DEVELOPMENT OF AN INTELLIGENT SELF-LEARNING PRODUCT ASSEMBLY SYSTEM USING VISUAL IDENTIFICATION

LUKE ROGERS

Dissertation submitted in fulfilment of the requirements for the degree

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

in the

Department of Electrical, Electronic and Computer Systems Engineering

of the

Faculty of Engineering and Information Technology

at the

Central University of Technology, Free State

Supervisor:

Prof HJ Vermaak (PhD)

Bloemfontein
2018

Declaration

I, Luke Rogers (identity number , student number) hereby declare that this research project which has been submitted to the Central University of Technology for the degree MASTER OF ENGINEERING: ELECTRICAL, is my own independent work, complies with the Code of Academic Integrity as well as other relevant policies, procedures, rules and regulations of the Central University of Technology, and has not been submitted before by any person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

Luke Rogers
Student

Acknowledgements

The author would like to acknowledge the following institute and individuals without whom the completion of this dissertation would not have been possible:

- Professor Herman Vermaak for all his support and guidance over the past four years.
- Doctor Nicolaas Luwes, the activity leader of RGEMS, for all his support.
- The Research Group in Evolvable Manumation Systems (RGEMS) for access to their equipment to attempt this project.
- The Central University of Technology, Free State (CUT), for their financial support and the opportunity to undertake this project.

Abstract

Modern automation systems rely on fixed programming to carry out their production routines. These systems are effective for production outputs but do not allow any flexibility within the production routine. Effort is required to change the ongoing production routine through reprogramming, redesign or complete overhaul of the system to cater for new production outputs. These efforts require down time and result in a loss of revenue.

If a completely automated flexible system is introduced into such a production line, the complete reprogramming process required to cater for new production needs could be automated without losing production time. Within this study, a real-time KUKA Robotic Control system is introduced. The KUKA Robotic Controller maintains its original programming methods with no reprogramming required when executing a new production assembly. This is achieved through manoeuvring the KUKA Robotic System in real-time to new destinations based on image-processing outputs and feedback.

For demonstration purposes and proof of concept, the system learns a design presented to it by an end user and then reproduces this seen design based on the image-processing results in terms of location and orientation. Therefore, instead of reprogramming each new required position, the system takes over real-time control of the KUKA Robotic System and carries out the required steps autonomously.

The benefit of such a system would be that the KUKA Robotic System would not require reprogramming to carry out new routines. It is controlled in a real-time environment to carry out new procedures based on external sensors (in this case, image-processing outputs). KUKA Robotic Sensor Interface (RSI) software is used to implement real-time control of the KUKA Robotic System and is explored extensively throughout this study.

Table of Contents

List of Figures

List of Tables

List of Equations

Acronyms and Abbreviations

CPU	Central Processing Unit
EOAT	End of arm Tool
FPS	Frames per Second
I/O	Input/output
IP	Internet Protocol
IPOC	Interpolation Cycle
PC	Personal Computer
PLC	Programmable Logic Controller
ROI	Region of Interest
RSI	Robotic Sensor Interface
TCP	Tool Centre Point
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 Chapter 1: Introduction

The manufacturing industry is a challenging yet changing market. Flexibility within this industry is highly sought after in any modern production system. This versatile industry is driven by customer quality needs and the ability to respond to changes swiftly and at the lowest cost [1].

Flexible manufacturing systems (FMS) must be able to convert quickly to the production of new models, rapidly adjust capacity and produce an increased variety of products in unpredictable quantities. An FMS can be dedicated or flexible and change as needed. Flexibility can be perceived as a system's capacity to change and assume various positions or states in response to changing requirements, with little or zero penalty in time, effort, cost and performance.

Introducing a flexible, automated adapting system that can adjust its programming routine automatically while satisfying the above-mentioned requirements is a challenge. Flawless system operation also requires uninterrupted accuracy, speed and safety. This research investigates what type of robotic system can be implemented that can adapt automatically in real-time, with zero down time.

A viable visual aid system will also be investigated that could possibly assist the flexible system to achieve its end goals. The visual system would be required to scan the design pallet to detect separate components which make up the design, and calculate optimal movement paths for the KUKA Robotic System to collect and place the detected components. For proof of concept and simplicity purposes, component pallets were made up of the same small shapes placed at 90º angles, while design pallets were made up of multiple shapes placed at random angles.

1.1 Problem Statement

Introducing new products on an existing automated assembly line leads to manual reconfiguration of the entire production line to accommodate the new product's needs. This can be a lengthy process, possibly requiring down time of the entire system. Automating this process using visual aid with zero manual human intervention is key to eliminating system down time. Successful integration of visual aid within an assembly line can be limited, but also beneficial when implemented on a KUKA Robot System.

