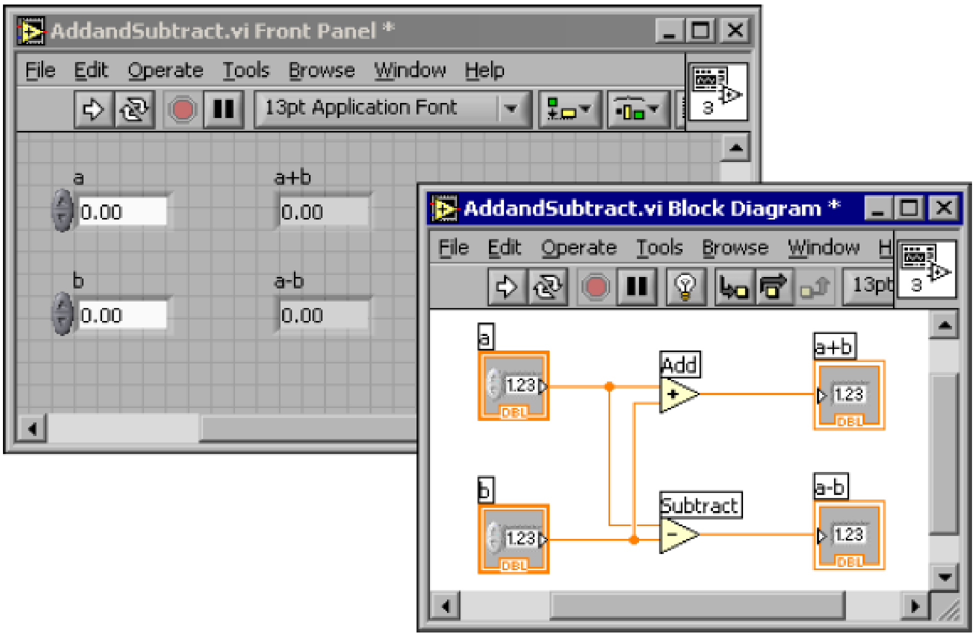

AddandSubtract.vi Front Panel *
File Edit Operate Tools Browse Window Help
13pt Application Font
3
a
a+b
0.00
0.00
b
a-b
0.00
0.00
AddandSubtract.vi Block Diagram *
File Edit Operate Tools Browse Window H
13pt
3
a
1.23
DBL
Add
a+b
1.23
DBL
b
1.23
DBL
Subtract
a-b
1.23
DBL

Figure 7: Example of a block diagram and corresponding front panel (National Instruments, 2003).

According to National Instruments (2003), the front panel is built with controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials and other input devices. On the other hand, indicators are graphs, LEDs and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data which the block diagram acquires or generates. After the front panel has been built, graphical representation is used to add code to control the front panel objects. The block diagram contains this graphical source code.

2.3.3 Key features of LabVIEW

Elliott C et al (2007) note that LabVIEW has several key features that make it a good choice in an automation environment. The features include simple network communication, turnkey implementation of common communication protocols such as RS232, GPIB and TCP/IP, powerful tool sets for process control and data fitting, fast and easy user interface construction and an efficient code execution environment.

Elliott C et al (2007) state that LabVIEW supports a distributed architecture by virtue of enabling seamless network communication through technologies such as VI Server and data sockets transfer protocol (DSTP). Data sockets allow easy transfer of data between remote computers with basic read and write functions. Furthermore, Elliott C et al (2007) note that NI and many third-party vendors provide a variety of LabVIEW toolsets for advanced data acquisition, data analysis, system control (PID and fuzzy logic), database connectivity, etc. Elliott C et al (2007) also note that each of these toolsets/libraries provides advanced functions that can be easily incorporated into applications.

LabVIEW also has debugging techniques and error handling mechanisms. National Instruments (2005) gives a number of debugging techniques that one can use to identify and correct problems with the VI or the block diagram data flow. The techniques include: execution highlighting, single-stepping, probe tools and breakpoints.

The execution highlighting is a technique which uses a highlight execution button to monitor the movement of data on the block diagram from one node to the other using bubbles that move along the wires. In the single-stepping technique the step over or step into button on the block diagram toolbar is used. Furthermore, the probe tool is used to view the data that passes through a wire. Finally, the breakpoint tool is used to set a breakpoint on a wire on the block diagram and then execution is paused at that location (National Instrument, 2005).

National Instruments (2005) and NI LabVIEW (2013) state that without a mechanism to check for errors, one only knows that the VI does not work properly, but cannot know why and where the error occurred. Therefore, National Instruments (2005) introduced the concept of error clusters that can be used to handle errors. The error clusters have the following components of information:

- *Status* is a Boolean value that reports TRUE if an error occurred.
- *Code* is a 32-bit signed integer that identifies the error numerically.
- *Source* is a string that identifies where the error occurred.

Furthermore, National Instruments (2005) and NI LabVIEW (2013) note that Case Structures can be used with a While Loop or For Loop for error handling. An error cluster can be wired to the conditional terminal of a While Loop or For Loop to stop the iteration of the Loop if an error occurs. Furthermore, National Instruments (2005) and NI LabVIEW (2013) state that when an error cluster is wired to the selector terminal of a Case structure, the Case selector label displays two cases namely Error and No Error and the border of the Case structure changes colour-red for Error and green for No Error. If an error occurs, the Case structure executes the Error subdiagram.

2.3.4 Data communication in LabVIEW

Data communication in LabVIEW allows a user to transfer data between VIs on the same computer or between VIs on different computers. Humayun et al (2013) give details about the methods of transferring data and their pros and cons. These methos are: global variables, notifiers, queues and shared variables.

Global variables are one of the options for inter-VI data transmission. Global variables can be accessed from any VI residing in memory space. Furthermore, when this variable is created, LabVIEW generates a VI containing no block diagram but only a front panel. The user can put indicators and controls to this VI in order to set whether it would be read from or written to, from another VI. The global variables are the easiest method to transfer data between VIs but they cannot be used for variables which carry continuously changing data since they only collect and store only one value at a time (Humayun et al., 2013).

Notifiers can only be used to transfer data between VIs executing on the same computer. The transferred data can be referred to as a notification. Notifiers can be used as indicators for other data transferring techniques between VIs e.g. in a queue, it can notify the queue when to read for data (Humayun et al., 2013).

Queues are also used for data communication between two different VIs. There are two types of queues used in LabVIEW environment: Last-In First-Out (LIFO) and First-In First-Out (FIFO). Both forms of queues work by forming lines of data. The difference between the two forms lies in the way data is queued. In LIFO the last data queued is the first to be removed (dequeued) while in a FIFO the first data queued is the first to be dequeued. Queues are suitable for continuous data transfer as they have a buffer that ensures no loss of data.

As noted by Humayun et al (2013), shared variable is a recent feature added in later versions of LabVIEW. Shared variables can be accessed over a network. Furthermore, shared variables have no buffer created and thus less memory space is required. However, it must be noted that due to lack of a buffer, use of shared variables might cause loss of data for continuous transmissions

2.3.5 Design architectures in LabVIEW

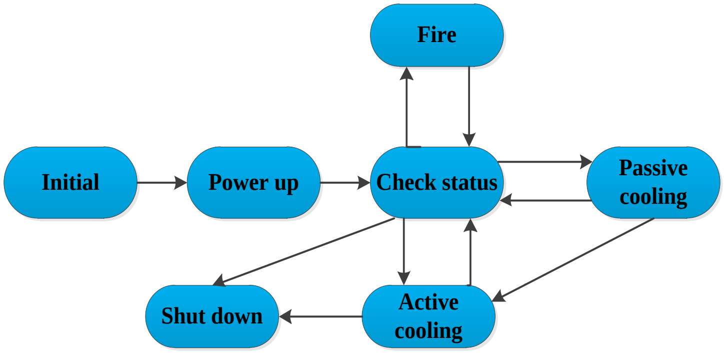
2.3.5.1 State machines architecture

According to NI LabVIEW (2013), a LabVIEW state machine contains four basic elements which are fundamental to its execution:

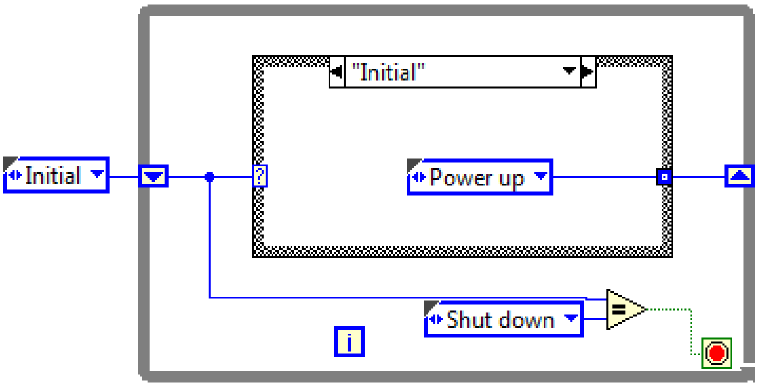
- While Loop – Executes multiple iterations until a stop condition is met.
- Case structure – Contains unique sections of code that represent each state of the state machine.
- State Enum – Defines all possible states of the state machine.
- Shift Register – Contains the state specified by the previous iteration to execute on the current iteration.

The state machine architecture can be used to implement complex decision-making algorithms represented by state diagrams or flow charts. Furthermore, besides having the powerful ability to implement decision making algorithms, state machines are also functional forms of application planning (Thuyphamxuan, 2013). Figure 8 shows an example of state machine architecture.

In the state diagram shown in Figure 8 (a), each state describes the actions that are performed when the control process is in that state. Furthermore, the transitions between states as shown by means of arrows describe when and how the process can move from one state to another. It can be seen from the state diagram that some states have only one transition whereas others have more than one transition. For example, the *Initial* state only has only one possible transition, i.e from *Initial* state to *Power up* state, while the *Check status* state has four possible transitions, from *Check status* state to: *Fire*, *Active cooling*, *Shut down and Passive cooling* states.



(a) State diagram



\

(b) State machine

Figure 8: State machine architecture: (a) state diagram and (b) state machine (adapted from Thuyphamxuan, 2013)

2.3.5.2 User interface event programming

Thuyphamxuan (2013) notes that the event structure makes it possible to implement event-driven programming in LabVIEW. Also noted by Thuyphamxuan (2013) is that LabVIEW takes advantage of features in the operating system (OS) to be notified when user interface activity occurs, making the OS to give the CPU to other programs running while the application is idle.

When the event structure executes, LabVIEW puts the VI to sleep until one of the events it is configured to listen for occurs, just as a Wait on Occurrrence sleeps until an occurrence is fired. When an event of interest happens, the Event Structure automatically wakes up and executes the appropriate sub-diagram to handle that event (Thuyphamxuan, 2013). Figure 9 illustrates the use of the User Interface Event programming architecture.

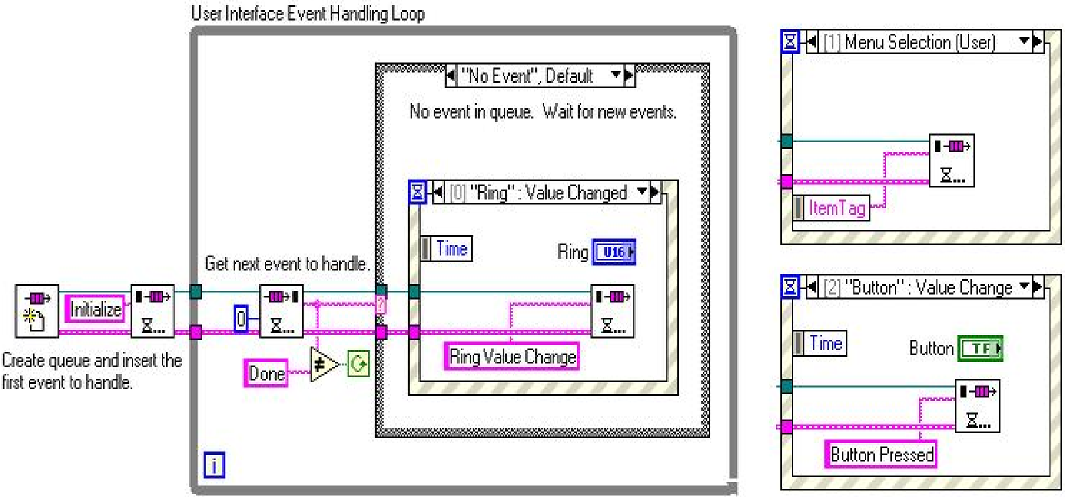

User Interface Event Handling Loop
"No Event", Default
No event in queue. Wait for new events.
[0] "Ring" : Value Changed
Time
Ring
U16
Ring Value Change
Get next event to handle.
Initialize
0
Done
Create queue and insert the first event to handle.
i
[1] Menu Selection (User)
ItemTag
[2] "Button" : Value Change
Time
Button
TF
Button Pressed

Figure 9: User interface event programming (Thuyphamxuan, 2013).

2.3.5.3 Master/slave architecture

The master/slave architecture is used when two or more processes need to run simultaneously and continuously, but at different rates (Thuyphamxuan, 2013). The master/slave architecture consists of multiple parallel loops. Furthermore, the loops may execute tasks at different rates. One of the parallel loops acts as the master and the others act as slaves. The master loop controls the slave loops and communicates with them using messaging architectures such as local or global variables, occurences, notifiers or queues (Thuyphamxuan, 2013).

Thuyphamxuan (2013) state that the master/slave architecture is very advantageous when creating multitask applications. This architecture gives a modular approach to application development because of its loops functionality.

2.3.5.4 Producer/consumer architecture

Figure 10 shows an example of the producer/consumer architecture. The producer/consumer architecture is based on the master/slave pattern and enables data sharing between multiple loops running at different rates. The parallel loops in this architecture are broken down into two categories: those that produce data and those that consume the data produced (Thuyphamxuan, 2013). Furthermore, Thuyphamxuan (2013) noted that the producer/consumer architecture has the ability to easily handle multiple processes at the same time while iterating at individual rates.

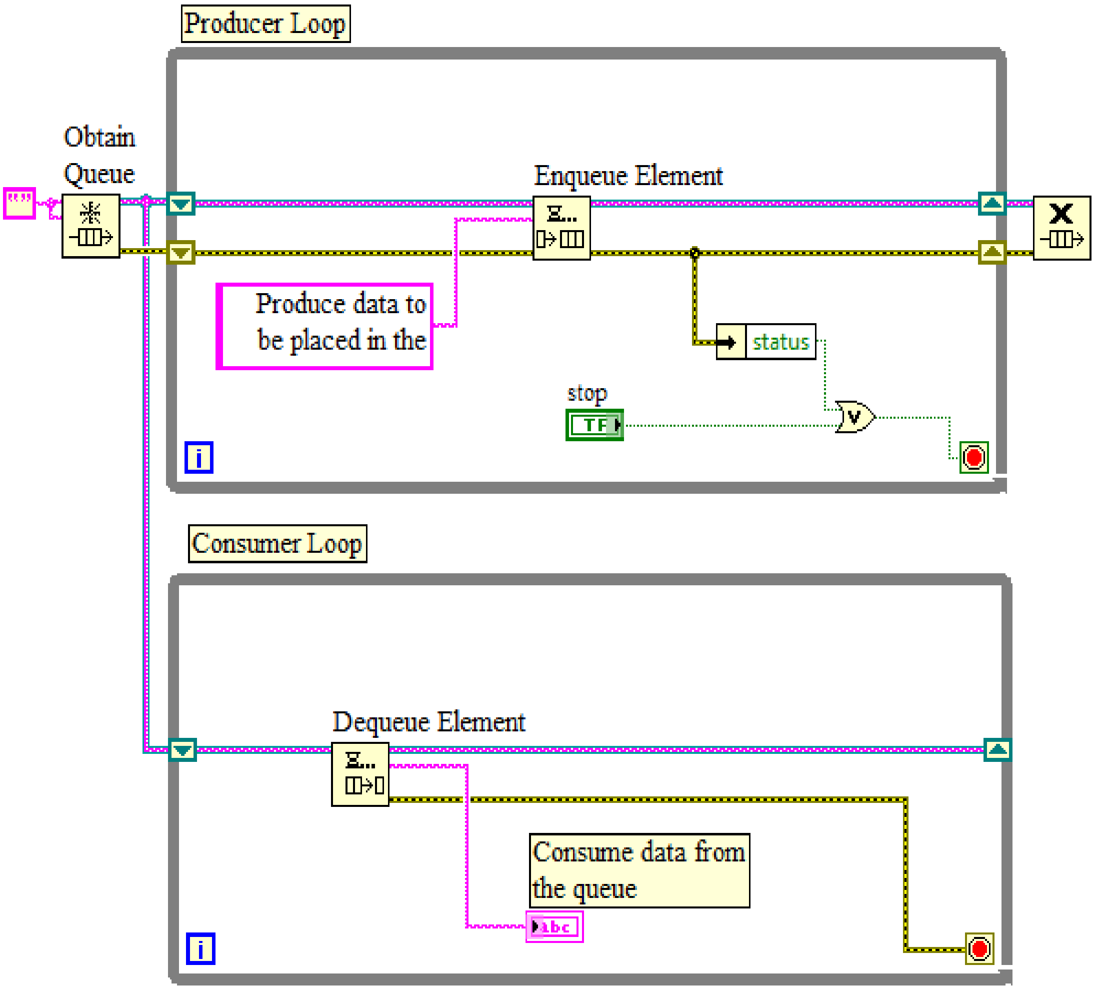


Figure 10: Producer/consumer architecture (adapted from Thuyphamxuan, 2013).

2.3.6 Industrial applications of LabVIEW

National Instruments (2010) point out that LabVIEW is widely used across a variety of industries, such as aerospace, consumer electronics and automotive, to create flexible automated test systems that meet the demands of today's business environment. Furthermore, National Instruments (2010) notes that more test managers and engineers are required to develop automated test systems faster to keep up with accelerated product development cycles and globalization of design and test. With the LabVIEW graphical development environment, drag-and-drop graphical icons are used to rapidly develop test software.

National Instruments (2012) notes that LabVIEW is used in industry for instrument control using the following tools:

- Instrument drivers
- Instrument I/O Assistants
- VISA Application Programming Interface (API)

In LabVIEW an instrument driver is defined as a set of VIs that communicates with an instrument. Each VI corresponds to a programmatic operation, such as configuring, reading from, writing to and triggering an instrument. Instrument drivers in LabVIEW simplify instrument control and reduce test program development time by eliminating the need for learning the complex, low-level programming commands for each instrument (National Instruments, 2012).

Instrument I/O Assistant provides a user interface to interactively write commands to a device, read data that the device returns, and specify how to parse the response into a format relevant to a particular application. What is more, Instrument I/O Assistant simplifies the challenge of writing instrument control applications by automatically generating code from your configurations in your environment (National Instruments, 2013). The Instrument I/O Assistant is launched by the Instrument I/O Express VI which is found on the Function>>instrument I/O palette. The Instrument I/O Assistant can be used to communicate with message-based instruments and graphically parse the response. It organizes instrument communication into ordered steps (National Instruments, 2013).

VISA is a standard I/O API for instrumentation programming and can control GPIB, serial, USB, Ethernet, PXI or VXI instruments. The I/O controls on the *Controls>>I/O* and *Controls>>Classic I/O* palettes can be used to specify the instrument or device resource to communicate with while the VIs and functions on the *Functions>>Instrument I/O>>VISA* palette are used to build VIs that control instruments (National Instruments, 2012).

2.3.7 Benefits of LabVIEW in academic research

National Instruments (2012) gives the many benefits of using an integrated development environment and programming language such as LabVIEW in academic research. The following are the benefits of LabVIEW in academic research:

- Compiled code speed and ability to create distributable EXEs and DLLs
- Powerful, flexible and scalable design (open, connects to external libraries and third-party tools)
- Easy to learn, use, maintain and upgrade (intuitive graphical programming, using graphical constructs)
- One tool for design, prototyping and deployment
- Multidisciplinary use (same easy graphical programming language for different applications and domain experts in different disciplines in science and engineering)
- Tight software-hardware integration (supports wide variety of data acquisition and embedded control devices)
- Multiplatform (Windows, Mac OS, Linux, RTOSs)
- Easy integration with legacy and traditional instruments (serial, GPIB, CAMAC, VME, and so on)
- Ability to solve and execute complex algorithms in real time (ODEs, PDEs, BLAS-based linear algebra, signal processing and analysis, optimization, and so on) using real-world signals (A/D)
- Bridge to industry – same tools used in academia and industry (academic-to-industry transition easier, technology transfer more transparent)
- Shorter time to prototype, time to discovery, time to deployment, and potentially time to market
- Help to develop better, faster algorithms (algorithm engineering)

Lim (2004) developed a LabVIEW program demonstrating the ease of programming a 5-DOF robot arm to move a wooden ring from one peg to another. The use of graphical programming was proposed to reduce the time overhead required to adapt the robot code, thus minimizing programmer effort for each software upgrade cycle. Furthermore, Shyam R.Nair (2012) introduced LabVIEW based control of a robotic arm. The paper focuses on designing a robotic arm for picking and placing an object controlled using LabVIEW. The LabVIEW program was designed in such a way that the user could input the coordinates of the object's position in real time. The action of picking and placing was given through the Front Panel as shown in Figure 11.

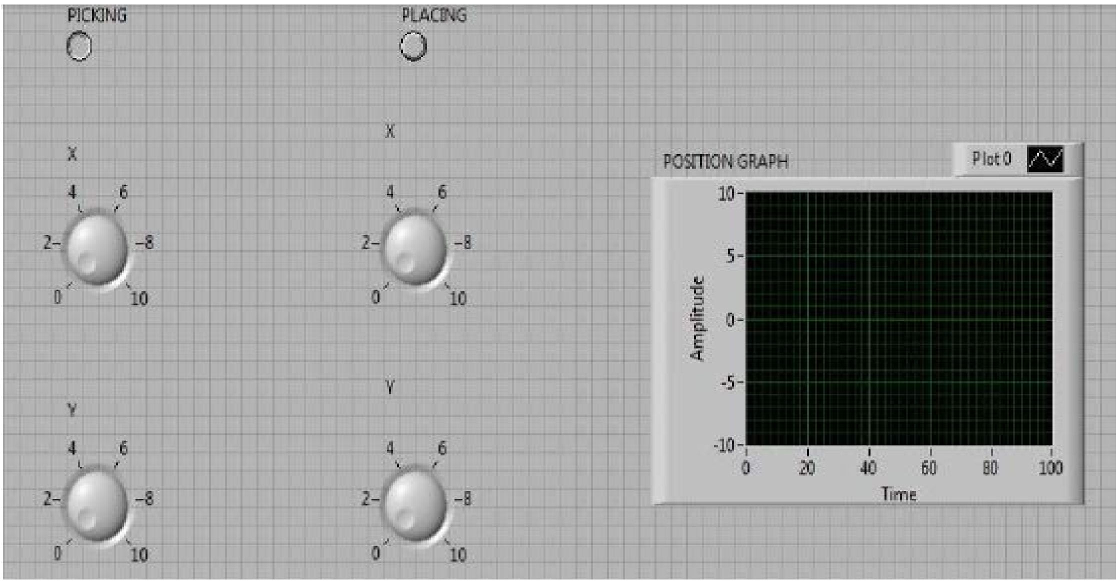
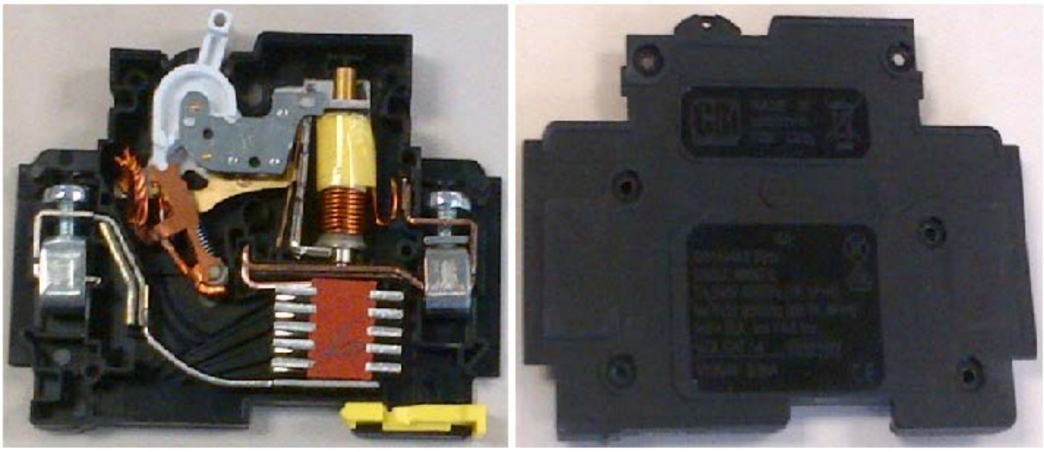


Figure 11: Front panel for the picking and placing actions (Nair, 2012).

3 CASE STUDY DESCRIPTION

This thesis uses a rivet feeder subsystem as a case study. The rivet feeder subsystem is part of the RMS cell used at the Department of Mechanical and Mechatronic Engineering at Stellenbosch University. The cell is aimed at automating some of the assembly and testing processes for circuit breakers produced by CBI Electric. CBI Electric produces different ranges of circuit breakers, such as the C-range, D-range and the Q-range. This thesis considers the Q-range of circuit breakers, which mainly comprises the base assemblies and shells as shown in Figure 12.



(a) Base assemblies **(b) Shell**

Figure 12: Circuit breaker: (a) Base assemblies and (b) Shell.

3.1 Reconfigurable manufacturing system cell

3.1.1 Physical cell architecture

Figure 13 shows the layout of the physical cell architecture of the RMS cell. The RMS cell comprises several stations, such as manual placing stations, machine vision inspection stations, manual assembly stations, electronic test stations, printing stations, stacking and buffering stations, and rivet placing stations. Furthermore, the RMS cell has two conveyors, namely the main conveyor and the secondary conveyor. The main conveyor is used for transporting pallets to all the stations around it. The secondary conveyor is used for receiving the riveted assemblies of circuit breakers. The stations in the RMS cell are discussed in the paragraphs that follow.