1.2 Objectives

1.2.1 Aim of the study

The aim is to use a visually aided system that can learn new product designs in real-time, allowing the KUKA Robot System to adjust accordingly without the need for manual reconfiguration. This involves implementing real-time control of the KUKA using the RSI subsystem, which uses data from the visually aided system. This can increase production of newly introduced products, while also having alternative implementation options.

1.2.2 Hypothesis

The designed system should be able to build a new product design completely autonomously without any user intervention or manual KUKA reprogramming. Once a new product is introduced into the existing assembly line, the system should automatically adapt and build the new product, while ensuring all preconditions are met (i.e. the required components for the design must be available within the component catalogue on the component conveyor). The design will consist of multiple components uniquely placed to test the capability of the system.

1.2.3 System Layout

In Figure 1-1 below, we can see the initial floor layout concept. There were two independent conveyor systems designed, namely a “Design Assembly Area” where new designs were studied by the visually aided system and rebuilt, and a “Component Catalogue Area” that maintained a repository of components for rebuilding the seen design within the “Design Assembly Area”.

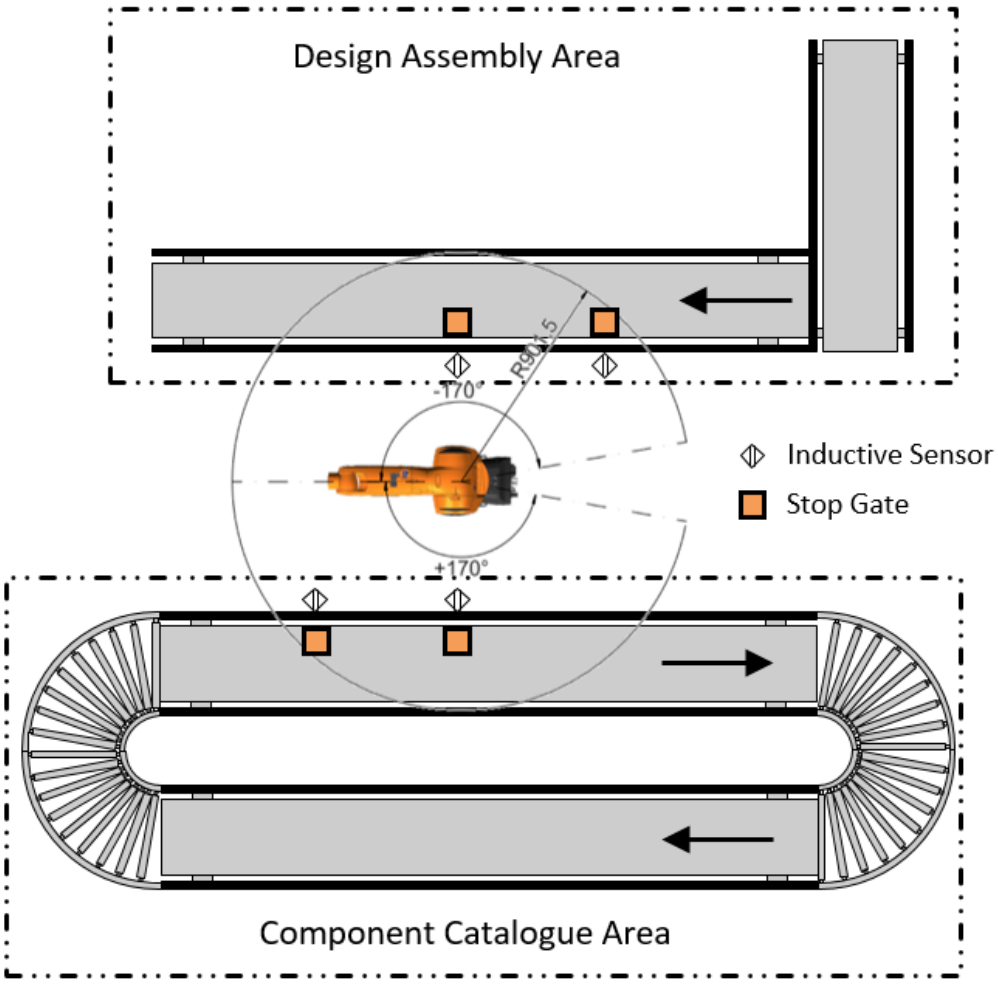


Figure 1-1 Layout of the system used

Each conveyor area was equipped with inductive sensors and stop gates. This allowed for flow control of the pallets within the system while also relaying pallet position data back to the Main Control Interface.