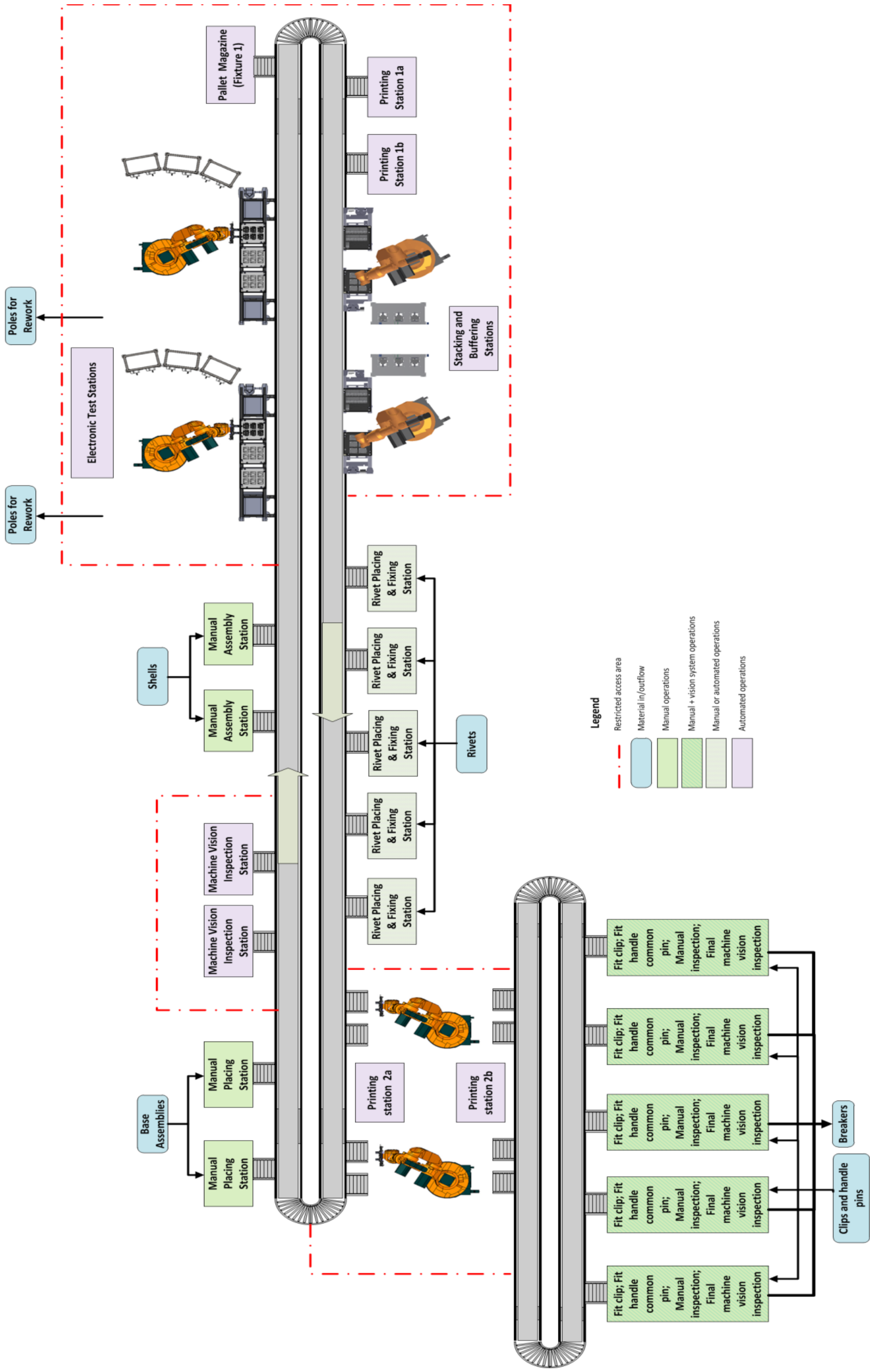

Pallet Magazine (Fixture 1)
Printing Station 1a
Printing Station 1b
Stacking and Buffering Stations
Electronic Test Stations
Poles for Rework
Poles for Rework
Rivet Placing & Fixing Station
Rivet Placing & Fixing Station
Rivet Placing & Fixing Station
Rivet Placing & Fixing Station
Rivet Placing & Fixing Station
Rivets
Shells
Manual Assembly Station
Manual Assembly Station
Machine Vision Inspection Station
Machine Vision Inspection Station
Base Assemblies
Manual Placing Station
Manual Placing Station
Printing station 2a
Printing station 2b
Fit clip; Fit handle common pin; Manual inspection; Final machine vision inspection
Fit clip; Fit handle common pin; Manual inspection; Final machine vision inspection
Fit clip; Fit handle common pin; Manual inspection; Final machine vision inspection
Fit clip; Fit handle common pin; Manual inspection; Final machine vision inspection
Fit clip; Fit handle common pin; Manual inspection; Final machine vision inspection
Breakers
Clips and handle pins
Legend
Restricted access area
Material in/outflow
Manual operations
Manual + vision system operations
Manual or automated operations
Automated operations

Figure 13: Physical cell architecture.

When production commences, the empty pallets, that are stored in the pallet magazine, are transported to the manual placing station by means of a conveyor. At the manual placing station, the base assemblies of the circuit breakers are placed on the pallets. The manual placement of base assemblies is followed by visual inspection where a camera checks whether all parts are present in the base assemblies. Thereafter, the pallets are transported to the assembly station where the shells are placed on the bases. The resulting assembly of the shells and the base assemblies can be referred to as single poles.

From the manual assembly station the pallets are transported to the electronic test station where the circuit breakers undergo an electrical test. If breakers fail the electrical or the visual inspection, they are returned for rework, whereas the breakers that pass the electrical test are transported to the first printing station where circuit breaker information is laser printed on the shells of single poles.

From the first printing station, circuit breakers are transported to the stacking and buffering station, where the circuit breakers are buffered until the number of poles required is attained. When all the required poles are available, they are stacked to form the required width-pole assemblies. When the circuit breakers have been stacked, the pallets are transported to the rivet placement and fixing stations.

The rivet placement and fixing stations are responsible for feeding rivets into the circuit breaker assemblies and fixing them. As can be seen from Figure 13, there are five rivet placement stations. Having a multiple number of these stations is necessary to achieve the required throughput rate. The length of the rivet to be fed is dictated by the required number of poles that have been stacked. For this research only the rivet lengths for the 2-pole and 4-pole circuit breakers will be fed. The rivet placement station is discussed in detail in section 3.2. The subsequent fixing is not considered in this thesis.

After the rivets have been fixed, the pallets are transported to the second printing station where information is laser printed on the front edge of the breakers. When printing has been completed, the plastic clips and handle common pins are fitted. Machine vision inspection is then carried out to check whether the clips and handle common pins are present or not.

3.1.2 Cell control architecture

The cell control architecture of the RMS cell consists of the PROSA holons, discussed in section 2.2.2.1. Each station in the RMS cell is contained in a Resource Holon within the cell control architecture. Furthermore, each Resource Holon in the cell control architecture includes a station controller. The function of the cell controller (CC) is to manage production schedules, as well as to coordinate all the stations in the cell through communication with each controller. Therefore, the CC ensures that each station performs its task on the products.

The CC and the controller of the rivet feeder station exchange information in XML strings, using the TCP/IP protocol. Furthermore, the rivet feeder station and the CC implement asynchronous communication so that there is no loss of information in the RMS cell, while communication delays do not delay execution of tasks wherever possible.

3.2 Rivet feeder station

This section begins by highlighting the design requirements of the rivet feeder station, which entails all the important characteristics that the station must possess in order to meet the research objective. The section also discusses the design concept development and selection for the rivet feeder station using the morphological chart method and the weighted objectives method, respectively. Finally, the station physical architecture is then discussed.

3.2.1 Design requirements

It should be noted that the current circuit breaker design does not adequately provide for automated placement of the rivets, since there are no chamfers on the rivet holes and the clearance between the rivets and holes are too small. However, changes that may be required to the product design are not within the scope of this thesis.

The design of the rivet feeder station is aimed at automating the process of placing/feeding rivets into assemblies of circuit breakers. As mentioned earlier, the rivet feeder station is part of the RMS cell – a case study for CBI Electric. Therefore, in order to ensure that the rivet feeder station meets its intended goal in the RMS cell, design requirements incorporating both functional requirements and technical performance measures, were developed.

The transfer of pallets from the conveyor system to the rivet feeder station, as well as the design of the XYZ positioning table that positions the rivet holes under the rivet feeder, were considered to be outside the scope of this thesis. More details of the XYZ positioning table are given in section 3.2.3.4.

3.2.1.1 Functional requirements

The functional requirements of the rivet feeder subsystem are as follows:

- Receive rivets in bulk container
- Orientate rivets in required direction
- Transport rivets
- Singulate rivets
- Place rivets in positions obtained from the cell controller
- Inform the cell controller of the state and diagnostics of the station

- Interface with the cell controller using TCP/IP and XML strings

3.2.1.2 Technical performance measures

The technical performance measures developed include the following:

Rivet length:
- 13.2 mm and 51.6 mm

Cycle time:
- Six (6) seconds per breaker

Floor space and height off the ground:
- Space of not more than 1000 mm x 3000 mm
- Positioning table, at home position: 1030 mm off the ground

Reliability:
- Not less than 99.5%

Operator requirements:
- Not more than one operator needed
- Read subsystem's diagnostic status via the front panel (user interface)

Target purchase cost:
- Not exceeding R100 000, which is the estimated cost for the rivet feeder subsystem

Weight:
- Circuit breaker fixture and its bracket: not more than 5kg

3.2.2 Concept development and preliminary evaluation

The morphological chart method, as shown in Table 2 was used to generate alternative design concepts for the rivet feeder station. This involved identifying the essential functions of the rivet feeder subsystem and then listing for each of the functions the alternative ways, or sub-solutions, of achieving that function. The weighted objectives method was then used to select the best concept.

Table 2 shows the rivet feeder functions in the first column and the means of achieving the functions in columns two to four. Therefore, different combinations of design alternatives (concepts) can be developed from Table 2. However, not all the concepts were considered for evaluation due to compatibility and feasibility limitations. The rivet feeder functions are discussed in the paragraphs that follow, with reference to their corresponding means.

Table 2: Morphological chart for the rivet feeder station

	MEANS		
FUNCTIONS	1	2	3
Contain rivets	Vibratory bowl feeder	Rotary/centrifugal feeder	
Orient rivets	Slotted trap	Groove	Narrowing track and slotted trap
Transport rivets	Guided rail	Tube	Robot
Singulate rivets	Pneumatic actuator	Spring-loaded device	
Feed rivets	Guiding jaws	Pick-n-place mechanism	

- Contain rivets

This function can be achieved by using a vibratory bowl feeder or rotary/centrifugal feeder. Unlike in a vibratory bowl feeder, it is very difficult to orient parts in a rotary or centrifugal feeder as there are no vibrations and movement of parts is very fast. Therefore, the vibratory bowl feeder is a better choice over the rotary or centrifugal feeders, to perform the function of containing rivets.

- Orient rivets

The function of orientating rivets can be achieved by means of the slotted trap, groove, and narrowing track with a slotted trap. Despite the slotted trap being able to reliably orientate rivets of varying lengths, the slotted trap shown in Figure 14 has a limitation in the sense that it can only orientate rivets that approach with their tails first. The slotted trap is also prone to cause rivet jam when two or more rivets are orientated at the same time.

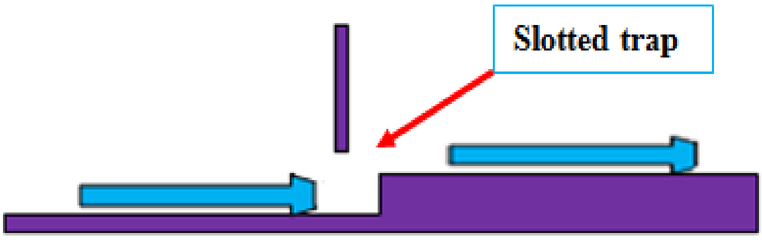


Figure 14: Slotted trap on the rivet track.

Unlike the slotted trap, the groove shown in Figure 15 is able to orientate rivets in a vertical direction whether the rivets approach it with their heads or tails first. Furthermore, the use of a groove eliminates the risk of rivet jam since all rivets that do not get into the groove are forced back to the bowl feeder. However, the use of a groove results into a decrease in the motion of rivets because the rivets are only pushed by their heads and not their entire mass.

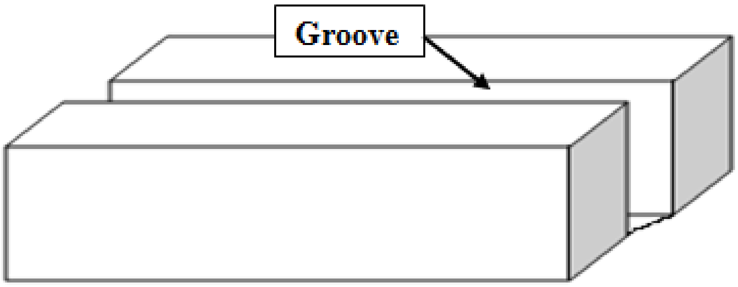


Figure 15: Grooved track for rivet orientation.

The last means of orientating rivets is by making use of the narrowed path which incorporates a slotted trap. This entails positioning a slotted trap at the end of the narrowed path. Despite causing a decrease in the throughput rate of the orientated rivets, the narrowed path singulates rivets before they are orientated, thereby preventing rivet jam. Furthermore, narrowing the path has been proven to successfully orientate rivets that approach with their tails first. Therefore, the rivets that approach with their heads first are forced back to the vibratory bowl. The slotted trap is used for orientating rivets that may pass through the narrowed path with their heads first.

- Transport rivets

  After rivets have been orientated, they are transported to the singulation device. Transporting rivets can be achieved by means of: guided rail, flexible and transparent tube or a robot. Transporting rivets using the guided rail is unreliable as vibration is required for rivet propulsion. Using a tube or robot to transport rivets is more reliable compared to using the guided rail. The tube ensures a smooth motion of rivets, as well as the visual inspection of rivet orientation. Although the use of a robot is reliable, it is costly to implement. Therefore, using a flexible and transparent tube to transport rivets is the best of the three choices.

- Singulate rivets

  Singulating rivets can be achieved by using either a pneumatic actuator or a spring-loaded device. Both the pneumatic actuator and the spring-loaded device can provide reliable isolation of rivets during feeding/placement. However, considering the cost aspects, the pneumatic actuator is more costly compared to the spring-loaded device. Therefore, the spring-loaded device was chosen for singulating rivets.

- Feed/place rivets

  The function of feeding/placing rivets on assemblies of circuit breakers can be accomplished by means of either guiding jaws or a pick-n-place mechanism. The guiding jaws can be used to align singulated rivets to the holes of an assembly of circuit breakers. However, because of the small clearance between the holes of circuit breakers and the rivets, rivet placement is not guaranteed with the guiding jaws. In addition, some form of rivet propulsion is necessary to force rivets through the holes.

  The second choice, pick-n-place mechanism, consists of pneumatic actuators, namely a pneumatic gripper for picking singulated rivets and a pneumatic cylinder to move the pneumatic gripper. Unlike the guiding jaws, the pick-n-place mechanism ensures a reliable placement of rivets as it provides a force to push the rivets through holes of circuit breakers.

The different design concepts generated are as follows:

- *Concept one*: This concept consists of the vibratory bowl feeder, slotted trap, tube, pneumatic actuator and guiding jaws. This concept uses a transparent tube to transport rivets from the vibratory bowl feeder. Rivet orientation could also have been achieved using a rotary or centrifugal feeder. However, the decision to use a vibratory bowl feeder was arrived at after considering that it is easier to orient parts with vibratory bowl feeders than with a rotary or centrifugal feeder. Furthermore, the vibratory bowl feeder was chosen because its vibratory bowl is suitable for the geometry of rivets fed.

- *Concept two*: This concept is similar to concept one, except that it uses a pick-n-place mechanism in place of guiding jaws.
- *Concept three*: This concept consists of the vibratory bowl feeder, slotted trap, tube, and spring-loaded singulation device and pick-n-place mechanism.
- *Concept four*: This concept consists of the vibratory bowl feeder with a grooved track near the discharge, tube, pneumatic actuator and guiding jaws.
- *Concept five*: This concept is similar to concept three, except that rivet orientation is made more effective by means of incorporating a slotted trap into the narrowed rivet path. This concept consists of the vibratory bowl feeder, a narrowed track and slotted trap on the orientation device, tube, and spring-loaded singulation device and pick-n-place mechanism.

The qualitative evaluation of the alternative design concepts was based on the following design requirements discussed in section 3.2.1:

- Cycle time
- Reliability
- Reconfigurability
- Cost
- Weight
- Floor space

The decision matrix method discussed by Ullman (2003) was used for comparing alternative design concepts. To be able to compare the concepts, concept three was chosen as the datum. Therefore, the other design concepts were compared against concept three.

The design requirements were allocated relative importance weightings by ranking them. That is, the most important requirement was listed first and the least important one was listed last, as shown in Table 3. Row totals in Table 3 show the ranking of the design requirements, for instance 5 shows that throughput rate is the requirement with the highest rank and 0 shows that weight is the requirement with the lowest rank. It should also be noted that ones and zeros on the table denote relative importance of requirements, for example if the objective *reliability* is more important than the objective *cycle time*, then *reliability* gets a 1 and *cycle time* gets a 0.

Table 3: Ranked requirements

Objective	Cycle time	Relia-bility	Reconfi-gurable	Cost	Weight	Floor space	**Row total**
Cycle time	-	1	1	1	1	1	5
Reliability	0	-	0	0	1	1	2
Reconfigurable	0	1	-	1	1	1	4
Cost	0	1	0	-	1	1	3
Weight	0	0	0	0	-	0	0
Floor space	0	0	0	0	1	-	1

The evaluation of the design concepts was done by using three ways of rating, namely the *plus (+)* score, the *minus* (-) score and the *same* (*S*) score. The *plus* score was used to signify that a particular concept was better than the *datum* concept. The *minus* score was used to show that the concept was not as good as the *datum* concept, since it did not meet the design requirement. Furthermore, the use of the *same* score meant that the concept was the same as the *datum* concept.

Table 4 shows the decision matrix that was used to select the best design concept for the rivet feeder station. The best concept was selected by calculating the number of *plus* scores, the number of *minus* scores, the overall total and the weighted total. From the decision matrix, it can be seen that the alternative design concept with the best score was concept 5 (C 5).

Table 4: Decision matrix for the rivet feeder station

Criteria	**Importance**	**Alternatives**				
		C 1	**C 2**	**C 3**	**C 4**	**C 5**
Cycle time	5	-	-	-	-	S
Reconfigurable	4	S	S	-	S	S
Cost	3	+	-	-	-	+
Reliability	2	S	S	-	S	+
Floor space	1	-	S	-	S	S
Weight	0	+	S	-	+	S
Total +		2	0	-	1	2
Total -		2	2	-	2	0
Overall total		0	-2	-	-1	2
Weighted total		-3	-8	-	-8	5

3.2.3 Station physical architecture

The concept selected in the preceding section was developed into a prototype after conducting a subsystem functional analysis shown in Appendix A. This section describes the architecture of the mechanical parts. The rivet feeder station is made up of the vibratory bowl feeder, singulation device, pick-n-place mechanism and the XYZ positioning Table. The physical architecture is as illustrated in Figure 16. Furthermore, detailed hardware components with their respective suppliers are given in Appendix B.

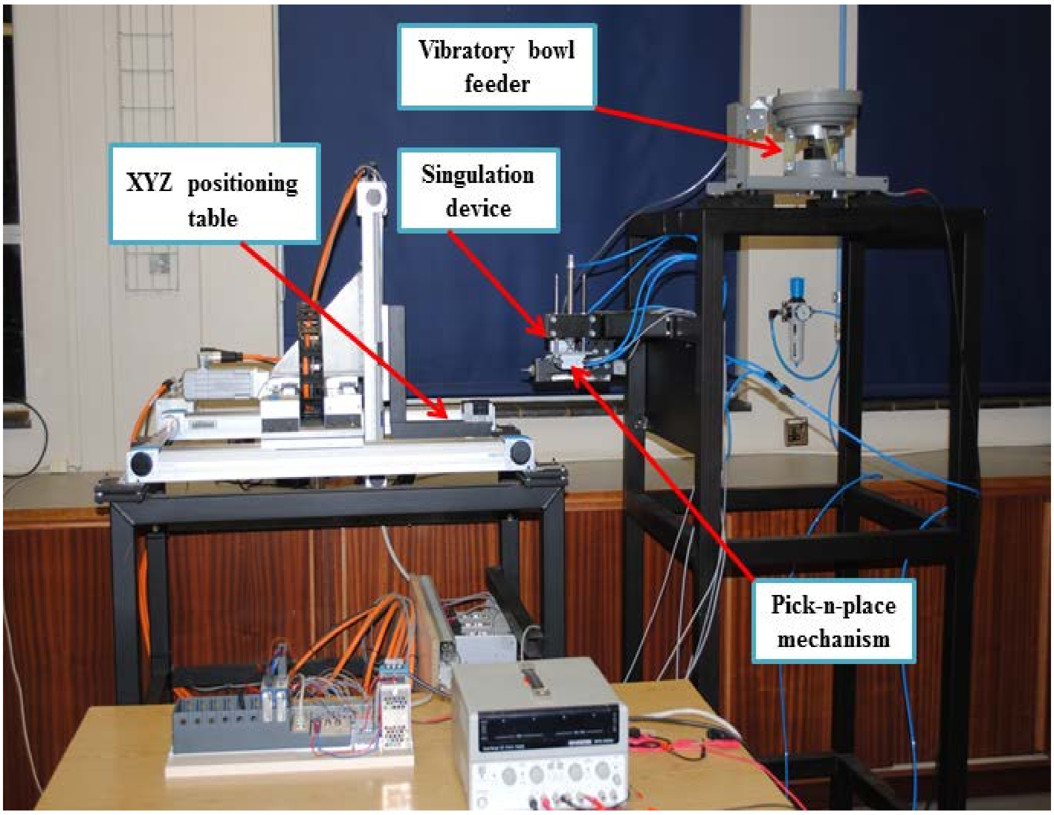


Figure 16: Rivet feeder station physical architecture.

3.2.3.1 Vibratory bowl feeder

The vibratory bowl feeder shown in Figure 17 is used in the rivet feeder station to orient rivets in the required direction and presents them to the transporting tube. In this application, rivet orientation entails extracting rivets from the vibratory bowl feeder with their heads up.

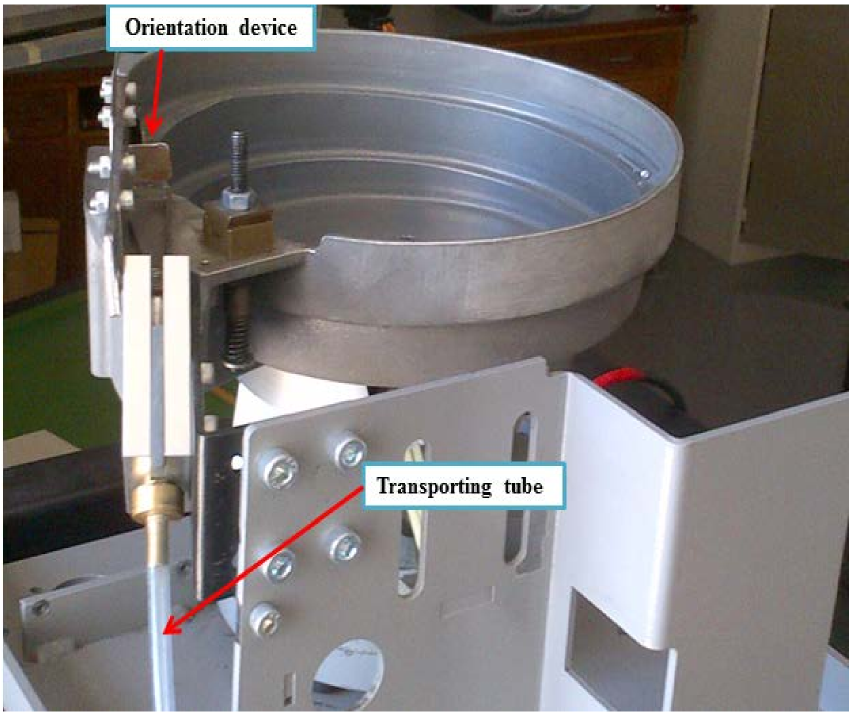
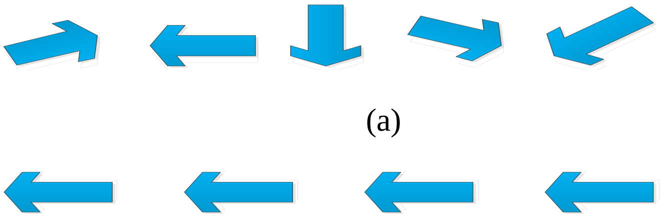


Figure 17: Vibratory bowl-feeder, orientation device and transporting tube.

Rivet orientation in the required position ensures a consistent feed rate and also prevents jamming of rivets during transportation to the singulation device. Therefore, operational disturbances of the rivet feeder are avoided with a reliable orientation device. A reliable orientation device must be able to set the orientation of rivets illustrated in Figure 18 (a) into the required orientation of rivets shown in Figure 18 (b).


(a)

(b)

Figure 18: Rivet orientation (a) different rivet orientations (b) required rivet orientation.

The vibratory bowl feeder has an electromagnetic vibrating drive unit and a spiralled track inside the bowl. In addition, the vibratory bowl feeder has three leaf springs between the vibratory bowl and the base of the electromagnetic vibrating drive unit. The leaf springs constrain the structure of the vibratory bowl feeder thereby causing torsional vibration. When the electromagnetic vibrating drive unit is switched ON, the torsional vibration is transmitted to the vibratory bowl. As a result, the vibratory bowl is set into vibratory motion causing the rivets to move up the spiralled track, in a hopping-like manner. As the rivets approach the exit of the vibratory bowl they are forced to go through a narrowed path so that they are presented to the orientation device in a single file to prevent jamming before and after orientation.

The narrowed path on the vibratory bowl was made by trimming the edge of the base plate and placing a piece of sheet at an angle, as can be seen from Figure 19. As a result of the narrowed path, some rivets are obstructed by other rivets on the track and are therefore forced to fall back to the vibratory bowl. Furthermore, the majority of the rivets that fall back to the vibratory bowl are those that approach the orientation device in a wrong direction due to the concentration of mass near the head of the rivet.

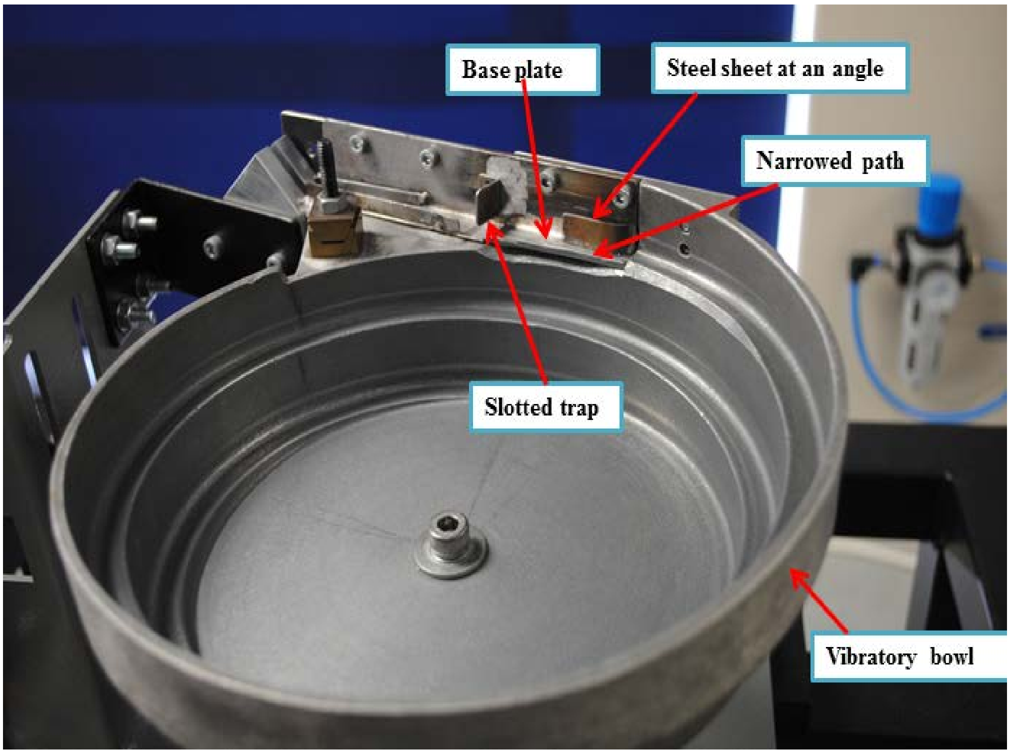


Figure 19: Orientation device on the vibratory bowl.

The orientation device is used for rejecting rivets that are not in the required position (orientation). This device is basically a slotted in-bowl tool made by placing a 1mm piece of steel sheet at a distance of 2.5 mm above the 3 mm base plate and with an offset of 4 mm in the horizontal direction from the base plate. Successful orientation of rivets was attained after the orientation device was adjusted mechanically by varying the size of its slot.

The orientation device on the vibratory bowl feeder can orient different sizes of rivets. Nevertheless, for experimental purposes only two rivet sizes were used namely the size for the 2-pole circuit breakers and the size for the 4-pole circuit breakers. In order to evaluate the orientation throughput rate of the vibratory bowl feeder, orientation experiments were conducted on both sizes of rivets. Figure 20 shows the graphs of results obtained from the experiments. It can be deduced from the graphs that the orientation throughput rate of the vibratory bowl feeder was higher with the 2-pole size of rivets compared to the 4-pole size of rivets.

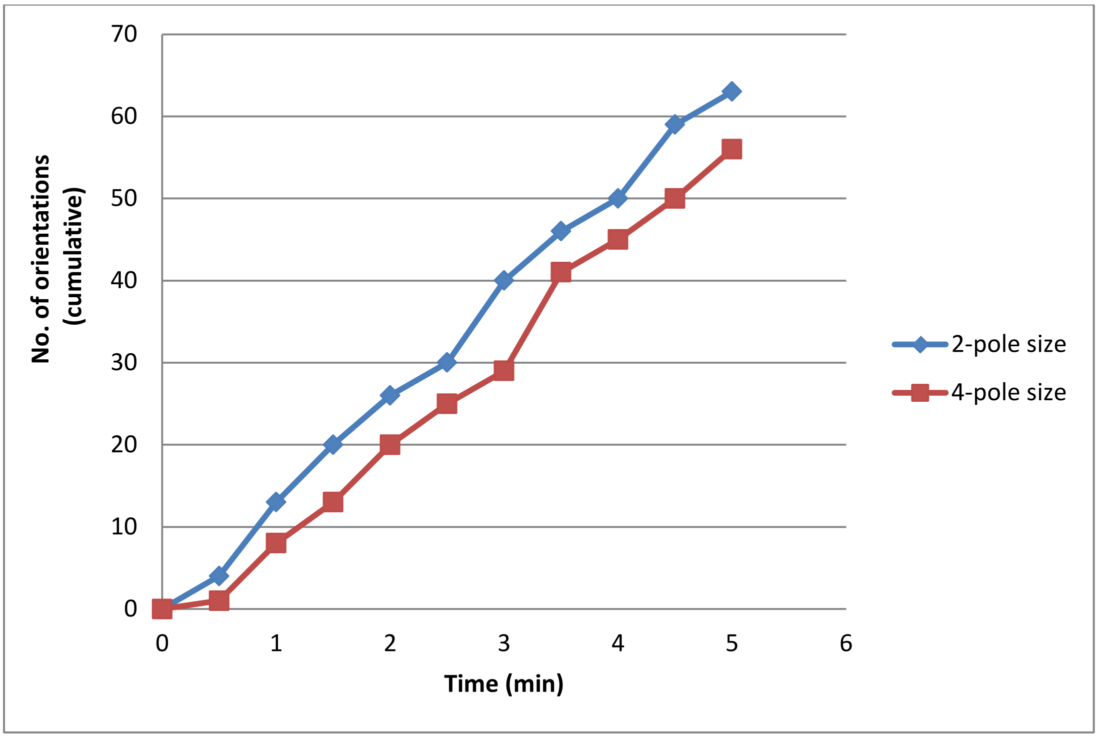


Figure 20: Measured orientations vs. time.

It can be noted from Figure 20 that the experimented feed rates did not meet the design requirements. The low feed rate can be attributed to the vibratory bowl not running at full capacity. It is the view of the author that the design requirement can be met by running the vibratory bowl at full capacity and also feeding rivets from multiple vibratory bowls.

3.2.3.2 Singulation device

The singulation device was custom made and serves the purpose of presenting rivets, from the transporting tube, one at a time to the pick-n-place mechanism, in a vertical orientation with heads up. The vertical orientation of rivets is needed to enable the pick-n-place mechanism to easily pick and feed the rivets into circuit breaker holes. The singulation device is described in detail in Appendix D.

3.2.3.3 Pick-n-place mechanism

In this thesis *pick-n-place mechanism* refers to a device designed to pick rivets from the singulation device and place them into the circuit breaker holes. From the operational point of view, the pick-n-place mechanism is basically a 1-DOF robot, since motion is constrained in the vertical direction. The pick-n-place mechanism consists of pneumatic actuators, namely a standard air cylinder (DSNU-16-100-P-A) and a pneumatic gripper (DHPS-20-A). The pick-n-place mechanism is as shown in Figure 22. The choice of the DSNU-16-100-P-A standard air cylinder was motivated by the need for a stroke that is greater than 53 mm, which is the length of the longest rivet. Additionally, each pneumatic actuator has a 5/2 flow control valve. The flow control valves have two switching positions for extending and retracting the standard air cylinder and also for opening and closing of the pneumatic gripper jaws. Furthermore, the pneumatic gripper has a proximity sensor fitted to it. The 5/2 flow control valve and the proximity sensor are discussed in sections 4.2.2 and 4.2.3, respectively.

The pneumatic gripper has the function of picking rivets in their vertical orientation and feeding them to the circuit breaker poles. The standard air cylinder's function is to produce vertical motion (up and down) of the pneumatic gripper.

The pneumatic gripper jaws used for picking and placing rivets have grooves on the gripping surfaces as shown in Figure 21. The grooves ensure effective gripping of rivets and also to prevent the skewing of rivets after they are picked from the singulation device. Furthermore, the pneumatic gripper jaws are shaped to avoid interference between the pneumatic gripper bottom surface and the stacked circuit breaker poles.

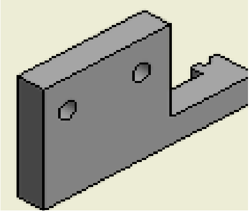


Figure 21: Profile of the gripper jaws.

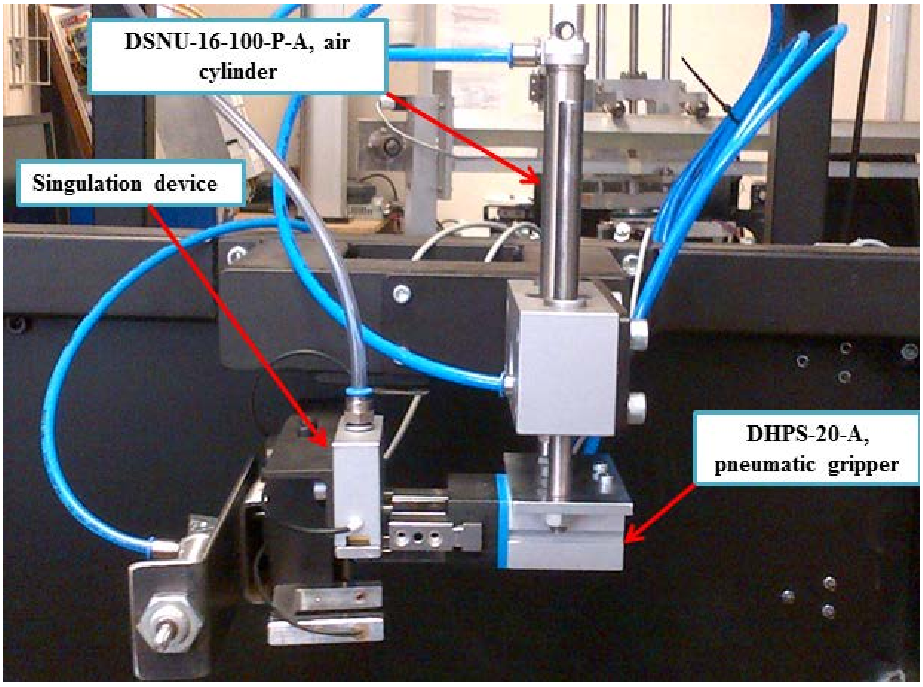


Figure 22: Singulation device and Pick-n-Place mechanism.

3.2.3.4 XYZ Positioning table

In order to position the circuit breakers for rivets to be fed, a modular Cartesian robot designed by Mulubika (2013) is used as a positioning table. In his thesis, Mulubika (2013) stated that the modular Cartesian robot was designed on the principles of reconfigurable machines, i.e. modular structure and software components.

The XYZ positioning table shown in Figure 23 has three degrees of freedom. Movement of the XYZ positioning table in the X and Y axes entails setting the circuit breakers in a position specified by the cell controller. Movement of the positioning table in the Z-axis allows rivet placement into circuit breaker poles of varying thicknesses. Furthermore, the homing positions for the axes are: X and Y axes home positions are to the right side of the X-axis motor while Z-axes homes at the botton right of the X-axis motor.

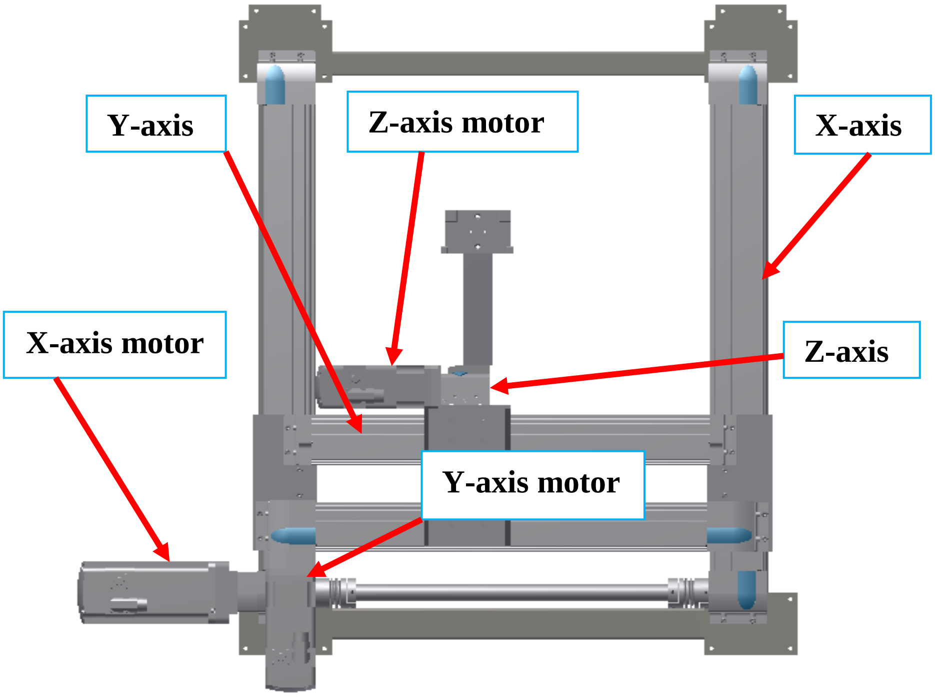
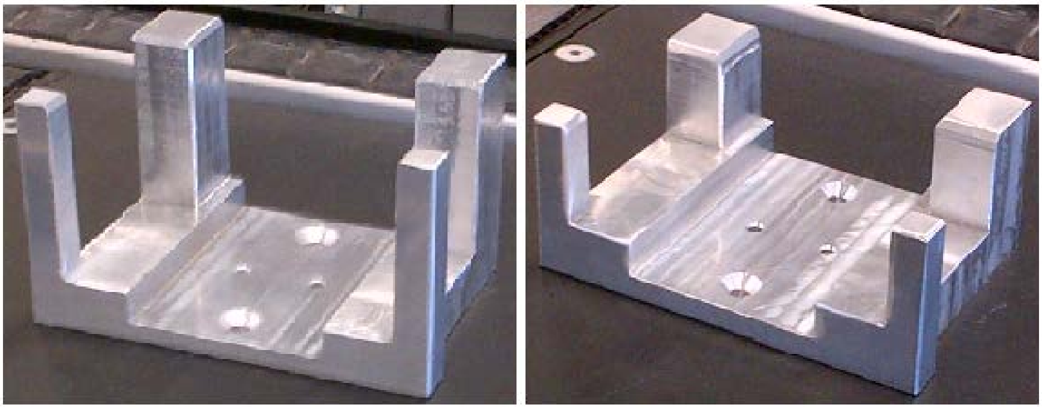


Figure 23: Top view of the XYZ positioning table

The Z-axis of the positioning table carries an L-shaped bracket for the circuit breaker fixtures. The L-shaped bracket was made by welding together two angle bars of size: 40 x 40 x 5 x 250 mm. The two circuit breaker fixtures are identical, except that they have different heights. One fixture is used for stacking 4-poles while the other is used for stacking 2-poles. The fixtures were machined from aluminium 6082T6 owing to easy machinability of this aluminium. Figure 24 shows the design of the two fixtures.



(a) (b)

Figure 24: Circuit breaker fixtures: (a) 4-pole fixture and (b) 2-pole fixture.

4 RIVET FEEDER STATION CONTROLLER

4.1 Control architecture

In this thesis, a holonic control architecture was adopted for developing the LabVIEW based controller. A holonic architecture was selected to facilitate reconfigurability of the rivet feeder subsystem. The PROSA reference architecture is a widely recognised holonic control architecture and was chosen to develop the LabVIEW based controller. Achieving reconfigurability of the rivet feeder subsystem was possible due to the fact that PROSA architecture has a high degree of self-similarity which reduces the complexity to integrate new components and enables easy reconfiguration of the system (Van Brussel, 1998).

Figure 25 shows the architecture of the LabVIEW based controller, consisting of six holons, i.e. Coms Holon, Request Manager Holon, Order Holon, Product Holon Manager, Pick-n-Place Holon and XYZ Table Holon, in terms of PROSA. The Coms Holon and the Request Manager Holon are both staff holons while the Pick-n-Place Holon and the XYZ Table Holon are the resource holons.

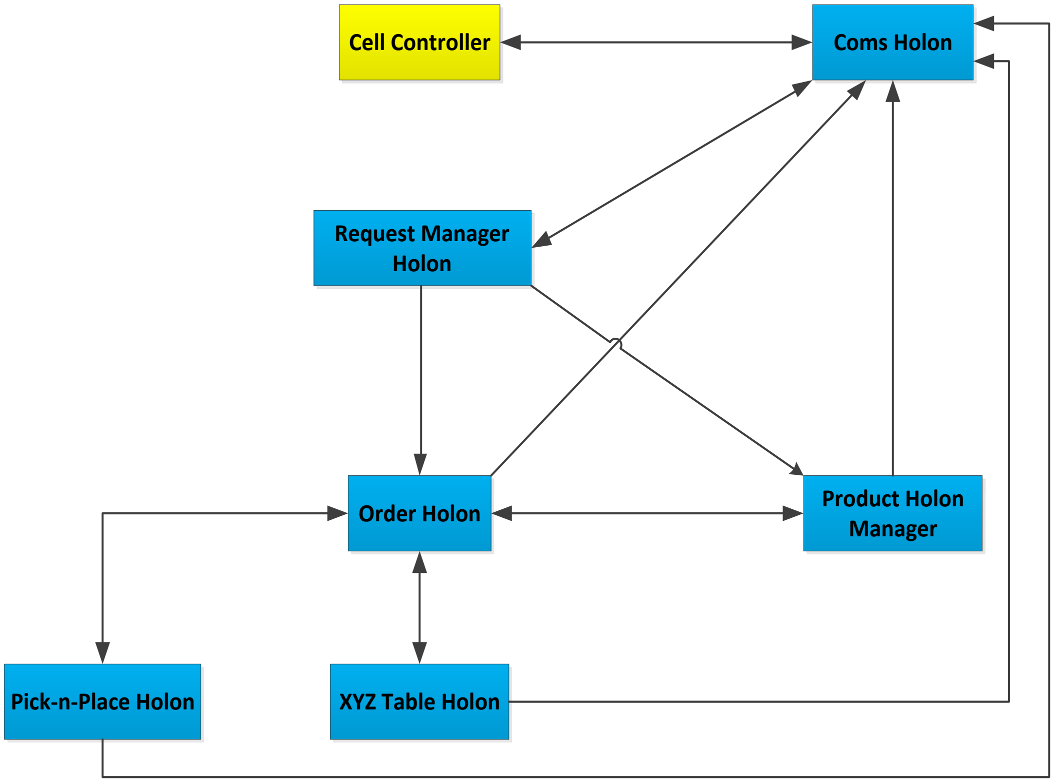


Figure 25: **Control architecture for the Rivet Feeder Station**.

In the control architecture shown in Figure 25, all the product information needed by the rivet feeder subsystem reside in the CC. The CC is responsible for sending messages to the rivet feeder subsystem and also receiving messages from the holons. Examples of messages sent by the CC include:

- Request to perform a task
- Product information
- Enquiries

On the other hand, examples of messages sent to the CC include:

- Confirming receipt of order information
- Confirming receipt of product information
- Confirmation of the size of rivets being fed
- Error messages
- Unhandled message warning

4.2 Inter-holon and intra-holon communication

Inter-holon and intra-holon communication was achieved by means of the methods provided by LabVIEW for transferring data between VIs. The decision of which method to use depends on the continuity of data, availability of a buffer, the number of variables to be passed and the type of data to be shared, as discussed in section 2.3.4.

The exchange of information between holons, which can be referred to as inter-holon communication, is implemented using queues and notifiers. When queues are used, the information is communicated as XML strings. The receiving holon parses the information into LabVIEW data. In this controller, notifiers were used to communicate notifications as Boolean values among different holons.

Intra-holon communication is communication within the VIs in each holon. Intra-holon communication is achieved using queues, local variables and by passing data through wires, which are flow paths connecting different function nodes on the block diagram. Functions used in queues include *Obtain Queue*, *Enqueue Element*, *Dequeue Element* and *Release Queue*. The *Obtain Queue* function returns a reference to a queue, *Enqueue Element* adds an element to the back of a queue, *Dequeue Element* removes an element from the front of a queue and returns the element, and the *Release Queue* function releases a reference to a queue. Figure 26 illustrates both intra-holon communication and inter-holon communication.

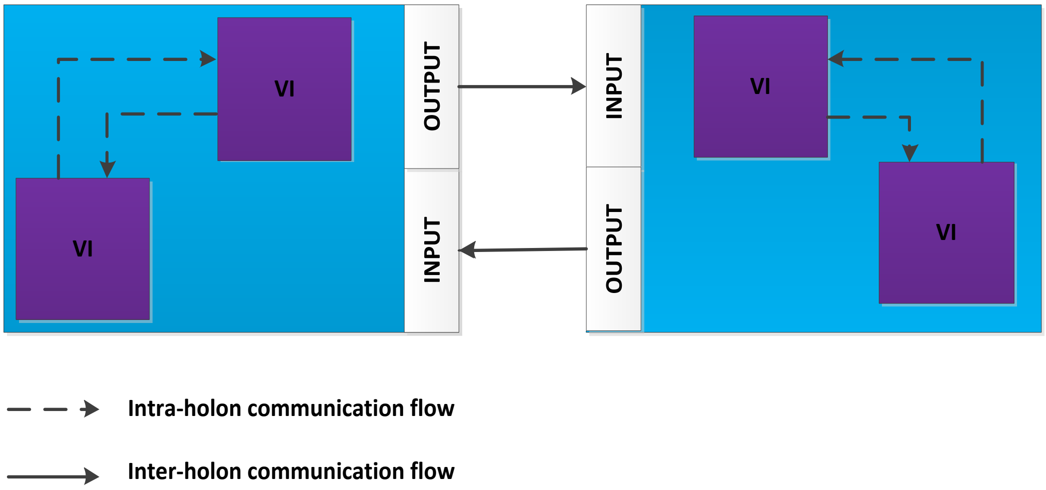

VI
VI
OUTPUT
INPUT
INPUT
OUTPUT
VI
VI
Intra-holon communication flow
Inter-holon communication flow

Figure 26: Inter-holon and intra-holon communication.

The queue names used in the station controller are given in Table 5.

Table 5: Inter-holon queue names

S/N	QUEUE NAME	SOURCE HOLON	DESTINATION HOLON
1	ProductInfo_from_RMH_to_PHM	RMH	PHM
2	Type_from_CC_to_RMH	Coms Holon	RMH
3	OrderInfo_from_RMH_to_OH	RMH	OH
4	ProductInfoReq_from_OH_to_PHM	OH	PHM
5	Information_to_CC	RMH, OH, PHM	Coms Holon
6	ProductInfo_from_PHM_to_OH	PHM	OH
7	ProductInfo_from_RMH_to_PHM	RMH	PHM
8	Information_from_CC_to_RMH	Coms Holon	RMH
9	Coordinates_from_OH_to_XYZ	OH	XYZ Table Holon
10	Completion_from_XYZ_to_OH	XYZ Table Holon	OH
11	Completion_from_Pick-n-Place_to_OH	Pick-n-Place Holon	OH

The choice for using queues was motivated by the fact that queues buffer and store all information written to them until it is read. Because of their buffering capability, queues ensure lossless transferring of information between holons. Furthermore, queues were used because they allow asynchronous communication.

This entails that a holon can receive information from other holons and keep it in a queue whilst communicating with another holon. Furthermore, asynchronous communication is necessary so that the holon that sent the information does not sit idle, waiting for feedback before executing another task

In this controller, only one queue could have been used for each holon. However, multiple queues were implemented considering the advantage that no message identifiers would be needed to check the received message in order to tell which holon sent it. Furthermore, multiple queues ensure that the holon gets the information it needs from another holon without having to wait for the message in a queue with several other messages, since it cannot randomly access the required message from a queue.

4.3 Station controller holons

4.3.1 Holon internal architecture

In order to implement intelligent decision-making algorithms and to allow for communication between holons, both the producer/consumer (see section 2.5.5.4) and the state machine architecture (see section 2.3.5.1) designs were used to develop holons in LabVIEW. In other words, holons were developed as state machines and communications between the state machines is made possible by implementing the producer/consumer architecture. With the producer/consumer architecture, one holon can send (produce) data to other holons (consumers).

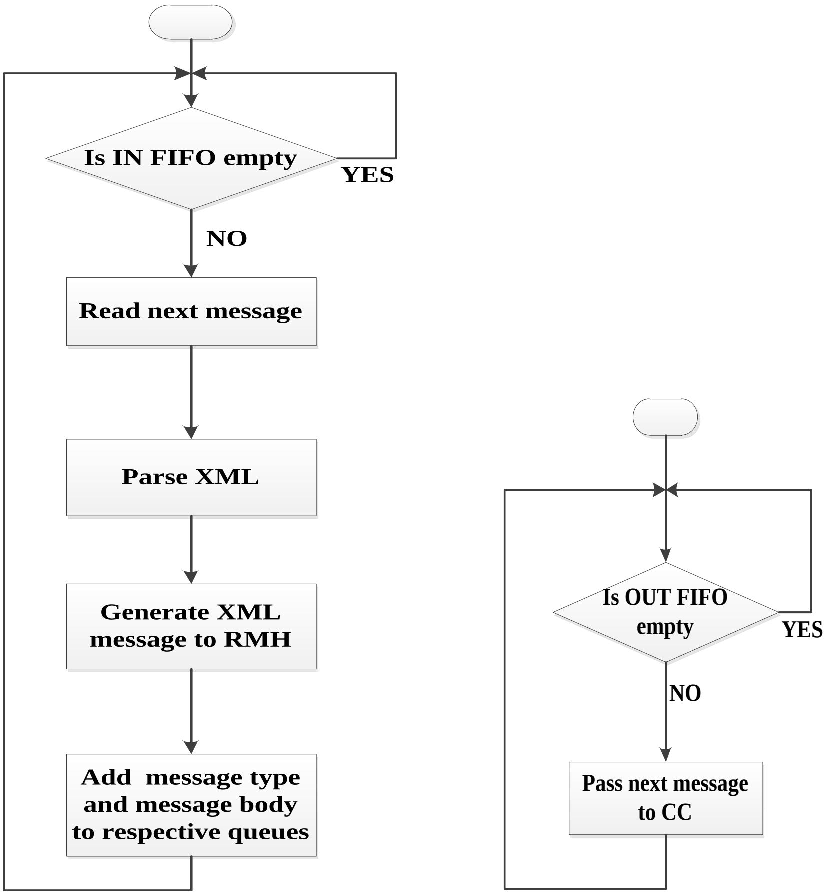
The state machines developed each consists of a case structure inside a while loop with shift registers. A case structure is used to allow the use of different states represented by each of its cases. The shift registers are used to pass data from one state to the next state. On the other hand, the while loop repeats the code within its subdiagram until a stop condition occurs. In addition, the while loop calls the case selector to execute the appropriate state.

The case selector in the state machine was used to control the order of state execution. Three options available for use as case selectors include an integer, a string and an enumerated constant. The enumerated constant was selected because it can be associated with a type definition to allow for additional states to be created when needed and also to allow reuse of the case selector in each of the cases in the case structure for making state transitions. Furthermore, with enumerated constants, descriptive names can be given to the states.

The following sections discuss each holon in the LabVIEW-based station controller. The discussion includes the functions of each holon in relation to the other holons, as well as the cell controller. The LabVIEW VIs used in developing the holons are given in Appendix G. Furthermore, the block diagrams of the station holons for some selected states are given in Appendix F.

4.3.2 Coms Holon

The Coms Holon is a staff holon, designed to handle both *IN FIFO* messages as well as *OUT FIFO* messages. *IN FIFO* messages refers to the information received by the rivet feeder subsystem from the CC, while the *OUT FIFO* messages entail the information sent from the rivet feeder subsystems to the CC. Therefore, the Coms Holon functions as a communication interface between the CC and the rivet feeder subsystem. The functioning of the Coms Holon is illustrated by means of flow charts shown in Figure 27.



(a) IN FIFO messages (b) OUT FIFO messages

Figure 27: Coms Holon: flow-charts for (a) IN FIFO messages and (b) OUT FIFO messages

In order to achieve the desired functionality of the Coms Holon as described above, the buffer of the TCP/IP socket is considered to be the *IN FIFO* queue, while the *OUT FIFO* queue was implemented using the queue operations found on the data communication palettes in LabVIEW. The other holons place their messages for the CC in the *OUT FIFO* queue.

The communication mechanism between the station controller and the CC is through TCP/IP with messages exchanged using XML strings. Therefore, the *TCP Read* VI is used to obtain information after reading a maximum of four bytes of data from a TCP/IP connection with the CC. Furthermore, the Coms Holon also uses the *TCP Write* VI to write data from the rivet feeder station controller to a TCP/IP connection with the CC. The format of XML messaging is discussed in Appendix C.1. Furthermore, the XML format was chosen during the cell architecture design, for the following reasons:

- The XML format allows exchange of information across different development platforms, e.g. LabVIEW and C#.
- The XML format provides flexibility by allowing one to create your own tags and as many as you require.

To be able to generate or parse XML strings for communication with the CC, the “Easy Generate XML_JKI EasyXML.vi” and “Easy Parse XML_JKI Easy XML.vi.” from the *JKI* EasyXML palettes were used. These VIs are given in Appendix E. LabVIEW’s built-in XML schema was considered, but it cannot generate or parse XML schemas defined by others.

“Easy Generate XML_JKI EasyXML.vi” is used to convert LabVIEW data to an XML string with the LabVIEW data names or labels converted to XML item names and the LabVIEW data values converted to XML item values. On the other hand, “Easy Parse XML_JKI Easy XML.vi” is used to convert an XML string to LabVIEW data, based on the string LabVIEW data type. In order to convert the output to the desired data type, the *variant to data* function is used.

The Coms Holon consists of two while loops, namely a reader and a writer. *IN FIFO* queues are handled in the reader while loop, while the *OUT FIFO* queues are handled in the writer while loop. Figure 69 and Figure 70 in Appendix F show the TCP *IN FIFO* and TCP *OUT FIFO* of the Coms Holon, respectively.

When a TCP/IP connection has been established between the rivet feeder station controller and the CC, the Coms Holon checks if the *IN FIFO* queue is empty or not. If the *IN FIFO* queue is not empty the Coms Holon converts the message from the externally used XML format to that used internally. The information is then passed to the Request Manager Holon. In a separate while loop, the *OUT FIFO* queue is monitored. If the *OUT FIFO* queue is not empty, the Coms Holon passes the next message in the queue to the CC.

The Coms Holon is also responsible for stopping the station controller when an error occurs. The station controller comprises multiple parallel while loops, each implementing a holon. Each while loop has a conditional terminal which determines its execution by means of a specified Boolean value. In the controller, all the while loops stop executing when the conditional terminal has a *true* Boolean value.

To ensure a successful station shut down, the use of *Notifier operations*, found under the Synchronization palette in LabVIEW, is implemented as shown in Figure 28.

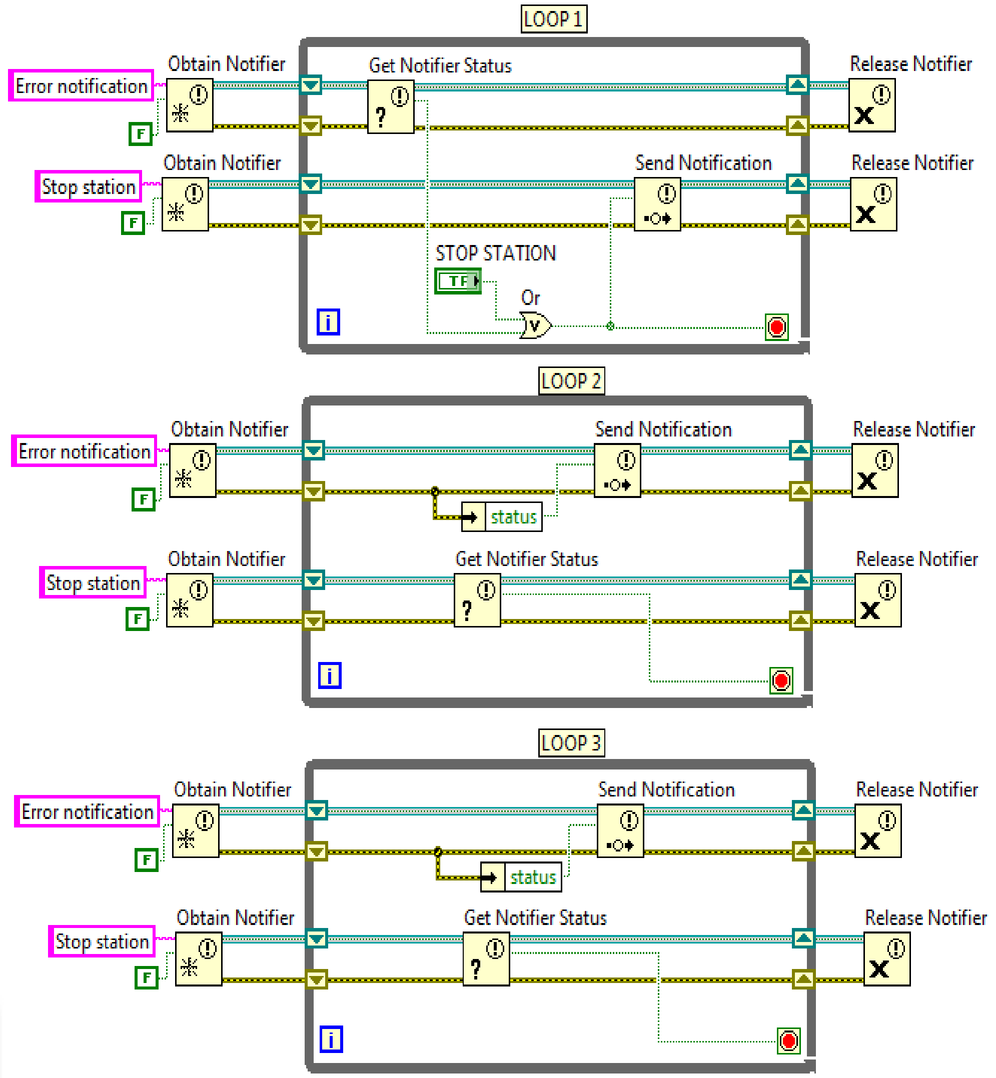


Figure 28: Use of notifiers to stop rivet feeder station controller.