1.2.4 Specific Objectives and contributions

Specific objectives and contributions of the study are to:

- Design an intelligent, self-learning product assembly system.
- Design a visually aided system that can assist the process of moving products automatically between the product pallet and design pallet in real-time.
- Achieve these results using real-time control of a KUKA Robot using the RSI sub-system while investigating its capabilities.
- Support multiple designs in real-time.
- Achieving a reprogramming-free environment on new product entry.
- Smart Pick and Placer

1.2.5 System Overview

The system comprises multiple separate systems (that can work independently) required to work as one unit to achieve the goals set within this study. Figure 1-2 depicts the proposed block diagram of the system. As seen in the figure, all systems communicated over Ethernet standard. A more detailed diagram is presented later within this document.

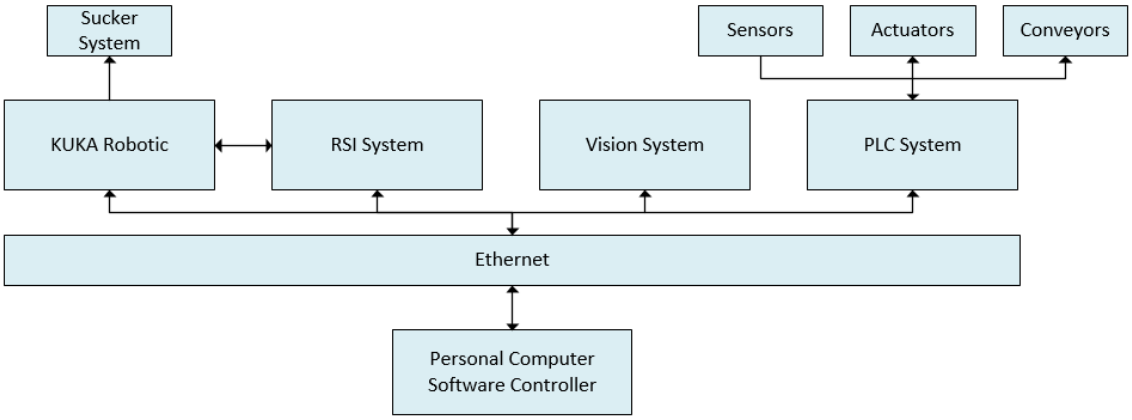


Figure 1-2 Proposed Block Diagram

As mentioned before, each sub-system can operate independently, but requires instructions from the main software controller.

1.3 Layout of the dissertation

- Chapter 1: Introduction to the Thesis — This chapter covers the general overview of the project, outlining goals and objectives.
- Chapter 2: Literature Review — This chapter looks at previous work done within this field, while also covering general theory of some processes.
- Chapter 3: Methodology and Implementation — This chapter explains in detail how the project was implemented to achieve results.
- Chapter 4: Results and Discussion — This chapter documents all the results achieved on the system and the step-by-step procedure of obtaining the results.
- Chapter 5: Conclusion — This chapter is a summary of project successes and potential future implementation.

1.4 Conclusion

The proposed system which was conceptually designed, was physically implemented with all the available hardware and software. A series of tests followed and the analysed results are seen in a later chapter. This chapter clarified the study objectives and an overall proposed system to complete the objective.

2 Chapter 2: Literature Review

This chapter contains a study of current techniques used within the industry to automate processes and related challenges. Certain techniques used within this study will also be outlined and their implementation benefits discussed. Software choices and uses will be explained in detail.

2.1 Manufacturing Systems

2.1.1 Introduction

Industrial automation in manufacturing is the use of “intelligent” machines required to complete a given task at the quickest possible rate. These machines are programmed to follow a fixed set of commands to complete their task. Every so often, adapting machines are created that can adapt to changes in requirements of the project at hand with minimal effort. These machines are extremely flexible, but may struggle to adapt correctly.

Various types of manufacturing systems exist, such as [2]:

- Dedicated Manufacturing System (DMS),
- Reconfigurable Manufacturing System (RMS) and
- Flexible Manufacturing Systems (FMS).

These systems each contain their own unique features, such as:

- Capacity,
- Functionality and
- Cost.

Each manufacturing system has certain advantages and disadvantages within these above-mentioned areas [3]. As the study presented is focusing mainly on an FMS, only this specific manufacturing system will be covered.

2.1.2 Flexible Manufacturing Systems

When it comes to Flexible Manufacturing Systems (FMS), companies are reluctant to make immediate changes to their current conventional systems. This could be due to a lack of knowledge about how to implement such systems [4].

Before we can detail the flexible manufacturing process, it is important to understand the definition of “flexibility” within an FMS context. Flexibility can be stated as a “range of motion” in its own context.