In Figure 28, Loop 1 represents the Coms Holon, while Loop 2 and Loop 3 represent other station holons. When an error occurs in a particular holon (Loop 2 or Loop 3), the error message is sent to the Coms Holon via the *Send Notification* VI. The Coms Holon receives the error notification via the *Get Notifier Status* VI.

Furthermore, the Coms Holon uses the *Send Notification* VI to send a *true* Boolean value to the conditional terminals of all holons.

As can be noted from Loop 1 of Figure 28, the station controller can also be stopped by means of a stop button. This is achieved by the use of a Boolean operator, *or*, which allows a *true* Boolean value to be fed to the conditional terminal from either a stop button or from the *Get Notifier Status* VI.

4.3.3 Request Manager Holon

Similar to the Coms Holon, the Request Manager Holon is also a staff holon. It is designed to direct all the requests passed into the station by the Coms Holon, while shielding the Coms Holon from any delays incurred when parsing the requests. The functioning of the Request Manager Holon is as illustrated by means of a flow chart in Figure 29.

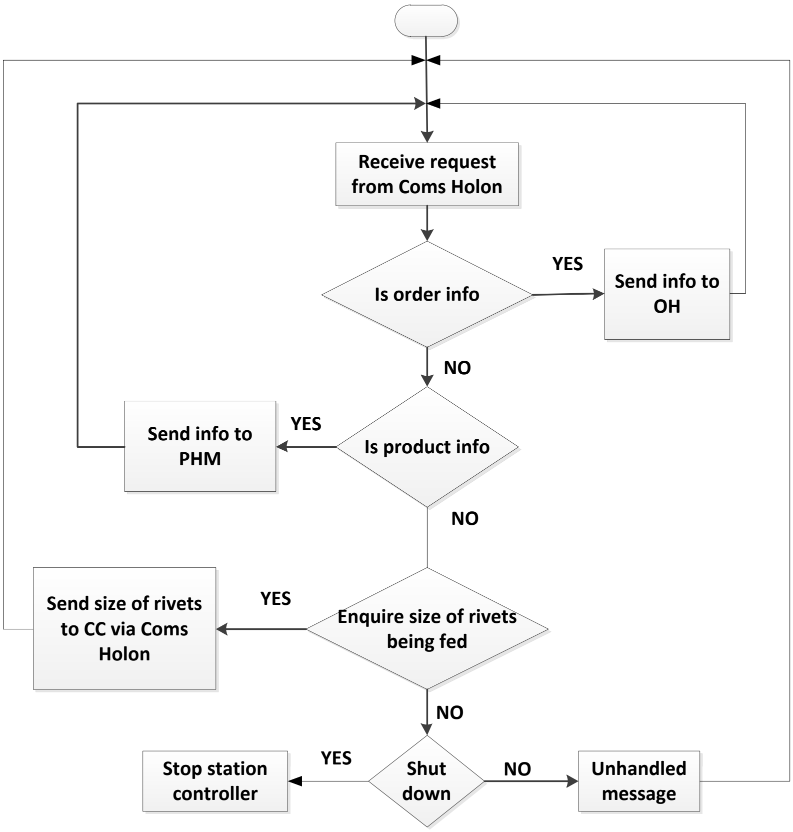


Figure 29: Request Manager Holon flow-chart.

The state machine architecture was implemented for the Request Manager Holon, as shown in Figure 30.

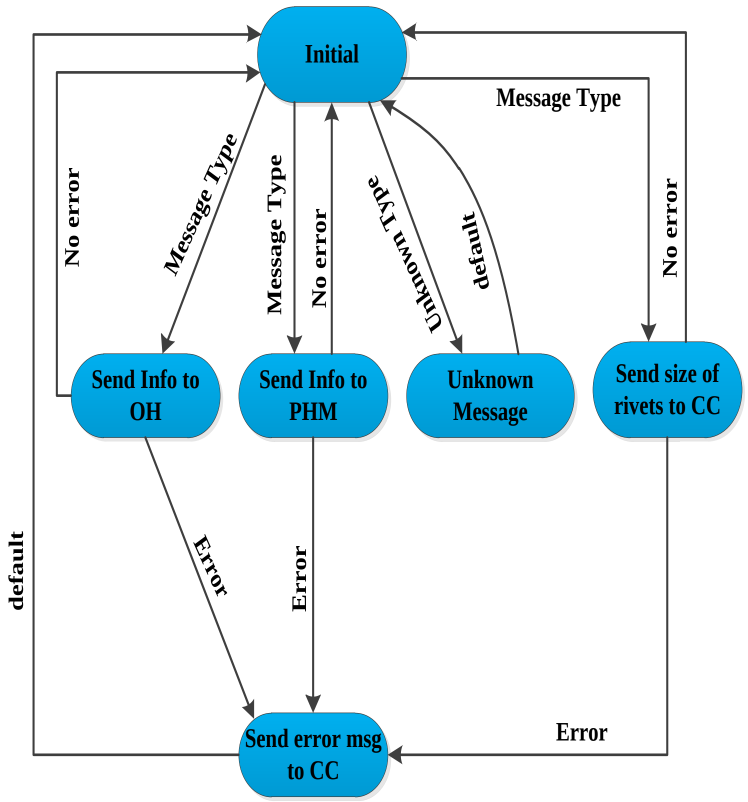


Figure 30: Request Manager Holon state diagram.

Referring to Figure 30, the states used in the state machine architecture are *Initial*, *Send Info to OH*, *Send Info to PHM*, *Send size of rivets to CC*, *Unknown Message* and *Send error msg to CC*. Each state accomplishes a specific function in the Request Manager Holon.

The Request Manager Holon receives information from the CC in the *Initial* state as shown in Figure 31. The received information is mapped to a specific message type. For example, if product information is received, the message type would be *ProdInfoRes*. Therefore, if the received message type matches any of the known message types in the station, then a transition is made from the *Initial* state to the appropriate state. However, if the received message type does not match the known message types, a transition is made from the *Initial* state to the *Unknown Message* state where a warning message is sent to the CC.

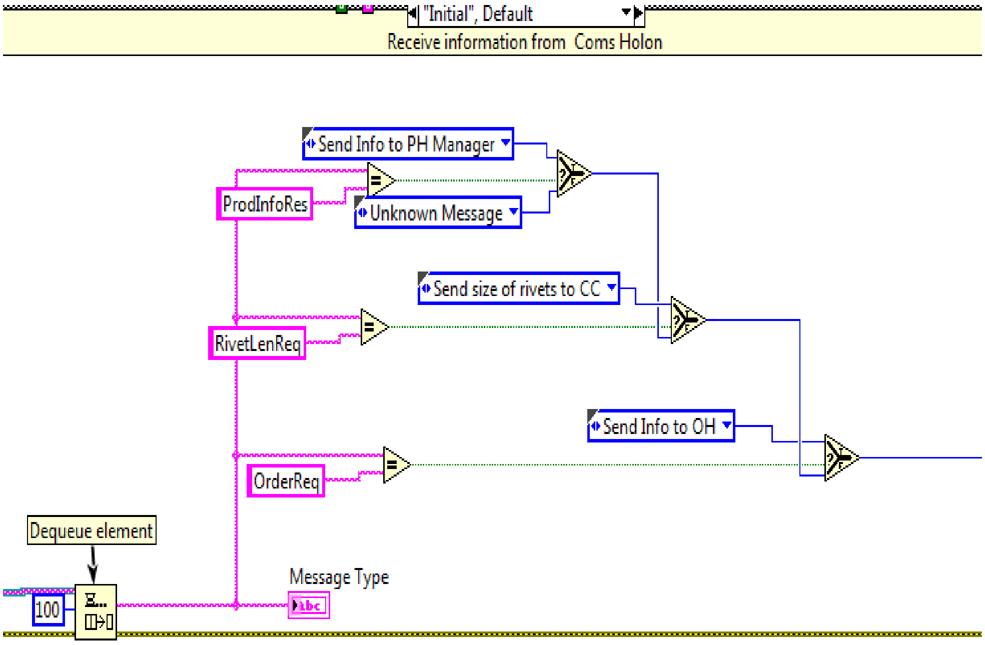


Figure 31: Request Manager Holon's *Initial* state

In the *Send Info to PHM* state, product information is extracted from the "Information _from_CC_to_RMH" queue. The product information is then sent to the Product Holon Manager by adding it to the "ProductInfo_from_RMH _to_PHM" queue as shown in Figure 32. Furthermore, from the *Send Info to PHM* state, a transition can be made either to the *Send error msg to CC* state if an error has occurred, or to the *Initial* state, if no error has occurred.

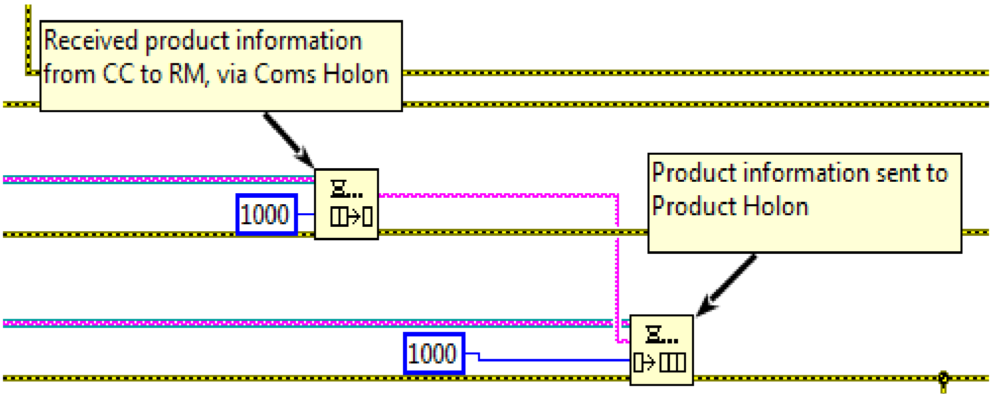

Received product information from CC to RM, via Coms Holon
1000
Product information sent to Product Holon
1000

Figure 32: Request Manager Holon-adding product information to the queue.

When the Request Manager Holon is in the *Send size of rivets to CC* state, it responds to the enquiry made by the CC about the length of rivets being fed. The Request Manager Holon responds by sending the length of rivets which can either be 26 mm or 51.6 mm, depending on the Boolean status read from the optical sensor used to monitor the rivet length. If the Boolean value is true, the implication is that the length of rivets is 51.6 mm. Alternatively, if the Boolean value is *false*, the length of rivets is 26 mm.

Similar to the *Send Info to PHM* state, in the *Send Info to OH* state, the received Order information is extracted from the queue with the reference name of “Information _from_CC_to_RMH”. The information is then added to the queue with a reference name of “OrderInfo_from_RMH_to_OH”. Furthermore, from the *Send Info to OH* state, a transition is made either to the *Send error msg to CC* state if there is an error has occurred in the *Send Info to OH* state, or to the *Initial* state if no error has occurred.

In the *Send error msg to CC* state, all error messages from all the other states are received. These error messages are communicated to the Coms Holon via a notifier VI with a reference name “Error notification”. Therefore, the error messages communicated to the Coms Holon trigger stoppage of the station controller. In addition, the error messages received in the *Send error msg to CC* state are displayed to the user by means of an error cluster output. The error cluster output shows the error status (either *true* or *false*), error code and also the source of the error.

4.3.4 Order Holon

The Order Holon is responsible for coordinating the task of placing rivets on a particular assembly of circuit breakers. This task is performed on request from the CC and can be referred to as an *order*. Therefore, the Order Holon is responsible for handling all the orders coming from the cell controller via the Coms Holon. Furthermore, an order to the Order Holon is created when the Request Manager Holon passes the order information from the Coms Holon to the Order Holon. The order information contains the product type ID of the product (circuit breaker) to be produced.

Figure 33 shows the flow chart, illustrating the functioning of the Order Holon. It must be noted that in this controller, only one Order Holon can be created at a time, instead of multiple Order Holons, as is conventionally done in holonic control. This limitation is on account that only one order can be handled at a time, in the rivet feeder station.

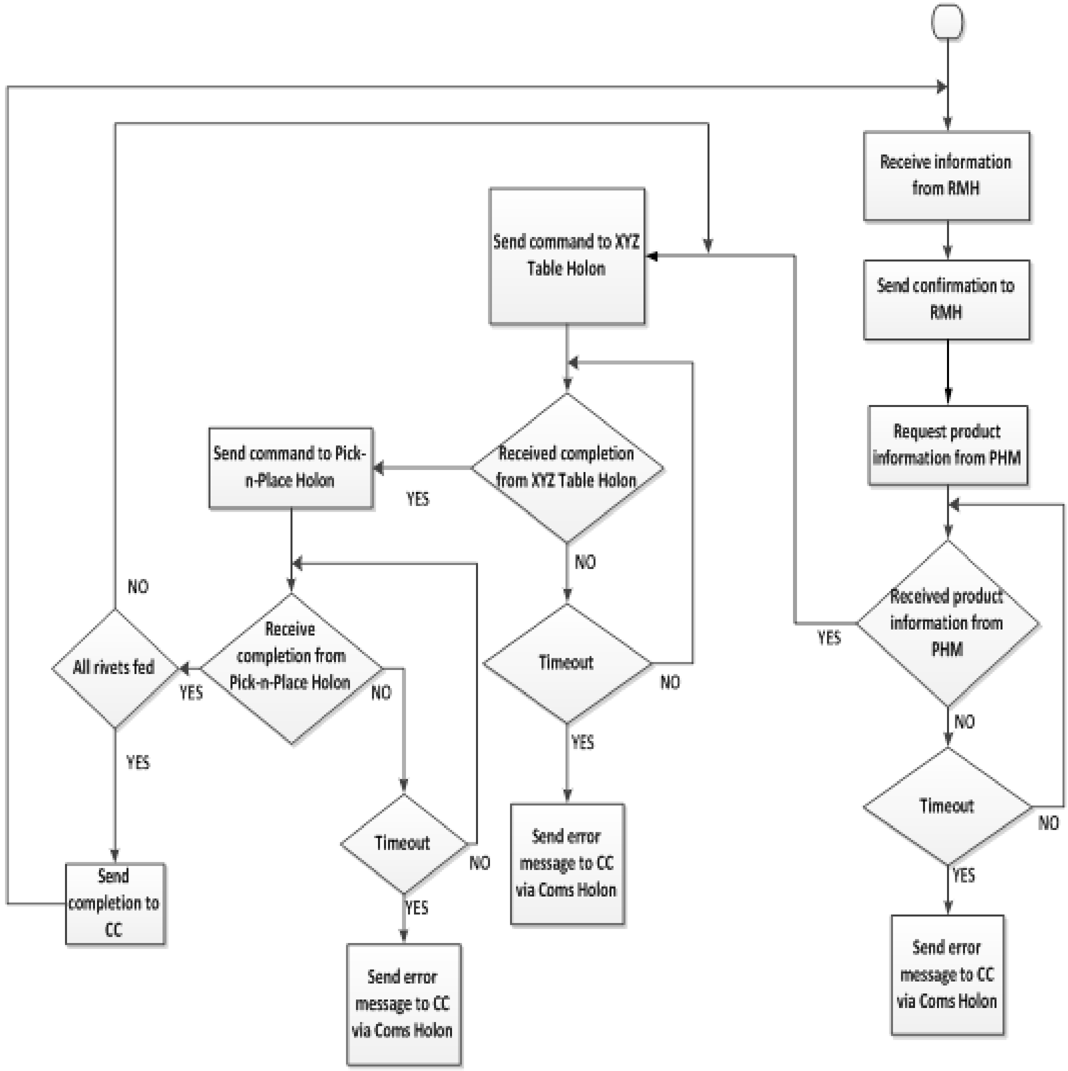

Receive information from RMH
Send confirmation to RMH
Request product information from PHM
Received product information from PHM
YES
NO
Timeout
NO
YES
Send error message to CC via Coms Holon
Send command to XYZ Table Holon
Received completion from XYZ Table Holon
YES
NO
Timeout
NO
YES
Send error message to CC via Coms Holon
Send command to Pick-n-Place Holon
Receive completion from Pick-n-Place Holon
YES
NO
Timeout
NO
YES
Send error message to CC via Coms Holon
All rivets fed
NO
YES
Send completion to CC

Figure 33: **Order Holon flow-chart.**

The order holon was designed based on the state machine architecture. The states are as follows: *Initial*, *Receive Info from RMH*, *Request Info from PHM*, *Receive Info from PHM*, *Send command to XYZ*, *Receive completion from XYZ*, *Send command to Pick-n-Place*, *Receive completion from Pick-n-Place*, *Send completion to CC* and *Send error msg to CC*. These states are shown in Figure 34.

The initial state is the starting state in the Order Holon. A transition is then made from the initial state to the *Receive Info from RMH* state where the Order Holon receives information from the request manager. The received information contains the product type ID of the product to be produced. The Order Holon uses this product type ID to request for product information from the Product Holon Manager in the *Request Info from PHM* state. The product type ID used to make a request is passed from the *Receive Info from RMH* state to the *Request Info from*

PHM state by means of a shift register. Furthermore, from the *Request Info from PHM* state, a transition is made to the *Receive Info from PHM* state.

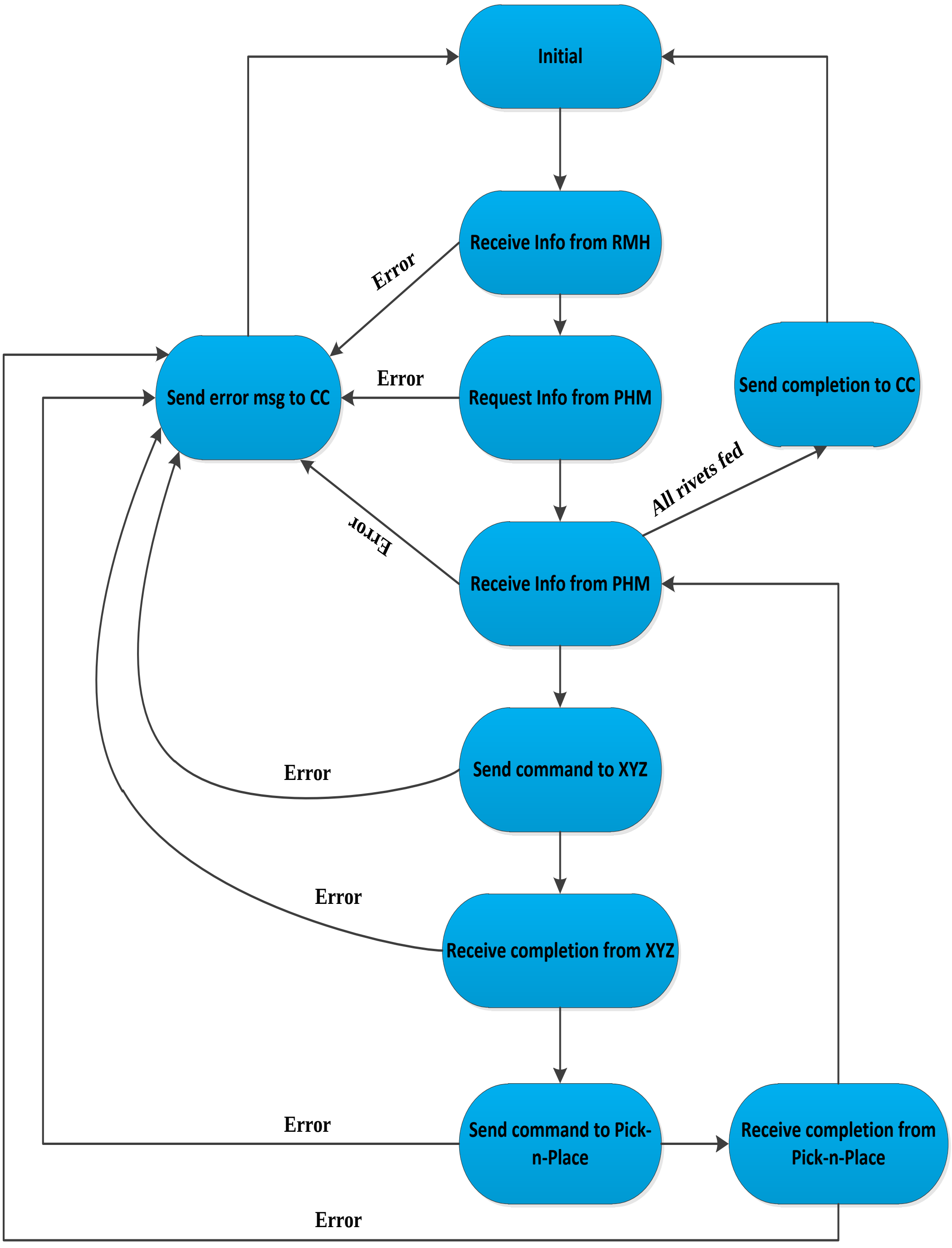


Figure 34: Order Holon state diagram.

In the *Receive Info from PHM* state, the Order Holon receives the product information from the Product Holon Manager if the required information is available. Therefore, in this state, the status of the message queue is checked to find out whether the queue is empty or not. If the queue is not empty it implies that the product information has been received. Furthermore, if the queue is empty a timeout check is conducted. If the timeout is true, an error message is sent to the CC via the Coms Holon. In addition, if the timeout is not true, then the Order Holon waits to receive information from the Request Manager Holon. Figure 35 shows this implementation.

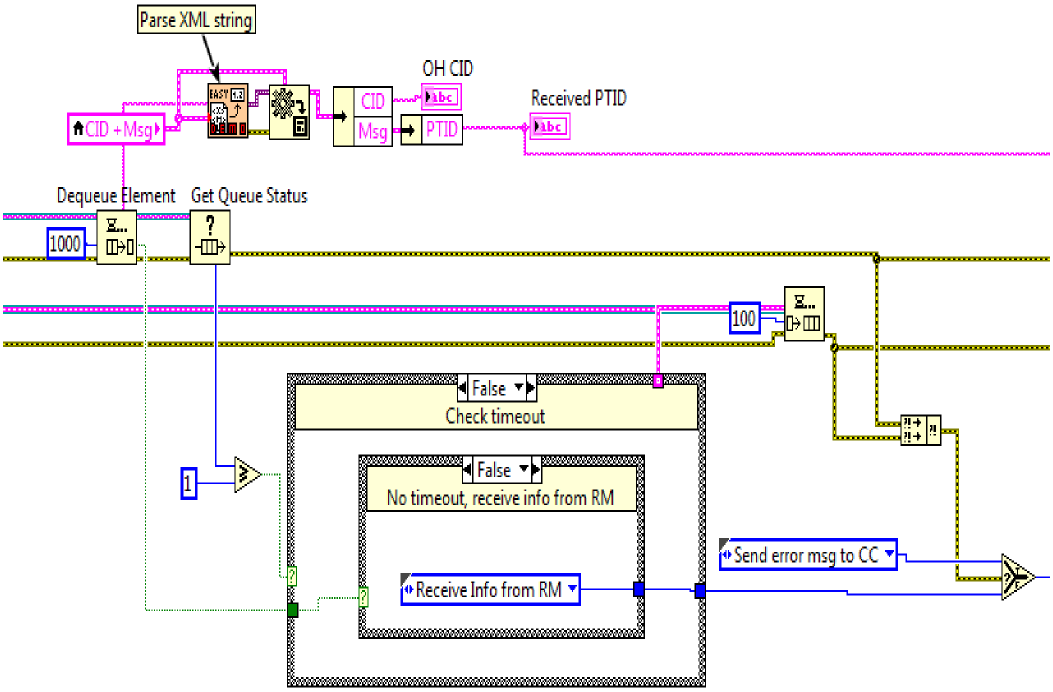


Figure 35: Order Holon: checking timeout if no message is received from RMH.

The presence of a message in the queue can be detected using the *Get Queue Status.vi*. Furthermore, in Figure 35, the Order Holon receives messages from the Request Manager Holon by using the *Dequeue Element.vi*. The *Dequeue Element.vi* has a timeout terminal from which the selector of the case structure can be connected. Therefore, the status from the *Get Queue Status.vi* and the timeout output from the *Dequeue Element.vi* are used to check for timeout. As shown in Figure 35 if there is no timeout, no transition in state is made so that the Order Holon can wait for a message from the Request Manager Holon.

The product information received from the Product Holon Manager is in XML format. Therefore, before commands are sent to the XYZ Table Holon and Pick-n-Place Holon, the product information is parsed into clusters of elements using *JKI*

EasyXml.vi and the *variant to data.vi*. The cluster of elements is then fed to an Index Array function which is used to extract coordinates for placement of each rivet on an assembly of circuit breakers.

The Order Holon sends commands to the XYZ Table Holon in the form of positioning coordinates to position circuit breakers. In addition, the Order Holon receives a completion notification from the XYZ Table Holon once the circuit breakers have been positioned. The Order Holon then sends a command to the Pick-n-Place Holon to perform rivet placement and waits for a completion notification from the Pick-n-Place Holon. The sequence of sending commands to both the XYZ Table Holon and the Pick-n-Place Holon continues until all the rivets have been placed on the circuit breakers. When all the rivets have been fed, the Order Holon sends a completion notification to the CC and waits for another order in the *Initial* state. In the event that the Order Holon does not receive a completion notification from either the XYZ Table Holon or the Pick-n-Place Holon, the Order Holon checks if a time out has occurred. If timeout has occurred the status is true, otherwise it is false if it has not occurred. Thus, if timeout is true, the Order Holon sends an error message to the CC via the Coms Holon, otherwise the Order Holon waits to receive a completion notification.

4.3.5 Product Holon Manager

The Product Holon Manager serves as a repository of product information. The purpose of having the Product Holon Manager in the rivet feeder subsystem is to allow for manual override (i.e. operation without a connection to the CC) and also to avoid excessive communication traffic which would exist if information for the same product type always came from the cell controller. The functioning of the Product Holon Manager in relation to other holons is depicted in Figure 36.

As shown in Figure 36, if the Product Holon Manager does not receive the requested product information within the specified time, a timeout is passed and an error message is sent to the CC via the Coms Holon.