Multiple definitions exist for flexibility within the context of an FMS. A simple definition could be “The sensitivity of a manufacturing system to changes. The more flexible a system, the less sensitive to changes occurring to its environment it is” [5].

A second example of this definition, put forward by Ranky [6], is that an FMS deals with high-level distributed data processing and automated material flow using computer-controlled machines, assembly cells, industrial robots, inspection machines and so on, together with computer-integrated material handling and storage systems. Although the definitions presented are somewhat open to interpretation, research is constantly undertaken to improve the versatility of an FMS.

Characteristics of an FMS have also been broadly defined. What exactly is required within an FMS to categorize it as an FMS? There is no set number of characteristics required for a system to be classified as an FMS, but some general characteristics should be present according to different researchers and authors. These characteristics are as follows [4][7]:

- Flexible production in terms of volume and mix on the same assembly line.
- General-purpose CNC machines present in system.
- An automated material-handling system.
- An overall method of control that co-ordinates all subparts of the system.

The characteristics mentioned above are guidelines that are required for a system to be classified as an FMS. These characteristics are not limited to the aforementioned, many variations of the systems exist dependant on application and implementation.

The main reason an FMS can be classified as flexible is due to its capacity to process a variety of part styles simultaneously, while the production output can be changed in response to demand patterns.

2.1.3 Production Lines

Automated production lines, used within workspaces that have multiple unique assembly areas, are systems that compromise many stations that each independently contribute to the product in production. These sub-systems, inter-connected through mechanical transport systems, move products between workspaces in pre-defined sequences for correct product assembly.

Mechanical transport systems usually consist of conveyors that move materials from point to point within the system. They are usually preferred over more flexible transport systems such as Automated Guided Vehicles (AGVs), as parts do not have to wait to be picked up by an AGV and can be delivered instantly to the next station. Each station contains either automated processes such as drilling or milling, but could also contain manual labour operations such as inspections or manual placement of components dependant on the type of product being built [8].

There are benefits to using automated production lines within factories, such as:

- Less human intervention and effort required in production.
- Fewer production errors, better quality control.
- Better safety control.
- Lower production cost.
- Increased production output.

Automated production lines are many unique systems working together to complete an automated task.

2.1.4 Robotic Systems

Robotic systems, used for a variety of purposes within an automated assembly line are machines capable of carrying out complex tasks automatically. These tasks include packaging, pick and place, welding, soldering, drilling, milling, etc. They replace the human factor within the manufacturing process, while increasing efficient productivity and general overall output.

Generally, these computer-controlled machines each follow set programming routines dependant on their task. This ensures that each station contributes its specific piece to the overall product.

Several types of robotic systems exist for different task requirements. These requirements may include accuracy, the ability to carry out small complex tasks or to move a heavy payload. These factors along with cost determine what robotic system is selected.

2.1.4.1 End Effector

An end effector also known as End of Arm Tooling (EOAT) is the device placed at the end of a robotic arm. These devices, designed specifically for the robot application, allows the robot to interact with its environment and complete an assigned task. End effectors are usually designed in a manner that is not restrictive to the robot in terms of weight or size, using lightweight material [9]. Common interaction devices can be categorized into different groups, such as:

- Gripping devices — claws, jaws or any device that can physically grip the object.
- Injection devices — needles, pins or any device that penetrates the object.
- Suction devices — vacuum cups, magnets that use the object's surface to grip the object.
- Adhesion devices — devices that place a substance on the object.

There are many other devices that could be attached to the robotic wrist such as drills, welders, tool changers, brushes, sensors, screw drivers, spray guns, etc. It is all dependant on the application of the robotic system. Custom-built end effectors

are commonly used within newer industrial applications where traditional end effectors are insufficient.

Figure 2-1 shows vacuum cup end effectors which use pressurized air to create a vacuum that can pick up objects. Many different selections are available and depend on purpose, such as the size of the object to be picked up.

Figure 2-1 Professional Vacuum Cup End Effectors

These vacuum-based end effectors are easy to use and set up, with minimal hardware requirements or electrical interfacing required to pick up an object.

2.1.4.2 Conveyor Systems

Conveyor systems are automatic transportation systems that can move bulk materials from one point to another. They are used in automated industrial systems within different applications, depending on application requirements.

These systems come in unique shapes and sizes to fulfil different requirements such as heavy material transportation or small component transportation. They are vital to the automation processes, where manufacturing processes are implemented without any human intervention. Overall, this increases the production output of the system due to increased speed and efficiency.

Add-on modules for conveyor systems add to the modularity and flexibility of these systems, as pallets can be manipulated throughout the system based on certain requirements. An example is an automated rework system, where a pallet is scanned using a visual aid and directed to a certain conveyor system based on the outcome of the inspection. This is done using add-on modules such as deflectors or push diverters that direct routing within the system.