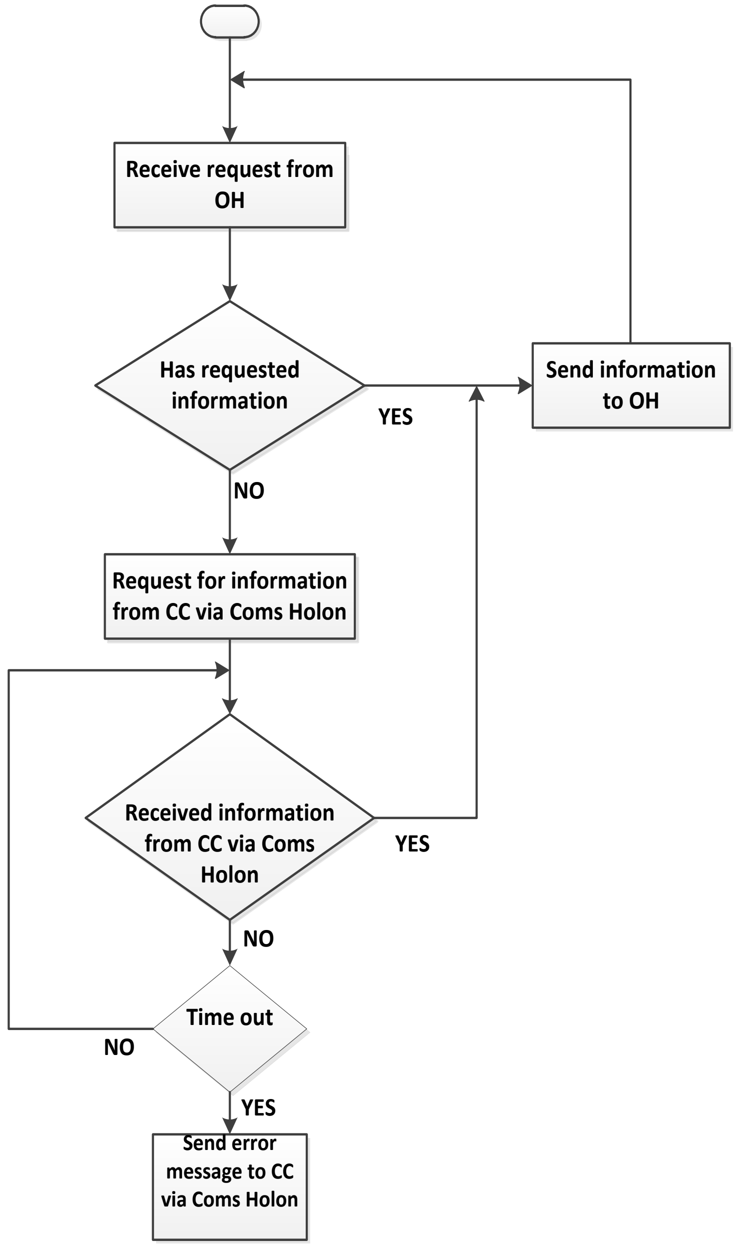

Receive request from OH
Has requested information
YES
NO
Send information to OH
Request for information from CC via Coms Holon
Received information from CC via Coms Holon
YES
NO
Time out
NO
YES
Send error message to CC via Coms Holon

Figure 36: Product Holon Manager flow-chart.

Like other holons, the Product Holon Manager was designed using the state machine architecture. The states for the Product Holon Manager are: the I*nitial*, R*eceive info from RMH*, *Receive request from OH and Send error msg to CC* state. These states are as shown in Figure 37 below.

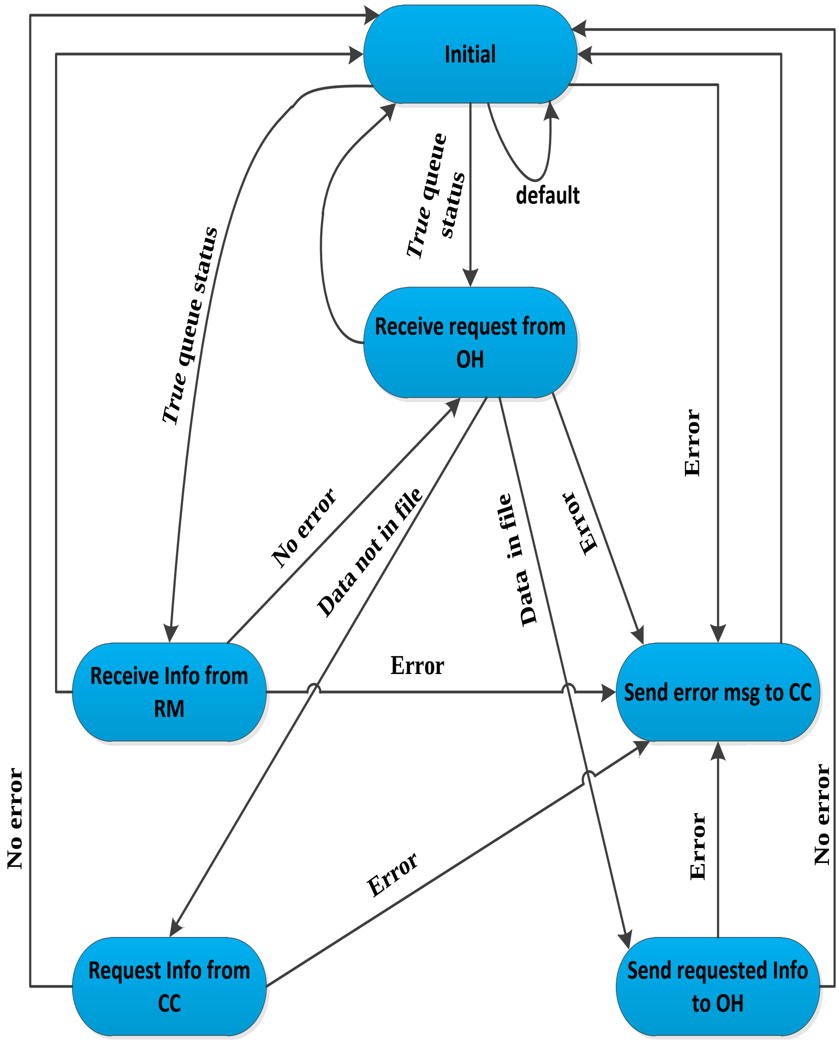


Figure 37: Product Holon Manager state diagram.

In the *Initial state* the Product Holon Manager waits to receive either information from the Request Manager Holon or a request for product information from the Order Holon. Furthermore, the Product Holon Manager has priority to receive information from the Request Manager Holon since the requested information from the Order Holon can only be supplied when the Product Holon Manager has received the information from the Request Manager Holon.

In the R*eceive info from RMH* state, the Product Holon Manager receives product information from the Request Manager Holon as an XML string. The product information includes the product type ID, number of rivets, positioning coordinates, etc. The product information is stored in a file created using the configuration file palettes found under the file I/O palette in LabVIEW. The structure of data storage is similar to a hash table where a value is identified by means of a key. In this case the key used for extracting the information from the file is the product type ID. Therefore, when product information is received from the cell controller, it is stored in a file (in rows) with the corresponding product type ID (key). Product holons can dynamically be created since each line in the table represents a product holon. The structure of the data storage table used is shown in Appendix C.3.

The *Open Config Data.vi* is used to open a reference to the configuration data found in the file. Like the *Close Config Data.vi*, the *Open Config Data.vi* is used in both writing data to the file and also reading data from file. The *Write Key.vi* is used for writing the XML string of data to the file by specifying the product type ID (key) in a specified section of the file. On the other hand, the *Read Key.vi* is used to extract the required product information in XML string format from a specified section of the file by specifying the product type ID (key).

In the R*eceive request from OH* state, the Product Holon Manager first checks the inbox queue status to determine whether a request is available or not. If the queue is not empty, the Product Holon Manager uses the product type ID received from the Order Holon to get the required product information from the file. If the requested information is not available from the file, the request is put back into the queue and the Product Holon requests the required product information from the CC via the Coms Holon. In addition, the product holon also writes the request to a file to avoid the same request to be made to the CC more than once. Thus, before the Product Holon Manager sends the request to the CC, it has to check if the request has been made before.

4.3.6 XYZ Table Holon

The XYZ Table Holon is a resource holon responsible for placing the fixtures of circuit breakers in positions obtained from the cell controller. This holon was also designed based on the state machine architecture. Figure 38 shows the state diagram used to develop the state machine.

As can be seen from Figure 38, the state machine consists of three states namely, the *Initial* state, *Receive and send coordinates* state and *Send error msg* state. Transitions in states are made from the *Initial* to the *Receive and send coordinates* states, then from the *Receive and send coordinates* to the *Send error msg* state when there is an error and finally from the *Send error msg* state back to the *Initial* state.

In the *Receive and send coordinates* state, the XYZ Table Holon receives commands, in the form of positioning coordinates, from the Order Holon and sends them to the motor controllers. Furthermore, when the XYZ Table Holon has received a confirmation of completion from the motor controllers, it sends a confirmation of completion to the Order Holon. If the XYZ Table Holon does not receive a confirmation of completion from the motor controllers, a timeout is passed and the XYZ Table Holon sends an error message, in the *Send error msg* state, to the CC via the Coms Holon. The functioning of the XYZ Table Holon is further illustrated by the flow chart in Figure 39.

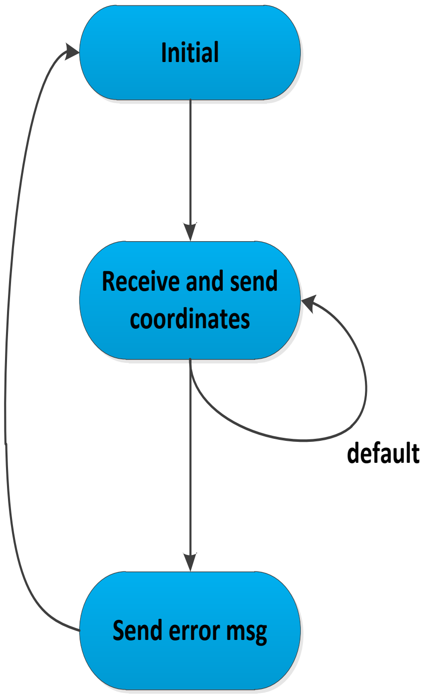


Figure 38: XYZ Table Holon state diagram.

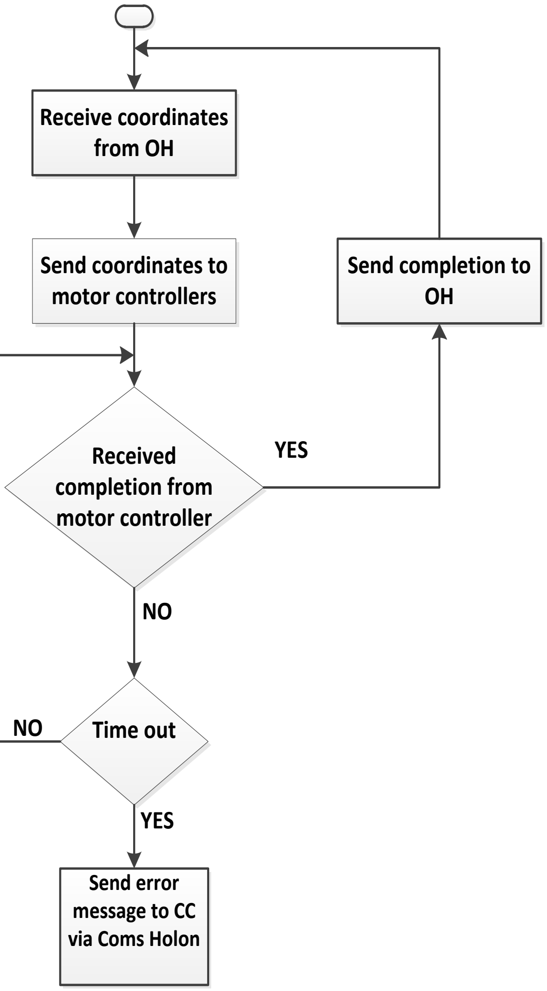

Receive coordinates from OH
Send coordinates to motor controllers
Send completion to OH
Received completion from motor controller
YES
NO
Time out
NO
YES
Send error message to CC via Coms Holon

Figure 39: XYZ Table Holon flow-chart.

4.3.7 Pick-n-Place Holon

The Pick-n-Place Holon is a resource holon responsible for accomplishing two tasks, namely picking rivets from the singulation device and placing rivets into the circuit breaker holes. The state machine architecture was used to design the Pick-n-Place Holon and has three states namely, the *Initial* state, *Receive and send commands* state and *Send error msg* state.

When a transition is made from *Initial* state to the *Receive and send commands* state, the Pick-n-Place Holon receives commands from the Order Holon and then sends digital signals to the pneumatic gripper and cylinder. As a result, the pneumatic gripper closes and the cylinder extends, thereby placing rivets on circuit breakers. The Pick-n-Place Holon then sends a confirmation of completion to the Order Holon after receiving a completion from the proximity sensor. Otherwise a timeout occurs and an error message is sent to the CC via the Coms Holon. Figure 40 shows the state diagram for the Pick-n-Place Holon.

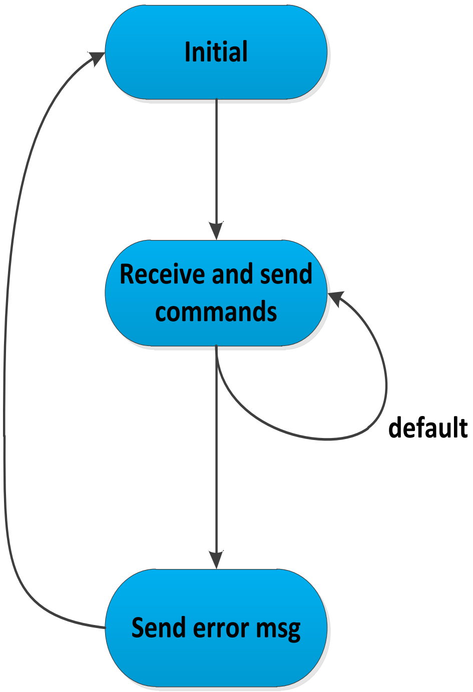


Figure 40: Pick-n-Place state diagram.

It can be seen from Figure 40 that possible transitions are from the *Initial* state to the *Receive and send commands* state, from the *Receive and send commands* state to the *Send error msg* state and finally from the *Send error msg* state to the *Initial* state. Furthermore, in the *Receive and send commands* state, if no error occurs then no transition is made. Therefore, the *Receive and send commands* state is a default state if there is no error. The flow chart shown in Figure 41 illustrates the functions of the Pick-n-Place Holon.

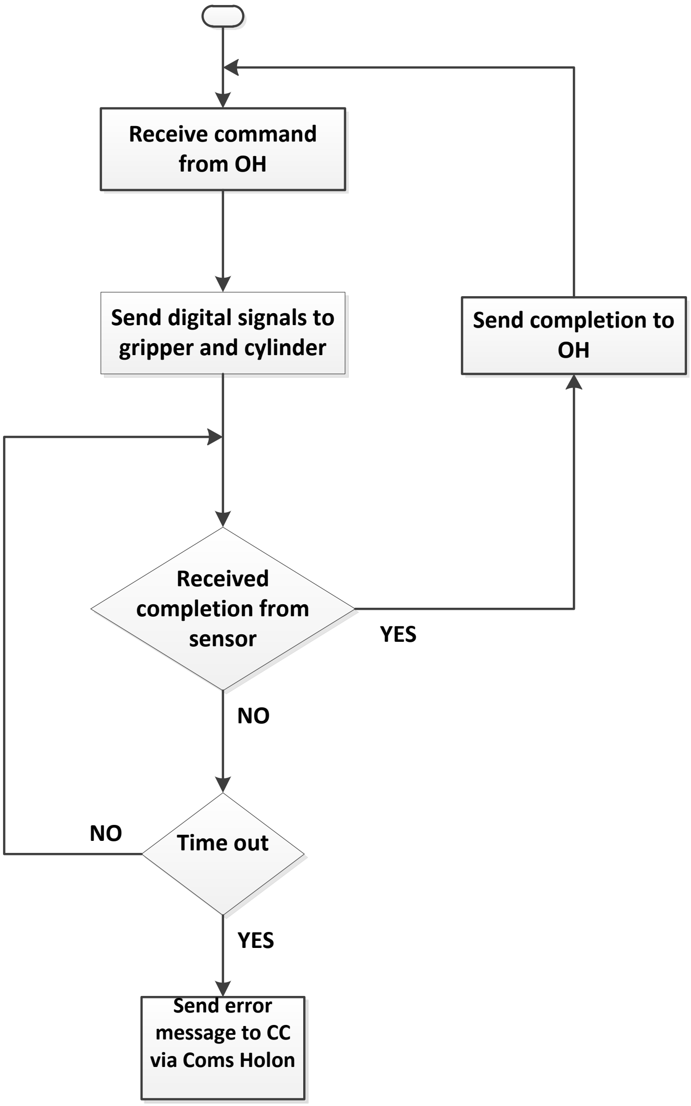

Receive command from OH
Send digital signals to gripper and cylinder
Send completion to OH
Received completion from sensor
YES
NO
NO
Time out
YES
Send error message to CC via Coms Holon

Figure 41: Pick-n-Place Holon flow-chart.

4.4 Hardware considerations

The hardware considerations were based on the functional requirements of the rivet feeder station. Therefore, the NI compactRIO-9068 (cRIO-9068), 8-slot integrated controller and chassis system was considered as a means of providing interface between the LabVIEW based controller and the rivet feeder station resources, which are the Pick-n-Place mechanism and the XYZ positioning table. The other hardware components considered are the solenoid valves, proximity and optical sensors. To achieve the aforementioned interfaces, the cRIO-9068 was integrated with an NI 9375 module and an NI 9881 module as shown in Figure 42.

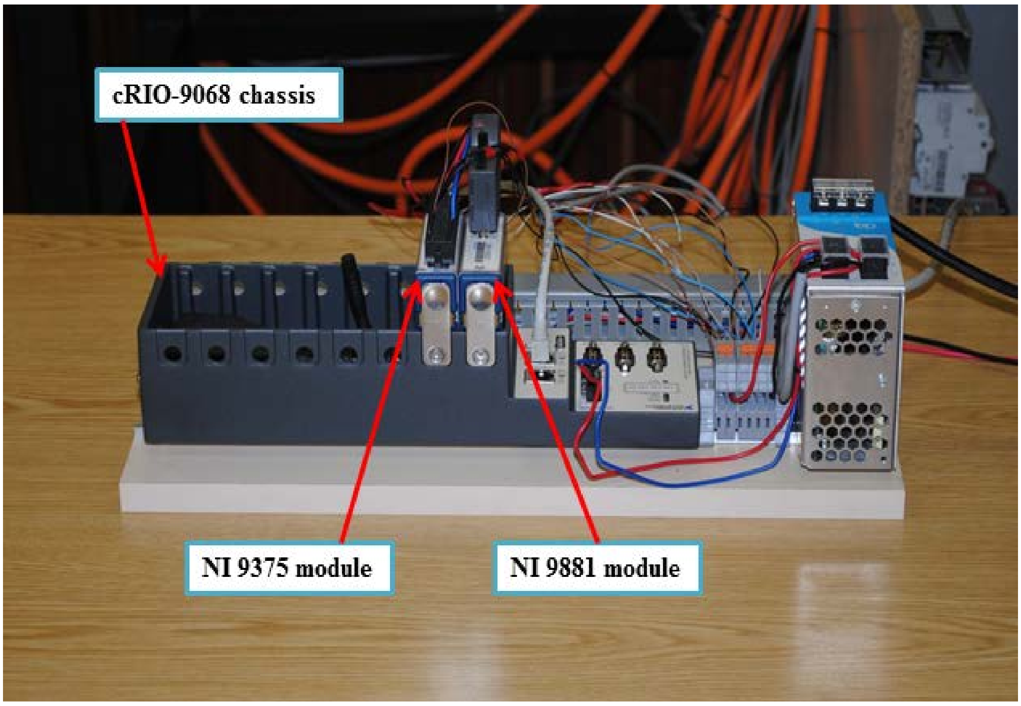


Figure 42: Setup of CompactRIO-9068 chassis with NI 9375 and NI 9881 modules.

The NI 9375 is a 32-channel module which provides connections for a bank of 16 digital input (DI) channels and a bank of 16 digital output (DO) channels. Each channel in the DI bank and DO bank has a pin to which connection to a device can be made. Furthermore, the DI bank has a DI COM as the common terminal and the DO bank has a DO COM as the common terminal. In this controller, the NI 9375 module was used to provide digital signals to the actuators.

The NI 9881, 1-port C series CANopen interface module has one 9-pin male D-Sub connector that provides connections to a CAN bus. In addition, the module also has pins for CAN_H and CAN_L to which CAN bus signals are connected. The NI 9881 module was used to provide communication between the LabVIEW based controller and the Festo servo drives of the XYZ positioning table.

4.5 Implementation

4.5.1 Sensors

The proximity and optical sensors shown in Figure 43 were connected to the digital inputs of the NI 9375 module. The sensors were used for monitoring the status of actuators. For example, a proximity sensor was installed on the DHPS-20-A, parallel pneumatic gripper to detect the closing and opening of the gripper jaws.

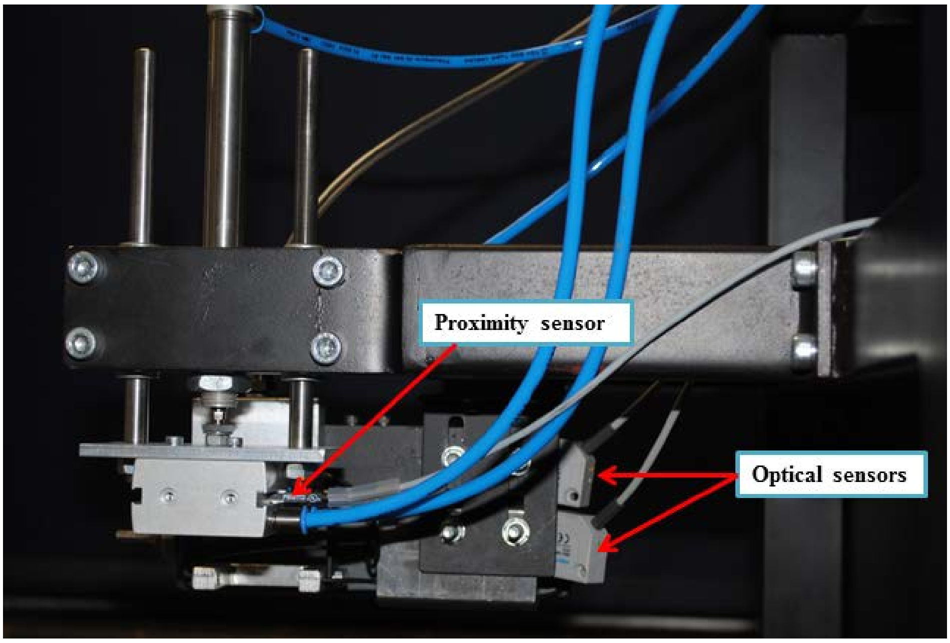


Figure 43: Proximity and optical sensors

As can be seen from Figure 43, two optical sensors were installed on the singulation device to detect the length of rivets fed and also the availability of rivets. Therefore, one optical sensor was installed underneath the singulation device, to check the size of rivets being fed into circuit breakers, by detecting the presence of a rivet from the singulation device. The second optical sensor was installed inside the singulation device, to detect the availability of rivets. Since two sizes of rivets are considered in this thesis, the detection of a rivet from the singulation device entailed the size of the longer rivets. However, when the rivet was not detected, it entailed the shorter size of rivets.

4.5.2 Actuators

The actuators that were controlled via the cRIO-9068 integrated and chassis system include the DHPS-20-A, parallel pneumatic gripper, the DSNU-16-100-P-A air cylinder and the Festo servo drives. As discussed in section 4.4, the Festo servo drives were controlled over NI CANopen interface in order to send commands to the X-axis, Y-axis and Z-axis of the positioning table.

The DHPS-20-A, parallel pneumatic gripper and the DSNU-16-100-P-A, air cylinder, of the *Pick-n-Place mechanism* were controlled by using two 5/2-way single solenoid valves. The valves were connected to the normally closed (NC) terminals of a set of relays. The solenoid valves were open in the initial state. The relays were energized from a 24 V DC supply. In the initial state of each solenoid valve, the gripper jaws were open and the air cylinder retracted. Thus, energizing the relays caused the gripper jaws to close and the air cylinder to extend. Furthermore, the signals from the solenoid valves were read from the digital inputs of the NI 9375 module.

In order to control the Festo servo drives, a physical connection between the NI 9881, 1-port CANopen module and the Festo servo motor controllers was required. This was accomplished by using four 9-pin D-sub connectors and Beckhoff ZB5100 CAN-bus, 2x2x0.25 mm^2, 4-core cable. The four 9-pin D-sub connectors were integrated with 120 Ω termination resistors since the CAN-bus is bidirectional.

4.6 Interfacing the controller with sensors and actuators

The LabVIEW based controller was interfaced with the sensors and actuators of the rivet feeder station via the cRIO-9068 discussed in section 4.4. The communication between the controller and sensors was achieved through reading from the I/O variables which were dragged and dropped inside the holons. Furthermore, the actuators for the pick-n-place mechanism were interfaced with the controller by means of solenoid valves. These valves were regulated through writing to the I/O variables which were also dragged and dropped inside the holons. Furthermore, the interface with the Festo servo drives was provided through the NI CANopen module.

The Festo servo drives are controlled by custom LabVIEW VIs (the Cartesian robot program) developed by Mr Reynaldo Rodriguez, the laboratory technician of the research group. The Cartesian robot program was used to set position parameters on the Festo servo drives over NI CANopen interface. Furthermore, the Cartesian robot program receives commands from the LabVIEW based controller which runs on the PC.

The LabVIEW based controller and the Cartesian robot program are interfaced through the use of network-published variables. As their name suggests, these variables enable the transfer of data over a network. In order to control the Cartesian robot, positioning coordinates are written to the network-published shared variable nodes on the LabVIEW based controller and read by the Cartesian robot program. Figure 44 shows how the positioning coordinates are sent to the motor controllers by writing to the network-published variables. The coordinates are sent by the XYZ Table Holon after receiving them from the Order Holon.

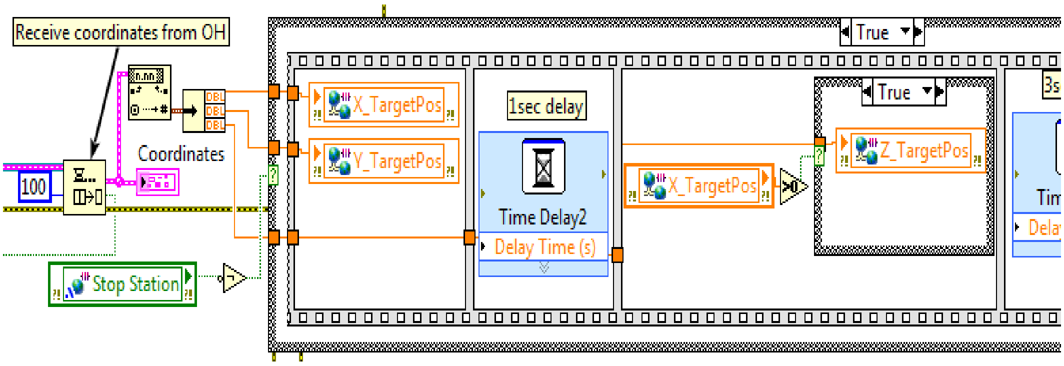
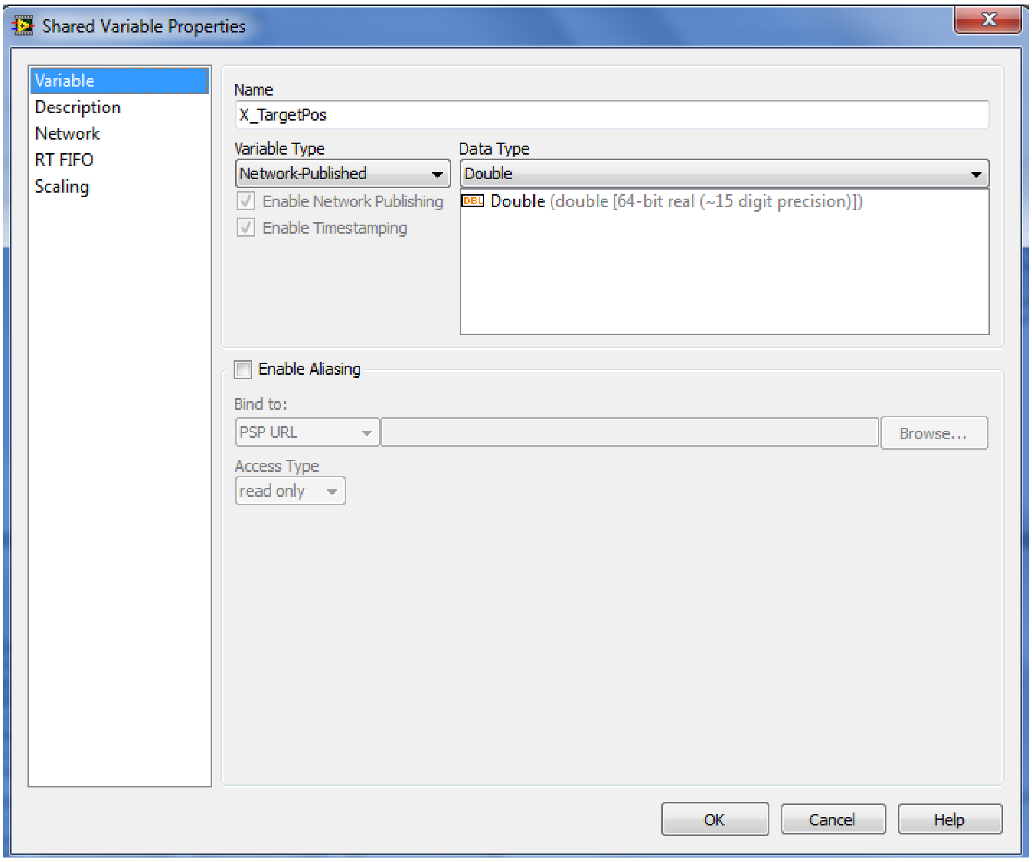


Figure 44: Sending coordinates to the motor controller.

The network-published variables were created and configured using the system manager as shown in Figure 45. The system manager allows creating variables of the desired data type. For example, in Figure 45, the variable *X_TargetPos* was created and assigned the *Double* data type. Furthermore, to allow for communication over a network with the network-published variables, windows firewall had to be turned off.


Shared Variable Properties

- Variable
- Description
- Network
- RT FIFO
- Scaling

Name: X_TargetPos

Variable Type: Network-Published

Data Type: Double

☑ Enable Network Publishing

☑ Enable Timestamping

Double (double [64-bit real (~15 digit precision)])

☐ Enable Aliasing

Bind to: PSP URL

Access Type: read only

Figure 45: Using the system manager to configure shared variables.

4.7 Accessing parameters of the motor controllers via CANopen

Each motor controller receiving the positioning coordinates has a specific node number. In addition, all three motor controllers are set to the same baud rate. The Festo Configuration Tool (FCT) was used to set the baud rate and node number for each motor controller. Each motor controller was set to baud rate of 1Mbits/s. Furthermore, motor controllers for the X-axis, Y-axis and Z-axis were assigned node-IDs of one, two and three, respectively.

The NI CANopen interface for the CompactRIO is supported by the Industrial Communications for CANopen driver, which features support for SDOs, PDOs, NMT, heartbeats, node guarding and synchronization. Furthermore, CANopen provides a simple and standardised possibility to access the parameters of the motor controllers. This is achieved by assigning a unique number (index and subindex) to each parameter (CAN object).

There are two methods for accessing parameters of the motor controllers via the CAN bus, namely the Service Data Objects (SDOs) method in which the motor controller acknowledges each parameter access and the Process Data Objects (PDOs) method in which there is no acknowledgement from the motor controller. As a rule, the motor controller is parameterised and also controlled via the SDO access.

The PDOs are used for the fast exchange of process data (e.g actual speed). FESTO (2012) gives a total of four transmit PDOs (TPDOs) and four receive PDOs (RPDOs) for the CMMP series motor controllers. FESTO (2012) also makes a distinction between the TPDOs and the RPDOs: in TPDO the motor controller sends a PDO when a certain event occurs, on the controller side, while in RPDO, the motor controller evaluates a PDO when a certain event occurs, on the host side. Furthermore, RPDOs contain data to be written to the device’s output while the TPDOs contain data to be read from the device’s input.

The two types of PDOs have CANopen bus identifiers (COB-IDs) in hexadecimal format, also referred to as identifiers. The COB-IDs used for the four transmit PDOs are calculated by adding node IDs to the base IDs as follows: TPDO1=180h+NodeID, TPDO2=280h+NodeID, TPDO3=380h+NodeID and TPDO4=480h+NodeID. Similarly, the four receive PDOs have COB-IDs determined as follows: RPDO1=200h+NodeID, RPDO2=300h+NodeID, RPDO3=400h+NodeID and RPDO4=500h+NodeID. The use of COB-IDs is to ensure that each motor controller receives a command only meant for it.

Commands are transmitted to all the motor controllers by means of a network management (NMT) VI, shown in Figure 46.

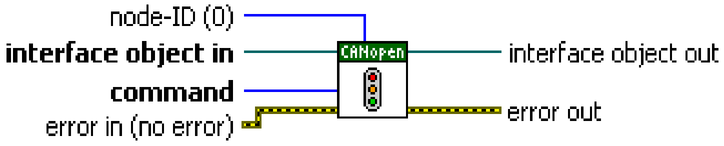


Figure 46: Network management VI.

NI LabVIEW (2013) gives details of the network management VI. The command transmitted by this VI controls the NMT states of the devices on the network. The *node-ID* is the input terminal that specifies the node-ID of a target device to which the command is sent. In this application, the node-ID was set to zero so that all the nodes on the network are controlled simultaneously. The *interface object in* specifies the reference number of an interface object. This object is an interface to the CANopen network. Furthermore, the command terminal specifies the command that the VI sends. In this application, the reset communication command was specified, which switches the devices into the reset communication state to initialise the CANopen object.

Every message sent on the CAN bus contains a type of address which is used to determine the bus participant for which the message is meant, since there are three motor controllers. Before a message can even be sent on the CAN bus, a CANopen interface has to be created in order for the cRIO-9068 to access the CAN bus. The VI shown in Figure 47 is used for creating a CANopen interface.

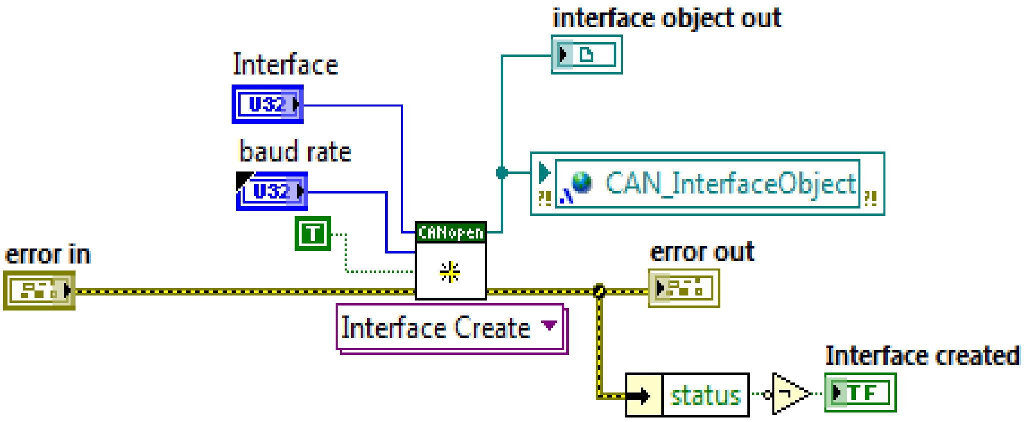


Figure 47: CANopen Interface Create.vi.

As soon as the CANopen interface is created, the motor controllers are set to the homing mode through the modes_of_operation object (6060_h). Thus, the object (6060_h) sets the operating mode of the motor controller. The other operating modes are: positioning mode, velocity mode, torque mode and interpolated position mode. Figure 48 shows a VI that sets the operating modes of the Festo motor controllers. This VI takes in an SDO object, created by a calling VI, as an input.

When the homing mode has been set, homing of the three axes takes place, with the Z-axes being the first to start homing before the X and Y axes. When the three axes have reached the home positions, the CAN object *target_position* (607A_h) is used to pass product information in the form of positioning coordinates to the motor controllers. The positions arrived at by the three axes is given by the CAN object *position_actual_value* (6064_h). Thus, in the CANopen protocol, the entire regulation of the motor controller is achieved via two objects: the host can regulate the motor controller via the *controlword*, while the status of the motor controller can be read back in the object *statusword*.

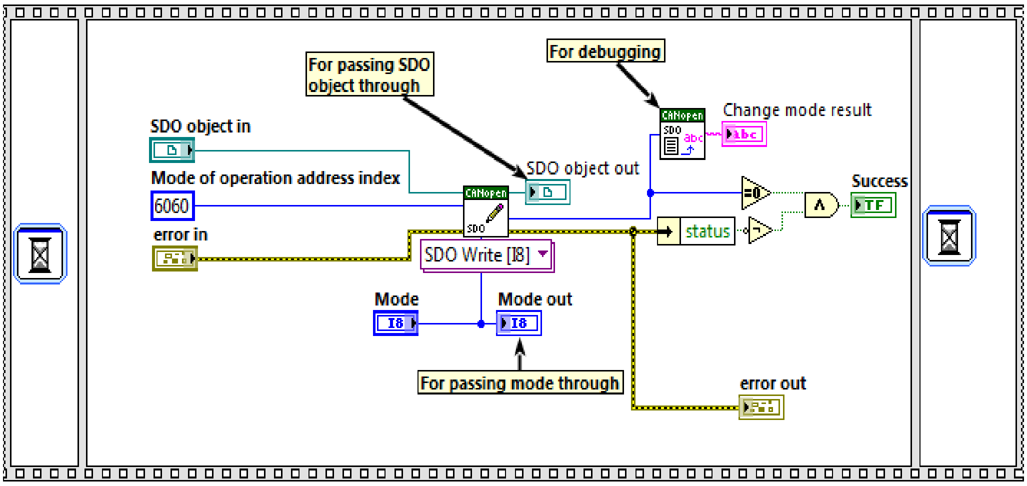


Figure 48: Mode_of_operation VI.

It is important to note that for safety reasons, the Z-axis is always started first in order to move it to a safe position before moving the X and Y axes. Furthermore, X and Y axes have a common “positioned” bit, while Z-axis has its own “positioned” bit. Therefore, to confirm positioning of the fixture, the X/Y “positioned” bit has to be checked.

5 RECONFIGURABILITY ASSESSMENT

This chapter discusses the assessment of the reconfigurability of the rivet feeder station. The reconfigurability assessment entails carrying out experiments and cycle time analysis to determine whether the key characteristics of reconfigurable manufacturing systems have been met. The experiments conducted are as follows:

- Changing the product type
- Introducing new devices to the station
- Simulating station diagnostics

5.1 Experiment 1: Changing the product type

This experiment was aimed at demonstrating the convertibility of the rivet feeder station by changing the product type. In this experiment, the Q-range of circuit breakers was used. However, a change in the product type was introduced by:

- Changing the types and orientations of the fixtures
- Changing the number of rivets fed into holes of circuit breaker assemblies

Two different fixture sizes were used to introduce a change in the product type by varying the thicknesses of circuit breaker assemblies. In addition, the orientations of the fixtures were changed in order to introduce a change of the positioning coordinates for each rivet. The steps involved were: unloading the circuit breaker assemblies from the fixtures, dismounting the fixtures, mounting the fixture back in a different orientation, loading the circuit breaker assemblies into the fixtures and finally obtaining the positioning coordinates of the new product. The positioning coordinates were obtained, using a model in Autodesk Inventor 2013, by locating the positions of the rivet holes relative to one rivet hole that was chosen as the reference.

Changing the number of rivets fed into the holes of circuit breaker assemblies by eliminating a set of positioning coordinates also changed the product type since it implied a reduction in the number of holes on the circuit breakers. This experiment was performed for both the initial orientations of the fixtures as well as after changing the orientations of the fixtures.

The changing of the product type resulted in the change of product information required to successfully accomplish rivet placement. Therefore, this entailed creation of new product holons. Each product holon stored the product information mapped to a specific product type. Therefore, this experiment also demonstrated the possibility of dynamically creating product holons, each time a new product type is launched in the rivet feeder station. As mentioned earlier in section 4.3.5, it was possible to dynamically create new product holons using a hash table-like storage data structure.

For this experiment, it should be noted that changing the product type did not result in the reconfiguration of any holon since the controller was designed to allow for creation of multiple product holons in the Product Holon Manager.

5.2 Experiment 2: Addition of new devices

Experiment 2 was aimed at demonstrating the integrability of the LabVIEW based controller with new devices. Integrability, as defined by Koren et al (2010), is the ability to integrate modules rapidly and precisely by a set of mechanical, informational and control interfaces that facilitate integration and communication. Therefore, integrability was demonstrated by adding a solenoid valve and a single acting cylinder to simulate the addition of multiple singulation devices in order to increase the throughput rate of rivet singulation. Furthermore, the addition of devices led to the addition of the Singulation Unit Holon (SUH) to the controller.

The SUH is a Resource Holon added in order to simulate the singulation of rivets from two singulation devices. One singulation device was assumed to be singulating rivets when the cylinder was in its retracted state and the other singulation device was assumed to be singulating rivets when the cylinder was in the extended state. Furthermore, the extending and retracting of the cylinder was controlled through a sequence structure and using timing. Timing the operation of the cylinder was necessary as it had to sequence with the pick-n-place mechanism. Therefore, the cylinder could only extend or retract during the extending motion of the pick-n-place mechanism.

Addition of the SUH to the controller resulted in the reconfiguration of the OH by adding the *Send Notification.vi* as shown in Figure 49. The OH was reconfigured in order to send a command to the SUH to start switching between the singulating devices so that more singulated rivets are made available for placement.

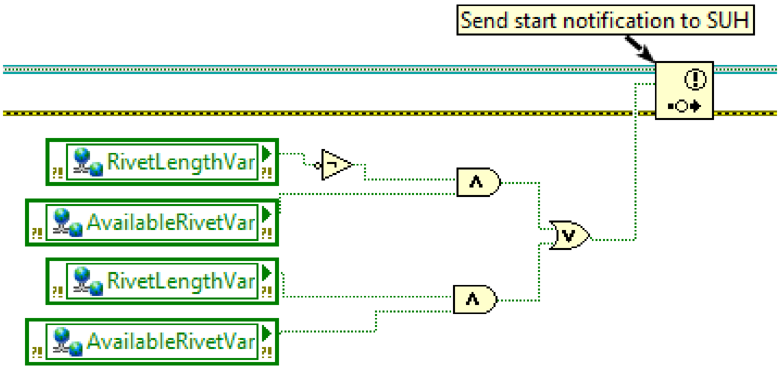


Figure 49: OH sending command to SUH by notifier VI.

From Figure 49, the variable names *RivetLengthVar* and *AvailableRivetVar* were used to give an indication of the length of singulated rivets and rivet availability, respectively. Furthermore, it can be seen from Figure 49 that the command sent to the SUH is a *true* Boolean value. The *true* Boolean value is sent on either of the two conditions:

- When the singulated rivet has been detected but rivet availability not detected
- When both the singulated rivet and rivet availability have not been detected

Addition of the SUH and the reconfiguration of the OH took three hours and twenty three minutes and required the expertise of at least a technician familiar with LabVIEW.

5.3 Experiment 3: Perform station diagnostics

Experiment 3 involved performing the rivet feeder subsystem's diagnostics to demonstrate its diagnosability. Diagnosability is the ability to automatically read the current state of a system to detect and diagnose the root causes of output product defects, and quickly correct operational defects (Koren et al., 2010). LabVIEW has debugging tools which makes it possible to carry out diagnostics to identify sources of errors.

In this experiment, the sources of errors investigated in the controller were mainly due to timeouts. A timeout error occurred when a response was not received within a specified period of time. Therefore, when a timeout occurred in the controller, an error message was sent to the CC. Furthermore, the error message was also displayed to the user and the station triggered to stop. Table 6 shows the station controller's timeouts that were investigated.

Table 6: Station controller timeout errors

TIMEOUT	CAUSE
Receive ProductInfo	OH has not received information from PHM
XYZ completion	OH has not received completion from XYZ Table Holon
Pick-n-Place completion	OH has not received completion from Pick-n-Place Holon
Receive Info from CC	PHM has not received information from CC
Completion from motor controllers	XYZ Table Holon has not received completion from the motor controllers
Completion from sensor	Pick-n-Place Holon has not received completion from the proximity sensor

The timeouts in Table 6 were investigated by specifying the time, in milliseconds, that the function waits for an element (information) to become available in the queue if the queue was empty. According to NI LabVIEW (2013) the default value is –1, indicating never to timeout. However, the default value was not used in the LabVIEW based controller as it was going to defeat the purpose of conducting diagnostics by means of timeout errors. Thus, timeout values were specified for investigations.

Figure 50 shows an example of a case structure for handling timeout errors which occur when the Order Holon has not received information from the Product Holon Manager within the specified time. All the timeout errors were handled in a similar manner shown in Figure 50. Furthermore, as can be seen from Figure 50, when the timeout is true on a queue function, the true case of the structure is executed, thereby sending the error message to the CC.

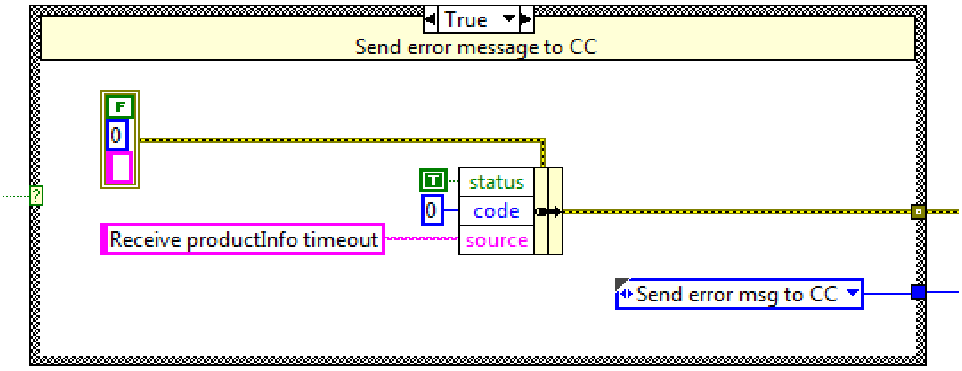
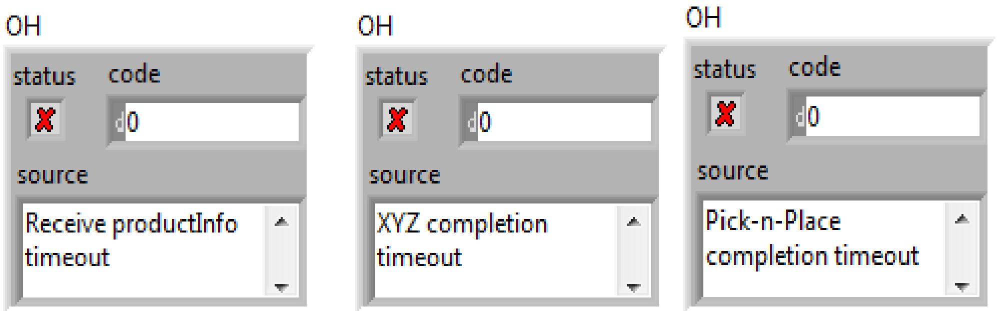


Figure 50: Example case structure for handling timeout errors.

From the investigation undertaken, the timeout errors were generated and displayed to the user through the front panel. The error messages displayed for the first three timeouts on Table 6 are as shown in Figure 51.



OH
status ✗
code 0
source
Receive productInfo timeout

OH
status ✗
code 0
source
XYZ completion timeout

OH
status ✗
code 0
source
Pick-n-Place completion timeout

Figure 51: Timeout error message displays.

5.4 Cycle time analysis

In addition to the three aforementioned experiments, cycle time of rivet placement was investigated. The investigation involved varying the time delays used to sequence the pick-n-place mechanism and the XYZ positioning table, in the same ratios. In other words, the time delays were iteratively divided by the following factors: 1, 2 and 2.5. The factors could not be increased beyond 2.5 since the XYZ positioning table then could not reach the required position on time. The minimum cycle time was found to be 12 seconds per breaker. This entails that two rivet feeder subsystems are required in order to attain the cycle time of 6 seconds per breaker, specified in the design requirements of the rivet feeder station.

6 CONCLUSIONS AND RECOMMENDATIONS

This thesis used a rivet feeder station as a case study to establish the feasibility of automating the process of placing rivets into the holes of circuit breaker assemblies. The rivet feeder station consisted of a vibratory bowl feeder, a singulation device, a pick-n-place mechanism and an XYZ positioning table. The objective of this thesis was to determine whether LabVIEW is a suitable development environment for implementing holonic control.

The LabVIEW based control was developed using the PROSA reference architecture, which is made up of the Product, Resource, Order and Staff Holons. The developed controller comprised the following holons: Coms Holon, Request Manager Holon, Order Holon, Product Holon Manager, Pick-n-Place Holon and XYZ Table Holon. All the holons of the controller except the Coms Holon were implemented using the state machine architecture. Furthermore, communication among the holons was achieved by the available methods of transferring data in LabVIEW such as queues, notifiers, shared variables etc.

From this thesis, it can be concluded that NI LabVIEW development environment promises to be a suitable platform for implementing holonic control owing to its modular nature. By taking advantage of LabVIEW's modular nature, it was possible to separate the functionality of holons. LabVIEW is suitable for multithreaded applications since it executes code in an inherently parallel manner. Therefore, all the holons developed in LabVIEW based controller had while loops running in parallel.

Another key feature identified from LabVIEW is the exception handling which improves the robustness and reliability of the code. Furthermore, LabVIEW's interactive debugging tools, such as execution highlighting, probes, break-points and single-stepping, were utilized during the development of the controller. The debugging tools were accessible on the block diagram's toolbar.

The reconfigurability assessment was conducted on the rivet feeder subsystem and involved three experiments: changing the product type, introducing new devices to the station and simulating station diagnostics. The assessment required conducting reconfiguration on the Order Holon, after adding the Singulation Unit Holon. Furthermore, from the reconfigurability assessment, it was demonstrated that the design of the rivet feeder station satisfied the key characteristics of reconfigurable manufacturing systems.

The convertibility of the rivet feeder station was demonstrated by changing the product types through the re-orientation of two different types of breaker fixtures as well as through changing the number of rivets fed. The experiment conducted to demonstrate convertibility required the creation of new Product Holons, one for each new product type.

Integrability is another good feature of LabVIEW that was investigated. According to National Instrument (2013), it is possible to use all hardware with a single LabVIEW development environment. Connectivity was made possible with driver software, which served as the communication layer between LabVIEW and the hardware. The driver software made it possible to interface the controller with the sensors and actuators via the NI cRIO-9068 and chassis system. The experiment done to demonstrate integrability was achieved by adding devices to the NI 9375 module (I/O module), i.e. sensors and actuators. Furthermore, a solenoid valve and a single acting cylinder added to the station simulated the addition of a Singulation Unit Holon. Therefore, the experiment did not only demonstrate integrability, but also scalability and convertibility.

The other three characteristics of RMSs were also assessed. Diagnosability of the rivet feeder station was assessed by conducting timeout diagnostics. As mentioned earlier, modularity was identified as one of the features possessed by LabVIEW. Furthermore, customizability of the rivet feeder station was achieved by adding a modular rivet orientation device to the vibratory bowl feeder.

The main limitations and disadvantages of LabVIEW, for the implementation of holonic control systems were found to be the graphical programming which makes the block diagram cumbersome to follow, for complex applications. Furthermore, dynamic instantiation of objects or memory cannot be achieved in LabVIEW. Hence, multiple Order Holons could not be created, as is conventionally done in holonic control. However, LabVIEW holds the following advantages: shared variables and TCP/IP components simplify communication over a network; easy construction of the graphical user interface using controls and indicators on the front panel, an XML format in LabVIEW provides flexibility to create your own unlimited tags, and immediate compilation of LabVIEW programs. In addition, LabVIEW's modular nature allows users to re-use the same code by converting it into SubVIs, which saves development time. Another advantage of LabVIEW to the users is that it has built-in functions that can be used for data acquisition and also storing the acquired data. Furthermore, the users can call a dynamic link library (DLL) from LabVIEW, using the *Call Library Function* node found under the Libraries and Executables VIs palette. This enables the use of code written in other programming languages such as C++. Lastly, as mentioned earlier, LabVIEW can be integrated with hardware such as the compactRIO with a CANopen card.

Based on the research undertaken for this thesis, the following are the recommendations made:

- Research should be conducted into the possibility of dynamically instantiating Order Holons, to allow for multiple executions of tasks.

- Research should be conducted on how to store product information within the CompactRIO system rather than on the computer’s hard drive, so that the CompactRIO system can be run as a stand-alone application.

- Further research should be done to investigate the feasibility of using multiple pick-n-place mechanisms and singulation devices on the same rivet feeder subsystem, in order to scale up production.

- Further research should be done to implement the rivet feeding concept on the conveyor system, with the XYZ positioning table used for placing rivets into holes of circuit breaker assemblies.

7 REFERENCES

Adams, A.O., 2010. ‘*Control of a Reconfigurable Assembly System*’. MSc.Eng. Thesis. Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Bi, Z.M., Lang, S.Y.T., Shen, W & Wang, L., 2008. ‘*Reconfigurable manufacturing systems: the state of the art*’. International Journal of Production Research, Volume 46, No.4, pp. 967-992.

Brennan, R. and Fletcher, M., 2002. ‘*An agent-based approach to reconfiguration of real-time distributed control systems*’. IEEE transaction on Robotics and Automation, Volume 18, No.4, pp.444-451.

Deloitte, 2013. ‘*How do you remain competitive as a manufacturing company in South Africa?*’ [Online] Available from: http://deloitteblog.co.za/2013/04/02/how-do-you-remain-competitive-as-a-manufacturing-company-in-south-africa/. [Accessed: 6th August 2013]

Duffie, N.A. and Prabhu, V.V., 1996. ‘*Heterarchical control of highly distributed manufacturing systems*’. International Journal of Computer Integrated Manufacturing, Volume 9, No.4, pp.270-280.

EIMaraghy, H.A., 2006. ‘*Flexible and reconfigurable manufacturing systems paradigms’*. International Journal of Flexible Manufacturing Systems, Volume 17, pp. 261-276.

Elliott, C., Vijayakumar, V., Zink, W. and Hansen, R., 2007. ‘*National Instruments LabVIEW: A programming environment for laboratory automation and measurement’*. Journal of Laboratory Automation, Volume 12, No.1, pp.17-24.

FESTO, 2012. *'Festo Motorcontroller CMMP system manual'*.

Halvorsen, H., 2011. ‘*Introduction to state-based Applications in LabVIEW*’. Tutorial, Department of Electrical Engineering, Information Technology and Cybernetics, Telemark University College.

Halvorsen, H., 2012. ‘*Introduction to LabVIEW*’. Tutorial, Department of Electrical Engineering, Information Technology and Cybernetics, Telemark University College.

Humayun, S., Mehmood, M., Mahmood, F. and UlIslam, Q., 2013. ‘*A study and comparison of data transfer methods in LabVIEW*’. World applied sciences journal, Department of Electrical Engineering, Institute of space technology, Islamabad, Pakistan, Volume 28, pp.1772-1775

Hussain, T. and George, F., 2007. ‘*Deployment of IEC 61499 compliant distributed control applications*’. IEEE, pp.502-505.

Katz, R., 2007. ‘*Design principles of reconfigurable machines*’. International Journal of Advanced Manufacturing Technology, Volume 34, pp. 430-439.

Koren, Y, 2005. ‘*Reconfigurable manufacturing and beyond’*. The summary of a keynote speech, CIRP 3rd international conference on Reconfigurable manufacturing.

Koren, Y. and Shpitalni, M., 2010. ‘*Design of reconfigurable manufacturing systems’*. Journal of Manufacturing Systems, Volume 29. pp. 130-141.

Kruger, K., 2013. ‘*Control of Feeder Subsystems in Reconfigurable Manufacturing Systems'*. MSc.Eng Thesis, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Leitao, P. and Restivo, F., 2006. '*ADACOR-A Holonic architecture for agile and adaptive manufacturing control*’. Computers in Industry, Volume 57, pp. 121-130.

Malhotra, V., Raj, T. and Arora, A., 2009. *‘Reconfigurable manufacturing system: an overview*’. International Journal of Machine Intelligence, Volume 1, Issue 2, pp. 38-46.

Malhotra, V., Raj, T and Arora, A., 2010. ‘*Excellent techniques of manufacturing systems: RMS and FMS’*. International Journal of Engineering Science and Technology, Volume 2, Issue 3, pp. 137-142.

Mehrabi, M.G., Ulsoy, A.G. and Koren, Y., 2000. *‘Reconfigurable manufacturing systems: Key to future manufacturing’*. Journal of Manufacturing, Volume 11, pp. 403-419.

Meng, F., Tan, D. and Wang, Y., 2006. ‘*Development of Agent for Reconfigurable Assembly System with JADE*’. Proceedings of the 6th World Congress on Intelligent Control and Automation, Dalian, China. pp. 7915-7919.