Selection of these systems is entirely dependent on application. Many different systems exist, with the main systems implementing mechanical belt-driven conveyors, vibrating conveyors, pneumatic conveyors and flexible conveyor systems.

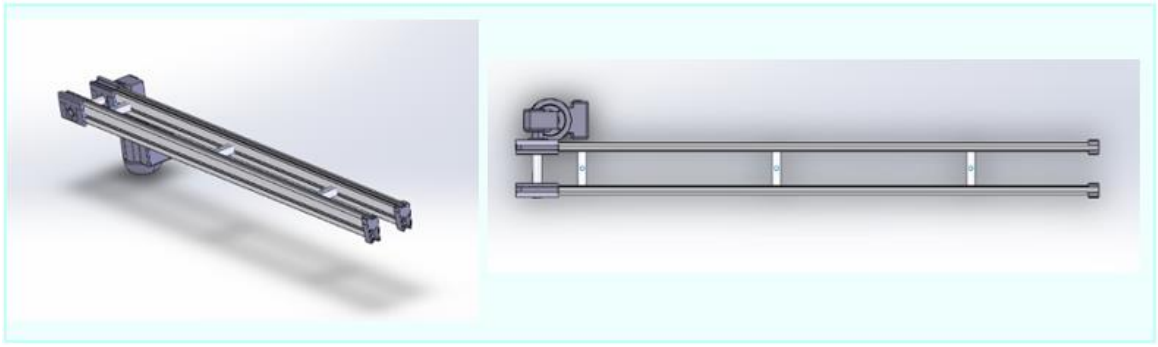


Figure 2-2 CAD Example of the TS1 conveyor by Rexroth

TS1 conveyor systems by Rexroth provide solutions for smaller payloads. These conveyors use anti-static pallet-based transfer systems to move small payloads around an assembly system. They are mobile and easily manoeuvred to apply new configurations in a short time frame.

2.1.4.3 Actuators

Actuators are mechanical devices that act as moving or controlling mechanisms on a production line. Different types of actuators include hydraulic, pneumatic, electric and mechanical.

Pneumatic actuators can control a system's flow by either opening or a closing a valve (for example) that would in turn do some physical work on a production line. They require a trigger mechanism and are frequently connected to a Programmable Logic Controller (PLC) for input signals. They are electronically triggered (for example to open a valve), but generally convert compressed air into a linear or rotary motion.

Figure 2-3 below shows two types of pneumatic actuators. The left is a linear motion piston that extends when triggered, while the right shows a Rexroth pneumatic stop gate used to control pallet flow within an assembly system.

Figure 2-3 Actuator Examples

In summary, actuators are vital within an industrial environment and widely used.

2.1.4.4 Sensor Systems

Sensors are vital within the automated assembly process. They provide important feedback on the real-time status of the system in monitoring applications while also providing feedback on the current assembly process (for example when a pallet passes a certain point) to the main controlling device. A variety of sensors is available within the automated production process, depending on the application.

Many factors are considered when selecting sensors to carry out application requirements such as accuracy, range, resolution, cost and repeatability. Repeatability is essential to ensure that readings returned by the sensor under the same conditions are consistent with minimal fluctuations.

Some of the main sensors within industrial assembly automation include proximity and displacement sensors (inductive, capacitance, photoelectric, etc.), while photoelectric can be used for a range of other applications such as detecting colours. Level sensors, vision sensors and pressure sensors are specific to assembly automation, but there are many more.

The data returned by the sensors enables the controller to make appropriate decisions in terms of the status of the system as well as the production line.

Figure 2-4 depicts the different types of sensors used within the assembly automation process.

Figure 2-4 Different Sensor Types within Assembly Automation

2.1.4.5 Logic Control Systems

Automation within industrial environments is accomplished using different methods based on requirements. One of the main controllers used within these environments is the Programmable Logic Controller (PLC), which controls all logistics involved in the automated solution. This includes (but is not limited to) all I/O devices such as sensors, conveyors, stop gates, LEDs, transverses, data sharing devices, etc. Complete flow control is determined by these devices and their relative inputs/outputs, assuming complete automated control over the system.

The logic control system is the “brain” of the system and requires precise programming to ensure error-free operation. It must also be designed with flexibility for use in an FMS. This includes creating flexible software that can adapt to changes as needed. In factories, it is not always possible to create fully functional software before the actual system is commissioned, due to specific customer needs. Thus, on-site adjustments are done during commissioning of the system.

Figure 2-5 shows two PLCs, each from leading brands in the automation industry, namely Siemens and Allen-Brandley.