Mulubika, C., 2013. ‘*Evaluation of Control Strategies for Reconfigurable Manufacturing Systems’*. MSc.Eng. Thesis. Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Nair, R.S., 2012. ‘*Design of a Robotic Arm for picking and placing an object controlled using LabVIEW* ’. International Journal of Scientific and Research Publications, Volume 2, Issue 5, pp. 1-4.

National Instruments, 2003. ‘*LabVIEW*’. User manual, April Edition, pp. 1-349.

National Instruments, 2005. ‘*LabVIEW*’. LabVIEW fundamentals, pp.62-68.

National Instruments, 2010. ‘*Industry Leaders use LabVIEW to improve Test*’. [Online] Available from: http://zone.ni.com/devzone/cda/pub/p/id/1236 [Accessed: 10th October 2013].

National Instruments, 2012. ‘*Advantages of using LabVIEW in academic research*’. [Online] Available from: http://www.ni.com/white-paper/8534/en/[Accessed [Accessed: 31st July 2013].

NI LabVIEW, 2013. ‘*LabVIEW help*’. Handling errors.

Setch, R.M. and Lagos, N., 2004. ‘*Reconfigurability and reconfigurable manufacturing systems: state-of-the art*’. IEEE Conference on Industrial Informatics, INDIN, pp. 529-535.

Scholz, R. B. and Freitag, M., 2007, ‘*Autonomous Processes in Assembly Systems*', CIRP Annals, Volume 56, Issue 2, pp. 712 – 729.

Thuyphamxuan, 2013. ‘LabVIEW design patterns: Software design approaches’. [Online] Available from: http://gtms1311.wordpress.com/2013/03/09/labview-design-patterns-software-design-approaches/ [Accessed: 7th September 2014].

Ullman, D.G., 2003. ‘The mechanical design process’. Third edition, McGraw-Hill Companies, Inc.

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P., 1998. ‘*Reference Architecture for holonic manufacturing systems: PROSA*’. Computers in Industry, Volume 37, pp. 255-274.

.

APPENDIX A: Subsystem functional analysis

Figure 52 shows the functional analysis developed for the rivet feeder subsystem. Setting up the rivet feeder subsystem hardware with control system entails assembling the subsystem's components and programming the low-level hardware by providing I/O interfacing. The functions of positioning circuit breaker fixture and feeding rivets are done after obtaining the positioning coordinates from the cell controller.

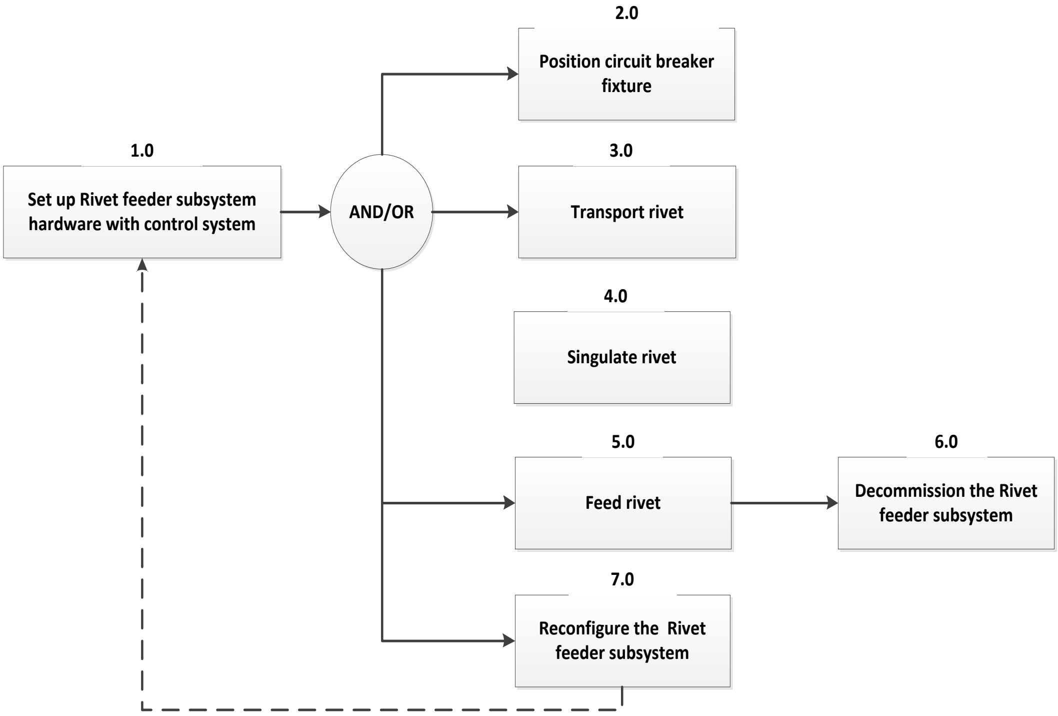

2.0
Position circuit breaker fixture
1.0
Set up Rivet feeder subsystem hardware with control system
AND/OR
3.0
Transport rivet
4.0
Singulate rivet
5.0
Feed rivet
6.0
Decommission the Rivet feeder subsystem
7.0
Reconfigure the Rivet feeder subsystem

Figure 52: Function analysis for the rivet feeder subsystem

APPENDIX B: Suppliers of hardware components

Table 7 shows the hardware components used for the rivet placement station. The table also shows the suppliers of the respective components.

Table 7: Hardware components and suppliers

S/N	Components	Quantity	Supplier
1	cRIO-9068 8-Slot Integrated Controller and Chassis System, Artix-7 FPGA	1	National Instruments
2	NI 9375, 16-ch Sinking DI, 16-ch Sourcing DO, DI/DO C Series Module	1	National Instruments
3	NI 9881, C Series CANopen Interface, 1 Port	1	National Instruments
4	Vibratory bowl-feeder	1	
5	Air cylinder, DSNU-16-100-P-A	1	Festo
6	Parallel gripper, DHPS-20-A	1	Festo
7	5/2 flow control valve	2	Festo
8	Proximity sensor, SMT-8G-…-24V	1	Festo
9	Optic sensor, SOEG-L-Q30-P-A-K-2L 165326bA2 B	2	Festo
10	Modular Cartesian Robot components		Festo

APPENDIX C: Cell controller and inter-station communication

C.1: Standard format of messaging

The Cell Controller communicates with four stations namely electrical test station, rivet placement/feeder station, buffering station and transport system. The communication mechanism adopted is TCP/IP with messages transmitted as XML strings. The standard XML format used for messaging is as follows;

<Message>

<Receiver>
Intended recipient of message
</Receiver>

<Sender>
Sender of the message
</Sender>

<Port>
Number of the port through which the message has been sent
</Port>

<Type>
Type of message
</Type>

<CID>
Unique ID of the conversation of which the message forms part
</CID>

<Msg>
Message content
</Msg>

</Message>

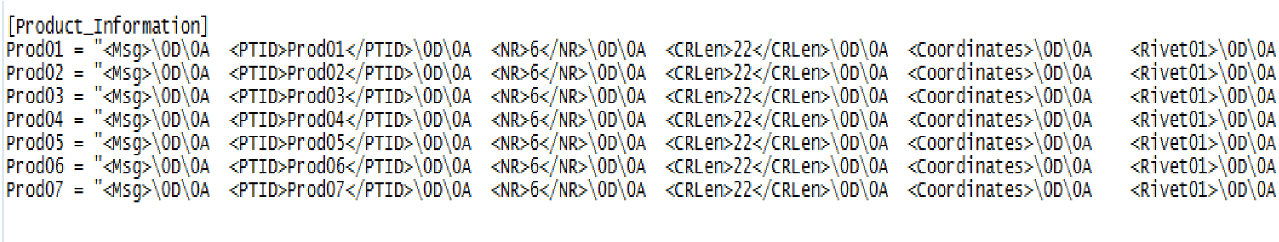
C.2: Rivet feeder station-specific messages

The station specific messages entails specific information included in the content of messages exchanged between the Cell Controller and rivet feeder station. The rivet feeder specific messages are as follows:

```
<Message>
    <Type>
        OrderReq  (Cell Controller (CC) requests holon to feed rivets)
```

```
            ConfOrderInfo (Holon confirms receipt of order information to CC)
            OrderRes  (Holon confirms completion of rivet feeding to CC)
            ProdInfoReq  (Holon requests CC for product information)
            ProdInfoRes (CC responds to holon’s request)
            ConfProdInfo (Holon confirms receipt of product information to CC)
            RivetLenReq (CC enquires length of rivet from holon)
            RivetLenRes  (Holon sends length of rivet to CC)
             RivetSizesReq (CC enquires from holon the sizes of rivets that can be
             fed)
            RivetSizesRes (Holon sends sizes of rivets that can be fed to CC)
            RemTimeReq (CC enquires from holon the remaining time to feed rivets)
            RemTimeRes (Holon sends the remaining time to feed rivets to CC)
        </Type>
        <Msg>
             <CID>   Conversation ID used between CC and the station
             <PTID>    Product type ID
             <VERSION>     Version of information from the CC
             <UID>     Unique ID of the product type
             <PCoord>    X Y Z positioning coordinates
          </Msg>
</Message>
```

C.3: Hash table-like data storage for product information



```
[Product_Information]
Prod01 = "<Msg>\0D\0A  <PTID>Prod01</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
Prod02 = "<Msg>\0D\0A  <PTID>Prod02</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
Prod03 = "<Msg>\0D\0A  <PTID>Prod03</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
Prod04 = "<Msg>\0D\0A  <PTID>Prod04</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
Prod05 = "<Msg>\0D\0A  <PTID>Prod05</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
Prod06 = "<Msg>\0D\0A  <PTID>Prod06</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
Prod07 = "<Msg>\0D\0A  <PTID>Prod07</PTID>\0D\0A  <NR>6</NR>\0D\0A  <CRLen>22</CRLen>\0D\0A  <Coordinates>\0D\0A    <Rivet01>\0D\0A
```

APPENDIX D: Singulation device components

The singulation device shown in Figure 53 was custom made and consists of a steel block and two spring-loaded parts. The steel block has length, width and height of 60 mm, 20 mm and 45 mm respectively. Furthermore, the steel block has a 4mm diameter through hole that runs from the middle on top to bottom and has also a groove at the bottom with the length, width and depth of 60 mm, 10.1 mm and 5.1 mm respectively.

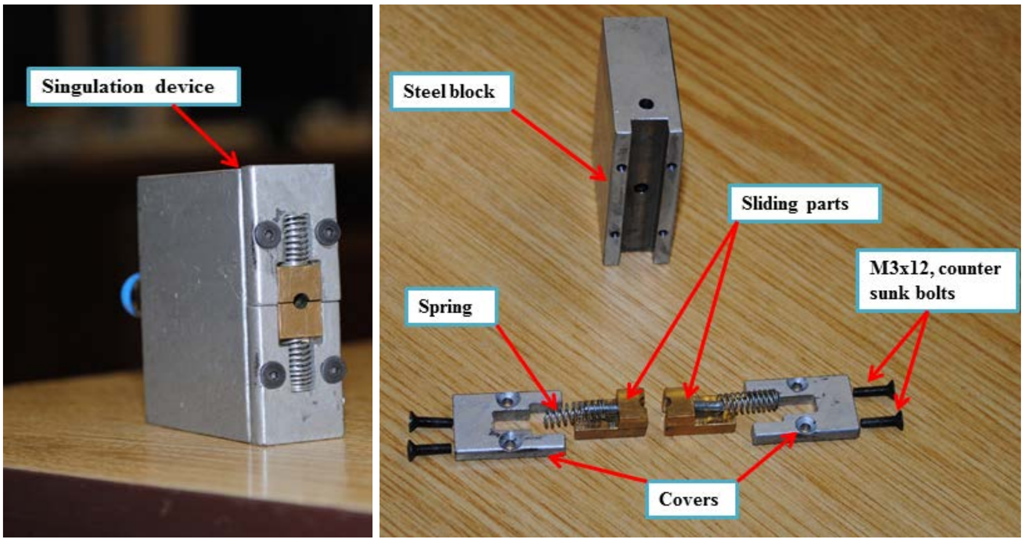


Figure 53: Singulation device and its components.

Two spring-loaded sliding parts made of brass slide apart on the groove by means of a pulled rivet by the pick-n-place mechanism. When stationary the two sliding parts form a conical gap between them. Therefore, rivet singulation is made possible by the conical profile made at the point of contact between the spring-loaded sliding parts.

APPENDIX E: LabVIEW's data communication VIs and the JKI Toolkits

The LabVIEW's data communications VIs implemented in the rivet feeder controller are: *TCP* VIs, *queue operations* VIs and *notifier operations* VIs. These VIs are shown in the figures that follow. Also shown in this appendix are *the JKI toolkits functions* used to generate and parse XML strings in the controller.

- *TCP Open Connection*: Opens a TCP network connection with the *address* and *remote port* or *service name*.

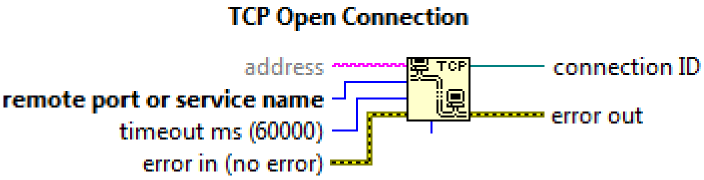


Figure 54: TCP Open Connection.vi

- *TCP Read*: Reads a number of bytes from a TCP network connection, returning the results in *data out*.

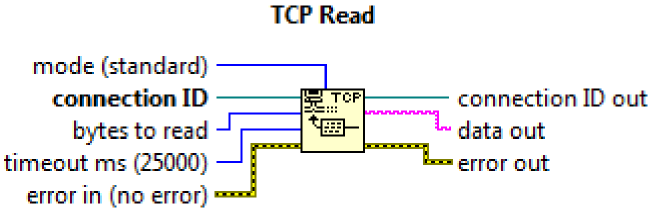


Figure 55: TCP Read.vi

- *TCP Close Connection:* Closes a TCP network connection.

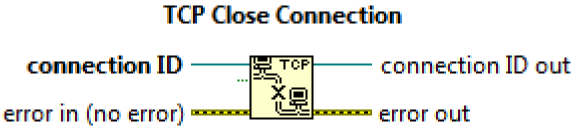


Figure 56: TCP Close Connection.vi

- *TCP Write:* Writes data to a TCP network connection.

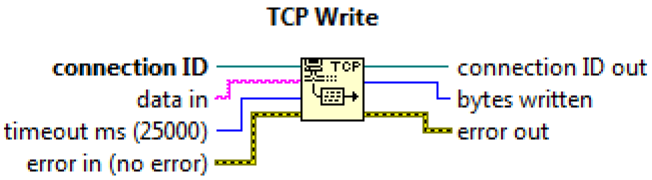


Figure 57: TCP Write.vi

- *TCP Create Listener:* Creates a listener for a TCP network connection.

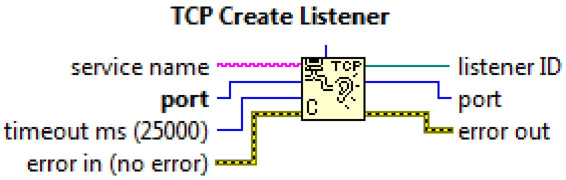


Figure 58: TCP Create Listen.vi

- *Enquere Element:* Adds an element to the back of a queue.

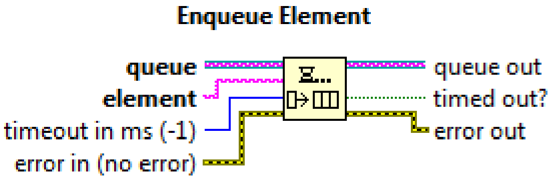


Figure 59: Enqueue Element.vi

- *Get Queue Status:* Returns information about the current state of a queue, such as the number of elements currently in the queue.

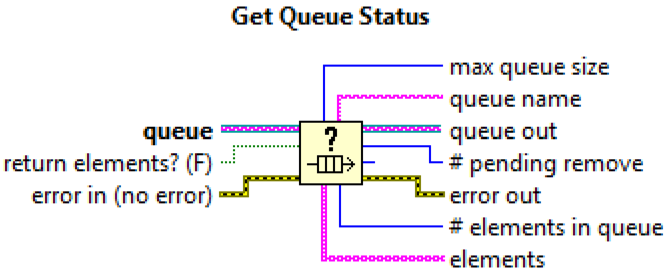


Figure 60: Get Queue Status.vi

- *Dequeue Element:* Removes an element from the front of a queue and returns the element.

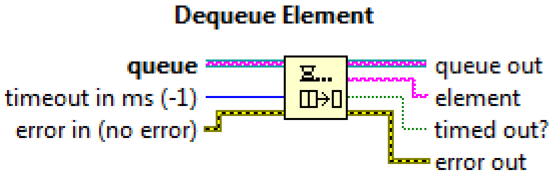


Figure 61: Dequeue Element.vi

- *Release Queue:* Releases a reference to a queue.

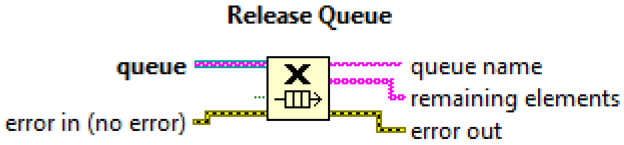


Figure 62: Release Queue.vi

- *Obtain Notifier:* Returns a reference to a notifier.

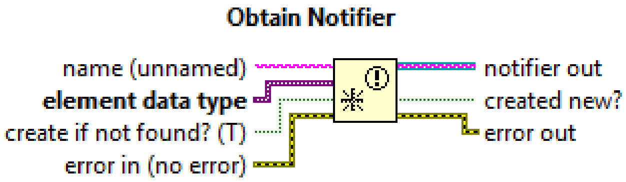


Figure 63: Obtain Notifier.vi

- *Send Notification:* Sends a message to all functions waiting on a notifier.

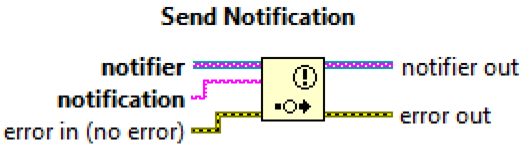


Figure 64: Send Notification.vi

- *Get Notifier Status:* Returns information about the current state of a notifier, such as the last uncancelled notification sent to the notifier.

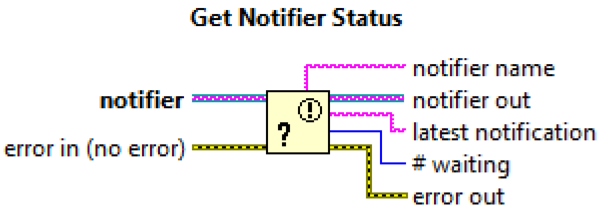


Figure 65: Get Notifier Status.vi

- *Release Notifier:* Releases a reference to a notifier.

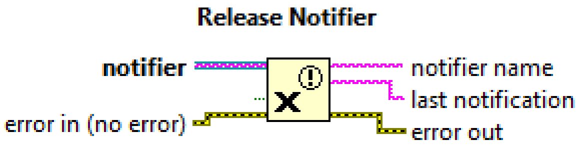


Figure 66: Release Notifier.vi

- *Easy Generate XML:* Converts *LabVIEW Data* to an XML string with the LabVIEW data names (labels) converted to XML item names and the LabVIEW data values converted to XML item values.

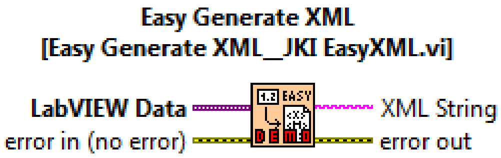


Figure 67: Easy Generate XML.v

- *Easy Parse XML:* Converts an **XML String** to **LabVIEW Data**, based on the **LabVIEW Data (Type)** argument. Use the Variant to Data function to convert the output variant **LabVIEW Data** to the desired type.

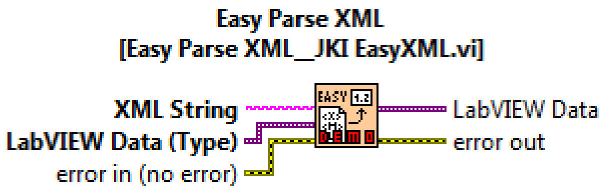


Figure 68: Easy Parse XML.vi

APPENDIX F: Station controller block diagrams

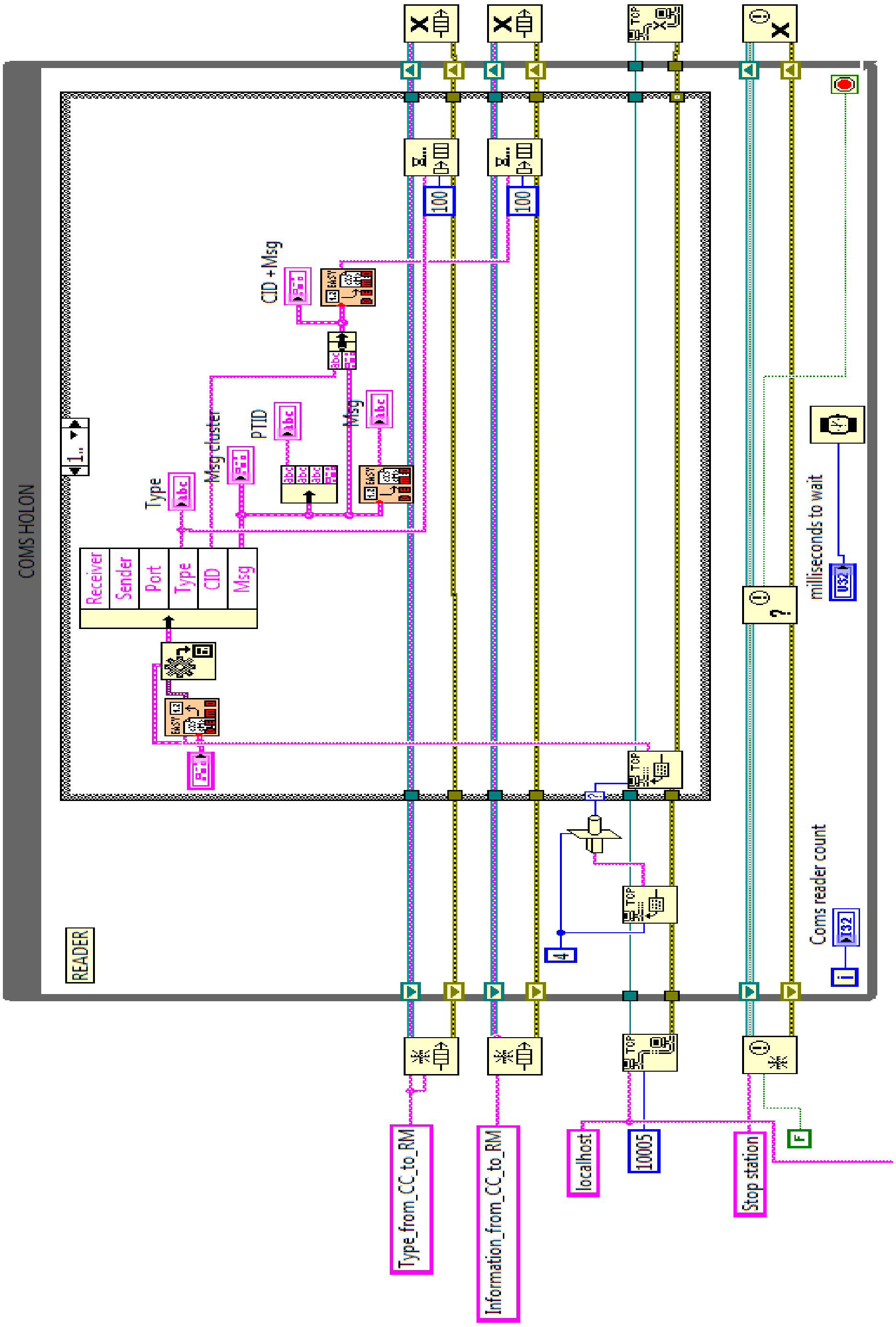


Figure 69: Coms Holon reader.

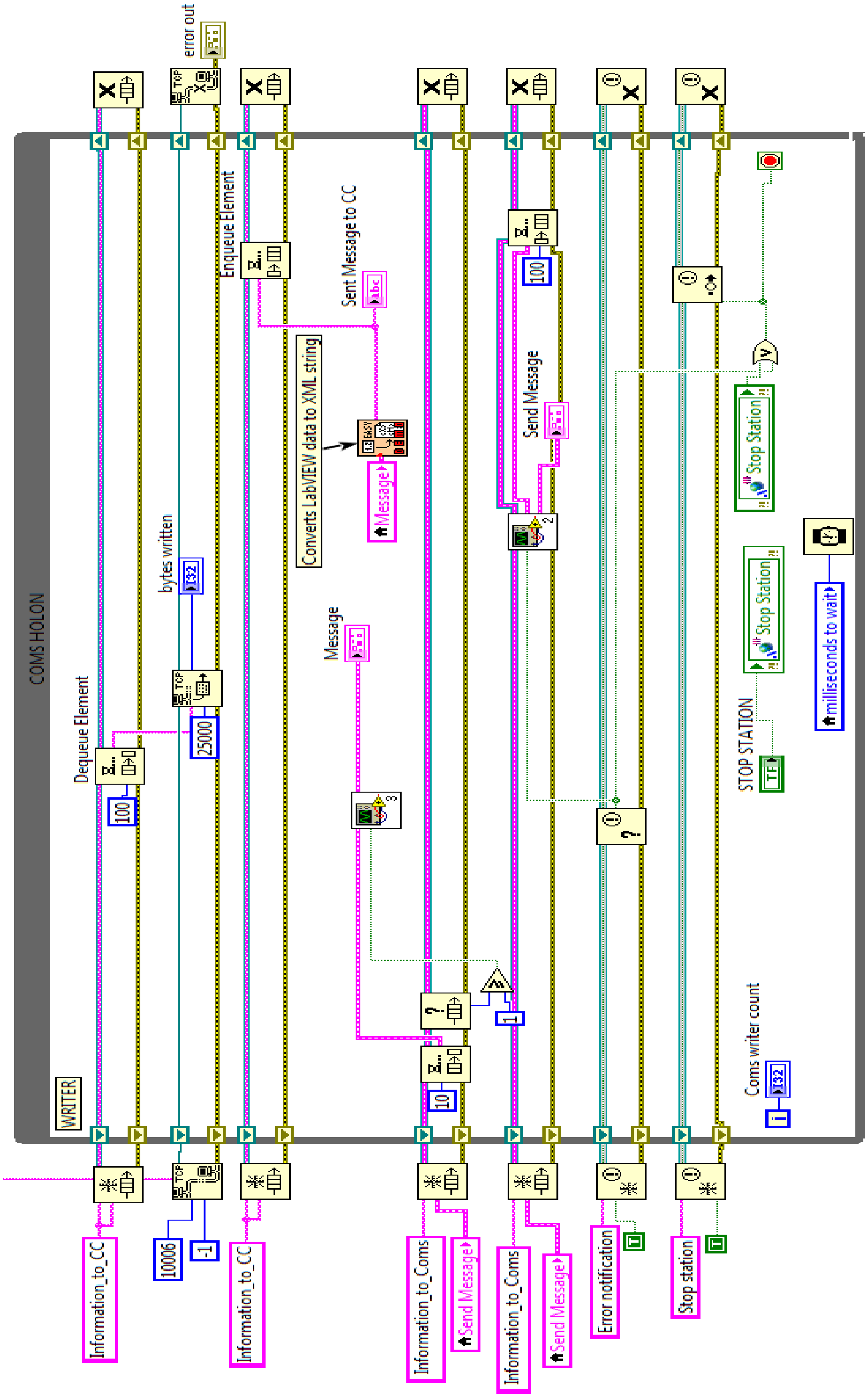


Figure 70: Coms Holon writer.

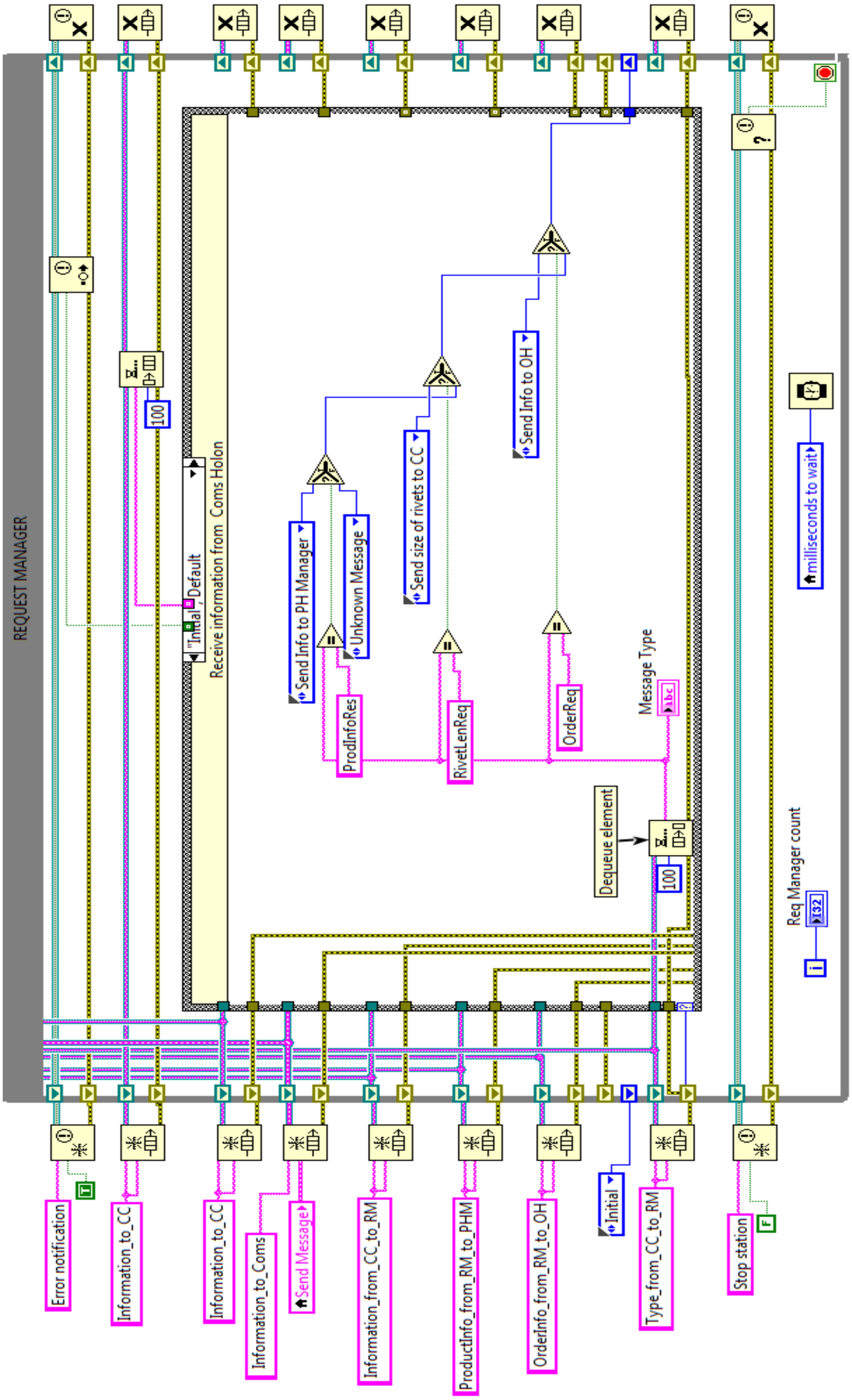


Figure 71: Request Manager Holon-*Initial* state.

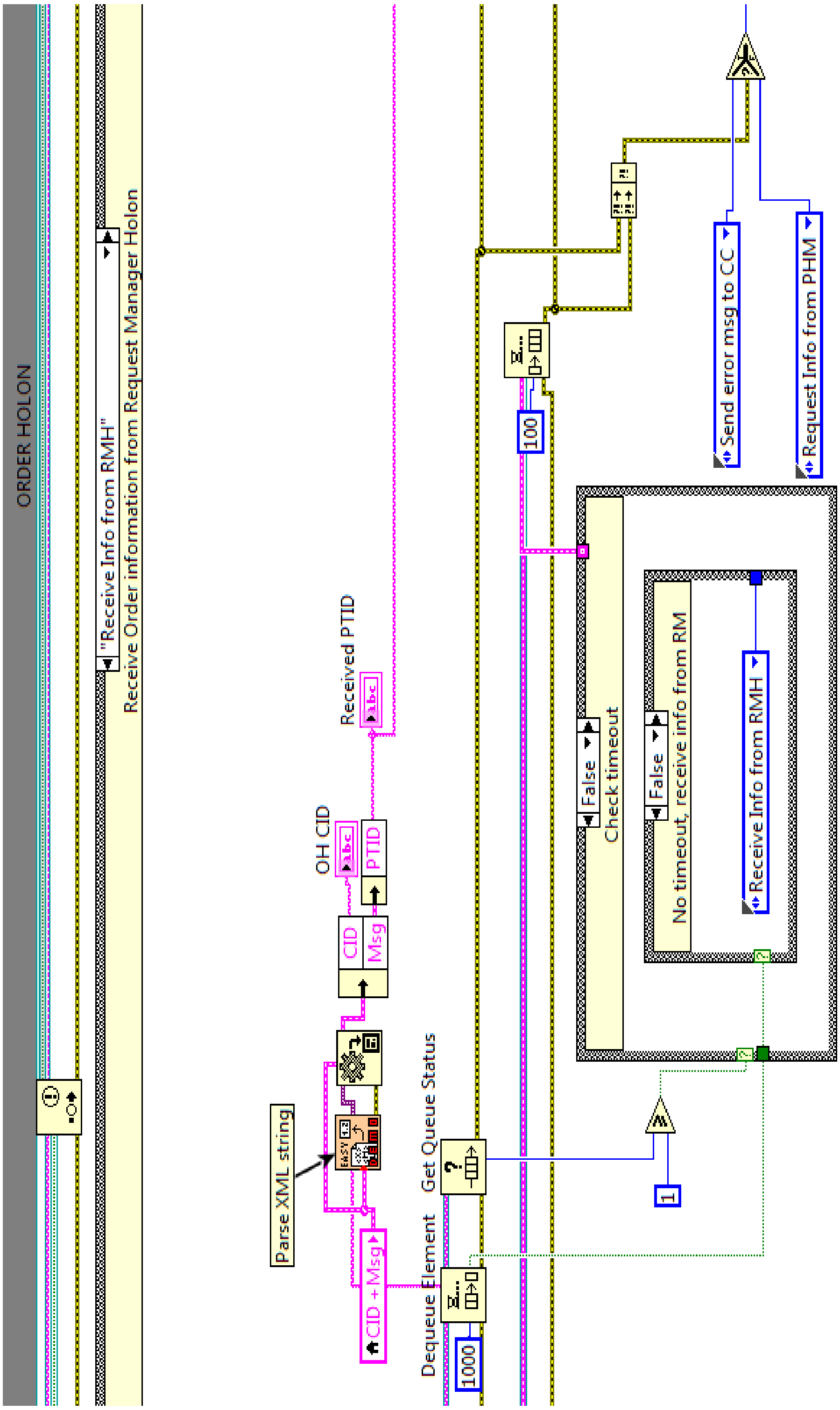


Figure 72: Order Holon-*Receive Info from RMH* state.

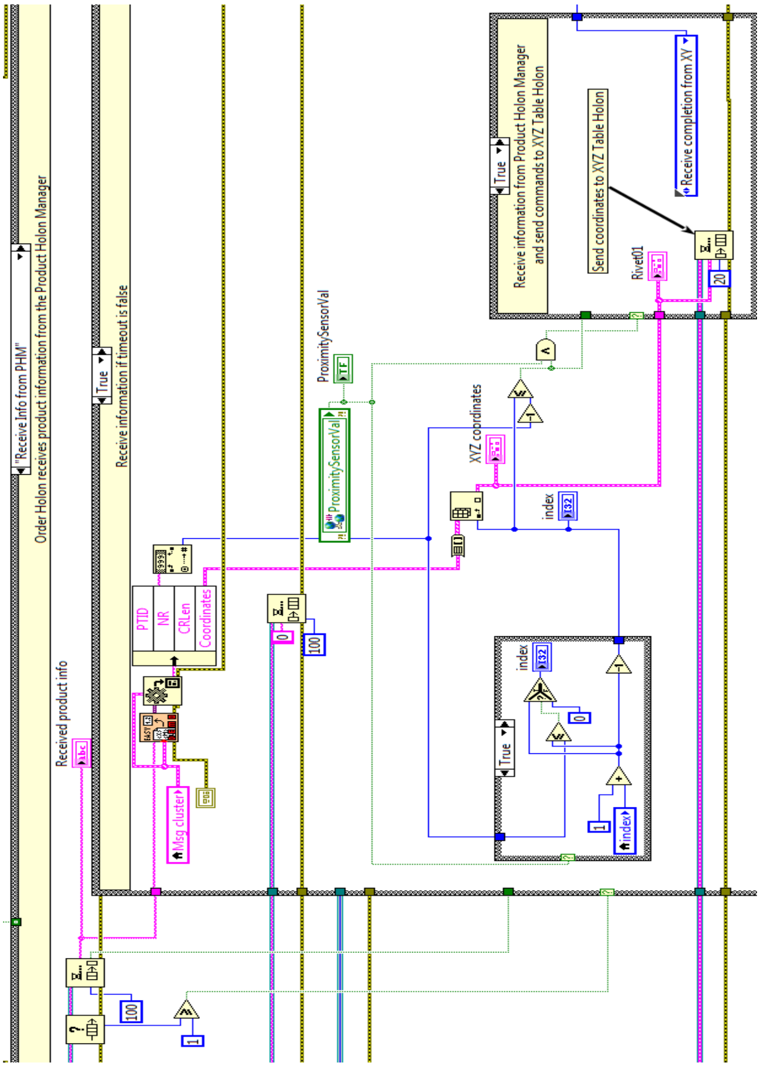

"Receive Info from PHM"
Order Holon receives product information from the Product Holon Manager
True
Receive information if timeout is false
Received product info
Msg cluster
PTID
NR
CRLen
Coordinates
ProximitySensorVal
XYZ coordinates
index
True
index
Receive information from Product Holon Manager and send commands to XYZ Table Holon
Send coordinates to XYZ Table Holon
Receive completion from XY
Rivet01
100
0
1
20

Figure 73: Order Holon-*Receive Info from PHM* state.

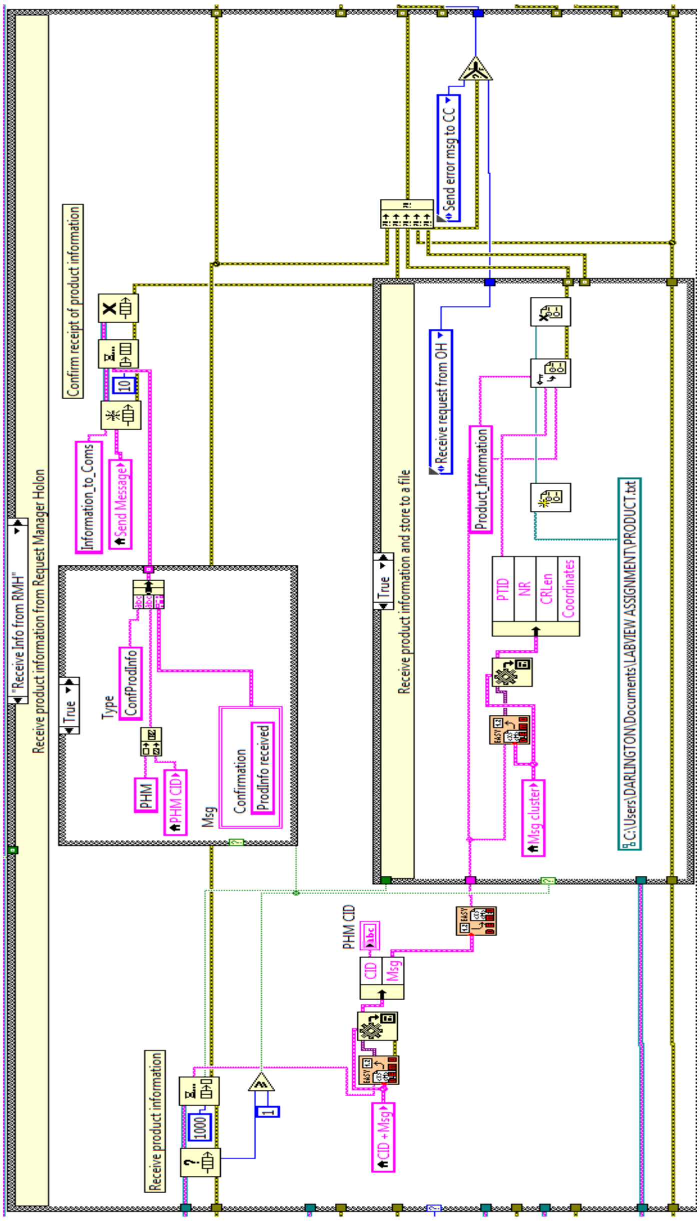


Figure 74: Product Holon Manager-*Receive Info from RMH* state.

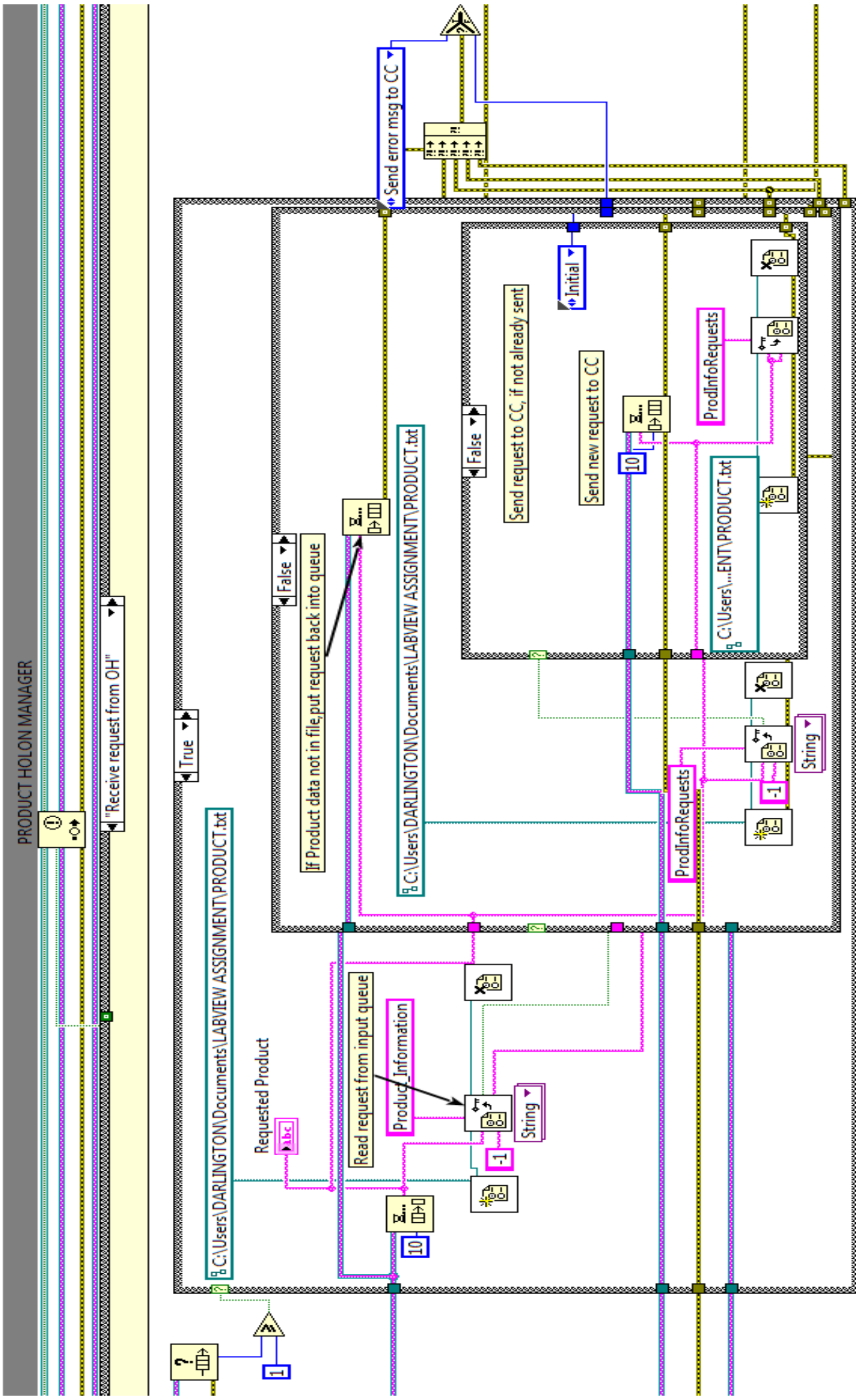

PRODUCT HOLON MANAGER
"Receive request from OH"
True
C:\Users\DARLINGTON\Documents\LABVIEW ASSIGNMENT\PRODUCT.txt
Requested Product
Read request from input queue
Product_Information
String
-1
10
1
False
If Product data not in file,put request back into queue
C:\Users\DARLINGTON\Documents\LABVIEW ASSIGNMENT\PRODUCT.txt
False
Send request to CC, if not already sent
Send new request to CC
Initial
10
ProdInfoRequests
C:\Users\...ENT\PRODUCT.txt
ProdInfoRequests
String
-1
Send error msg to CC

Figure 75: Product Holon Manager-*Receive request from OH* state.

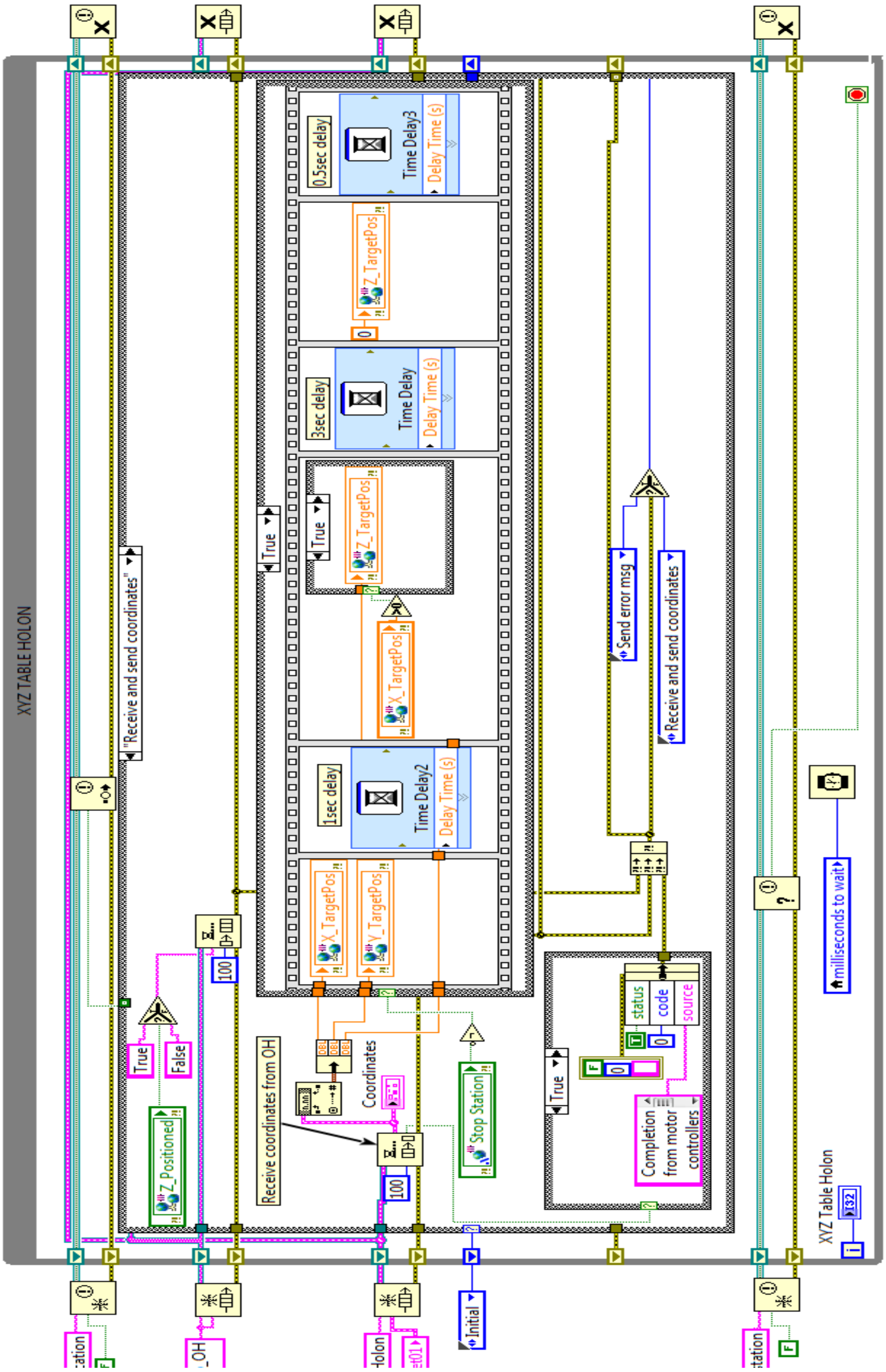


Figure 76: XYZ Table Holon-*Receive and send coordinates* state.

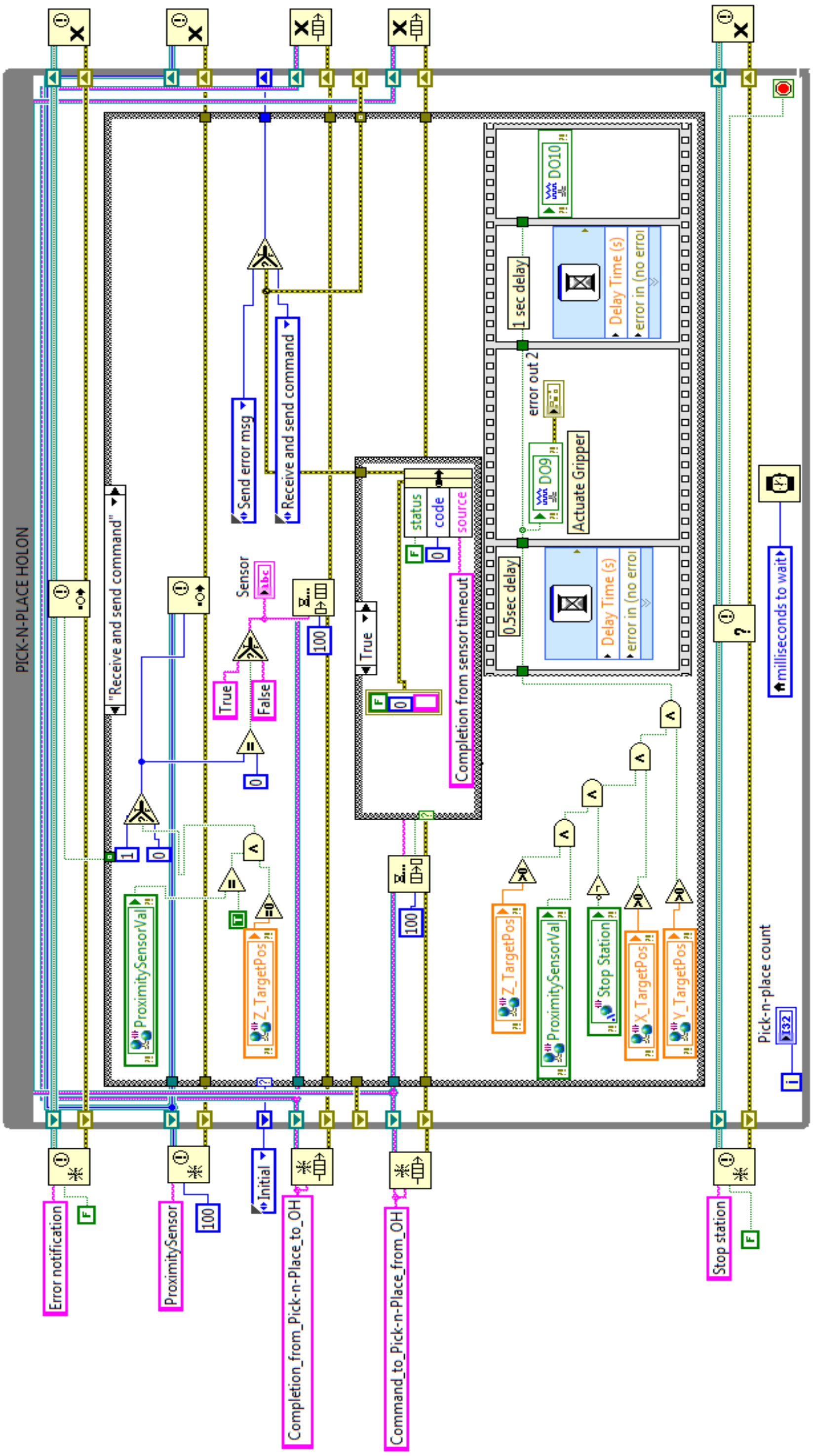


Figure 77: Pick-n-Place Holon-*Receive and send command* state.

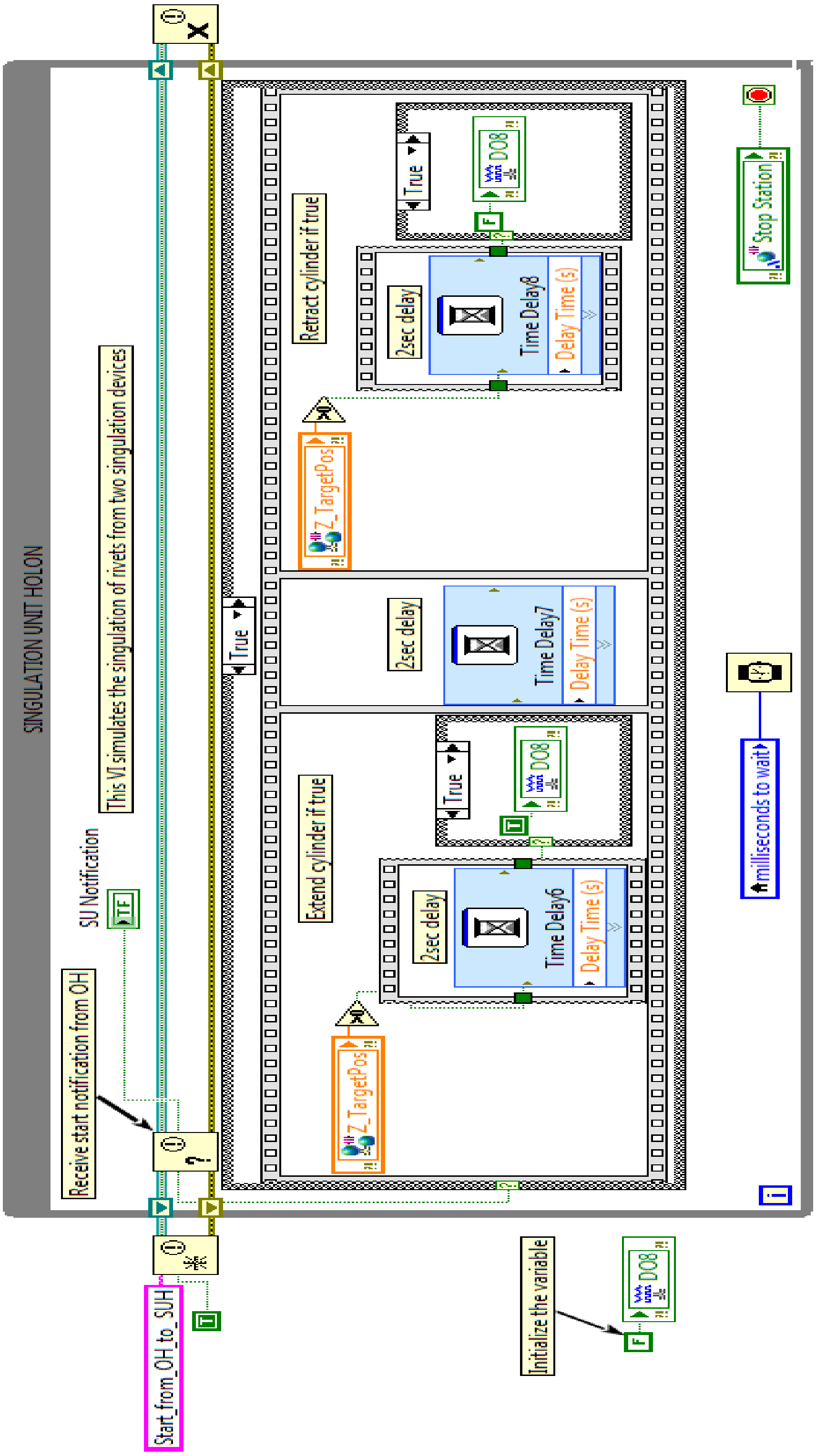


Figure 78: Singulation Unit Holon.