Figure 2-5 Siemens and Allen-Bradley PLC controllers respectively

2.1.4.6 Machine Vision

Machine vision technology is an important tool for capturing useful information about a scene from a two-dimensional viewpoint. It provides automatic inspection of objects in applications such as robot guidance, process control and quality control. The end goal of machine vision would be to create a real-world model purely from images. Machine vision is achieved through image sensors that capture images, which are subsequently transferred to a processing unit for analysis [10].

Figure 2-6 below shows where machine vision fits into the world of image processing [11].

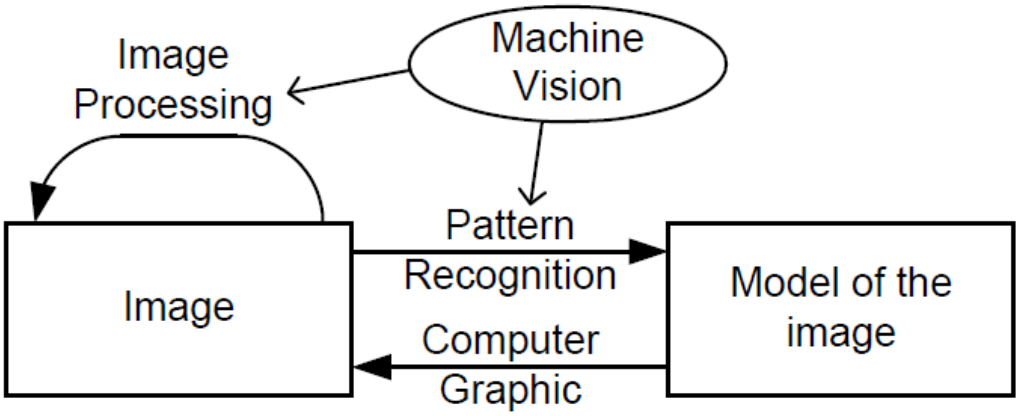


Figure 2-6 Machine Vision Processing

Computer vision and machine vision are usually used within the same context, but generally, computer vision refers to a more theoretical and algorithmic-based approach within image processing, while machine vision refers to the practical aspect, including image acquisition.

2.1.5 Digital Image Processing

Digital Image Processing is computerised manipulation of a digital image using certain techniques that alter and/or extract information from an image presented to accomplish certain tasks. These techniques add value to the image by looking for patterns and/or singling out certain interests within the image that can provide accurate information to the end user. The resulting data can be used to manipulate objects in the real world using physical devices, depending on the application area of these techniques.

Advantages of using computers to process images include [12]:

1. Flexibility and adaptability: Digital computers do not require any physical hardware modifications when solving different tasks, simplifying reprogramming.
2. Data Storage and Transmission: Digital data can be transferred easily between two points while being stored efficiently as new image-compression algorithms are developed.

Digital image processing starts with a digital image composed of elements called pixels, which are the smallest sample of an image. These pixels represent the brightness at one point and require two important operations, namely sampling and quantisation when being converted into a digital format from an analogue image source.

Figure 2-7 illustrates this process in block diagram format [12].

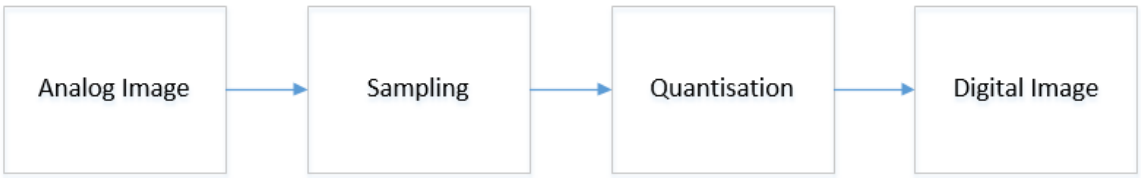


Figure 2-7 Analog to Digital Conversion

2.1.5.1 Acquiring an Image

Acquiring an image is the first step within the digital image processing environment. This process involves a camera sensor that captures the viewed scene and returns it as an image to a processing unit in digital format. Information is extracted from the digital image by the processing unit and converted into data values that can be understood by a computer system, resulting in decision-making based on these returned values. Images are usually captured as a single frame at a certain resolution, based on the requirements of the application. These images are sometimes also stored for later use if comparisons, for example, are required within the application.

2.1.5.2 Greyscale Images

Greyscale images are digital images where the value of every pixel correlates to the pixel's intensity information. In other words, each pixel's value is based on a shade of grey where the weakest intensity would correlate to the colour black while the strongest intensity would signify the colour white. The intensity of the pixel directly corresponds to the intensity of light captured by the camera sensor (within a greyscale-capturing sensor).

Greyscale images have certain depths that relate to the number representation for pixel intensity. For example, in an 8-bit grayscale image, each pixel would contain a number between 0–255 which correlates to that pixel's level of intensity. The depth can be increased for a more accurate representation of the shade but would require greater processing power and storage space to compute the data in the case of a 16- or 32-bit grayscale image, as the number representation is more substantial.

Figure 2-8 is a figured representation of the depths and their shading characteristics as the depth level is increased [13].

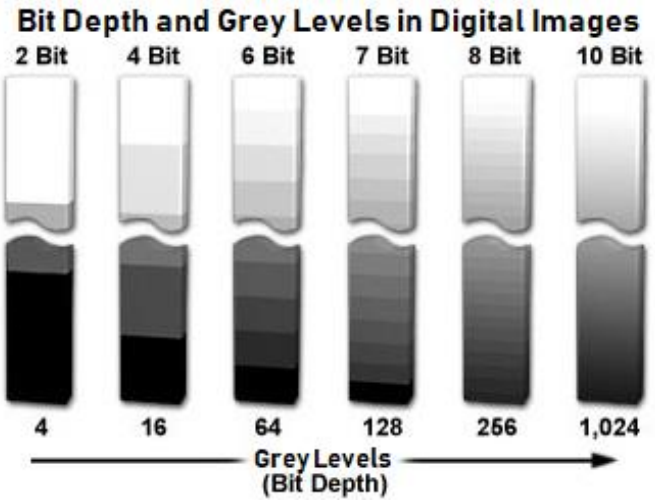


Figure 2-8 Bit Depths and Grey Levels

As seen in the figure above, as the depth increases, the amount of shades available increases proportionally. Dependant on application and resolution required, the correct bit-depth is selected.

2.1.5.3 Subtraction

Subtraction within image processing is the process whereby two separate images categorized into source and overlay image of the same size and pixel format are subtracted from each other to produce a new image. A simple procedure of completing this process is where each pixel is equal to the difference between the source and corresponding pixels, which results in a new image that is analysed.

2.1.5.4 Template Matching

Template matching is the process whereby a suitable template is moved over an image to determine at what positions in the image a precise match occurs. This operation reveals object matches and similar types of objects within the template and image being processed [14].

One of the simplest algorithms within template matching is the exhaustive template matching algorithm. This algorithm performs a complete scan of the source image and compares every pixel with the corresponding pixel of the template image. Although there have been more efficient algorithms presented (such as normalized

cross correlation template matching [15]), this process required minimal computation power with sufficient results required in this study.

2.1.5.5 Erosion

Erosion processing is a morphology image processing technique typically applied to binary images. The general concept is to erode boundaries of certain regions, resulting in the forefront of pixels taking preference.

Binary images consist of either one of two possible values: either a 0, which denotes background pixels, or a 1, which denotes foreground pixels. The erosion operator requires two inputs for the process to complete.

The image to be eroded usually uses a set of coordinate points (known as the structuring element) which determines the effect of the erosion on the image. In Figure 2-9, a clear example of the structuring element is presented.

1	1	1
1	1	1
1	1	1

Set of coordinate points =

{ (-1, -1), (0, -1), (1, -1),

(-1, 0), (0, 0), (1, 0),

(-1, 1), (0, 1), (1, 1) }

Figure 2-9 A 3x3 structuring element

A 3x3 matrix (for example) is used and applied to the image as the structuring element. This structured matrix would be superimposed over the input range at the centre of the input pixel (foreground pixel) coordinates. At this point, if every pixel in the structuring element corresponds with the image underneath, the input pixel

is left as is (this applies to neighbours of the input pixel in terms of the structuring element size).

If any of the corresponding pixels are background pixels, the input pixels are also converted to background pixels[16]. Figure 2-10 depicts the effect of a 3x3 square structuring element on a binary image.

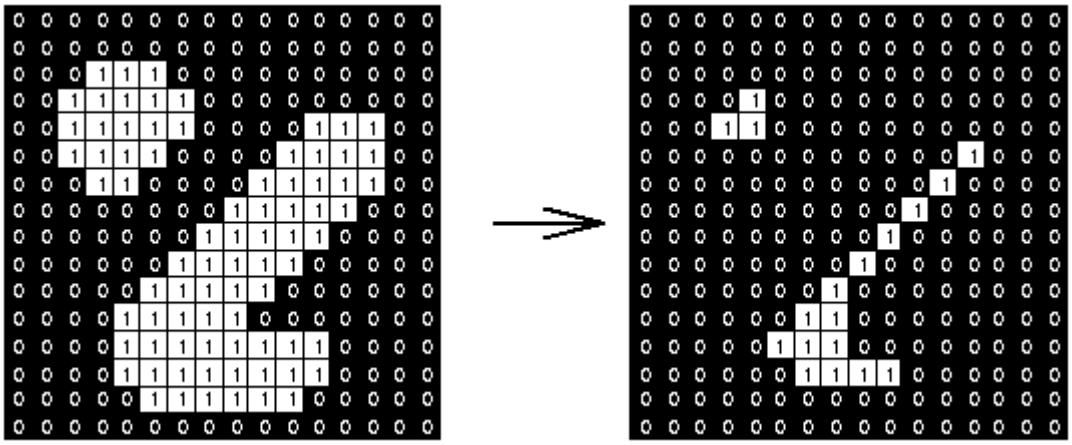


Figure 2-10 Effect of 3x3 structuring element on a binary image

As seen in the figure above, after the erosion technique is applied on the binary image, there is cleaner depiction of the binary image with the filter cleaning up surrounding pixels. This is particularly useful when an image contains excessive amounts of noise or static that must be removed.

2.1.5.6 Image Mass

When referring to centre of mass or centre of gravity on an everyday object, these two properties are different if the gravitational field is not uniform across the object. In image processing terms, centre of mass and centre of gravity in a binary image fall on the same location and is the same point. Centre of gravity is covered more thoroughly in section 2.2.5.8.

2.1.5.7 Image Moments

Within the field of image processing or computer vision, an image moment is a raw weighted average of the pixel intensities. These moments are used in a variety of applications and used to calculate items such as:

- Centre of Gravity
- Object Orientation
- Area of an Image, etc.

Images have two dimensions, so the formula requires two independent variables. A discrete way is used to describe each pixel in terms of moments. Equation 1 proves this concept on a mathematical level.

$$\mu_{m,n} = \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} (x - c_x)^m (y - c_y)^n f(x, y) \qquad \textit{Equation 1}$$

2.1.5.8 Centre of Gravity

The centre of gravity of an image is the average location of the weight of the image. Based on image moments, this function returns an X and Y coordinate that would equate the centroid of the image. The returned coordinate points are generally used as some sort of reference point within the application.

The centroid calculation is done by calculating the first moment of the image. All white pixels are added up using firstly the X coordinate and then the Y coordinate respectively (as seen in Equation 2 and 3 below). The sum of all these pixels is then divided by the overall pixel amount found within the zeroth moment. This results in the following Equations:

$$sum_x = \sum \sum x f(x, y) \qquad \textit{Equation 2}$$

$$sum_y = \sum \sum y f(x, y) \qquad \textit{Equation 3}$$

$$centroid = \left(\frac{\mu_{1,0}}{\mu_{0,0}}, \frac{\mu_{0,1}}{\mu_{0,0}}\right) \qquad \textit{Equation 4}$$

As seen in Equation 4, the resultant X and Y coordinate of the centroid is returned.

2.1.5.9 Area

When processing a binary image, the area is given by the number of pixels belonging to the object. Therefore, the matrix or pixel list is used to calculate the area of the object (or region of interest) simply by counting the number of pixels within the object [17].

2.1.6 Software

2.1.6.1 Microsoft Visual Studio

Microsoft Visual Studio is Microsoft's primary programming software used for software development in languages such C, C++ and C#. Free to use at express version levels, this software was used to create the operational server while also implementing the image processing required within this study.

2.1.6.2 KUKA WorkVisual

KUKA WorkVisual is KUKA's main programming environment when creating automated tasks for the KUKA Robotic System. All programming and configuration of the robotic system is done within this platform.

2.1.6.3 KUKA RSI

KUKA RobotSensorInterface(RSI) is the primary software environment used to set up a remote input into the KUKA Robotic System. Its primary purpose is to configure the remote inputs and outputs that will be used by the system, in a block diagram programming environment.

2.1.6.4 Total Integrated Automation Portal

The Total Integrated Automation (TIA) Portal provided by Siemens is their resident software to provide an end user with a complete range of digitalized automation services. This software, used to program all Siemens PLC's and HMI's in the user's preferred language, contains a range of features — from digital automation planning to integrated engineering.

2.2 Conclusion

This chapter covered a literature review to gain relevant knowledge of flexible manufacturing systems as well as hardware involved in making these systems operate. It also covered image processing techniques that could be used within these environments to allow visual aids to assist flexible manufacturing systems.

3 Chapter 3: Methodology and Implementation

This chapter contains a detailed overview of the techniques and design strategies implemented within the formation of this system. Various subsystems were deployed to complete the overall objective, including KUKA KR AGILUS sixx, KUKA RSI, KUKA WorkVisual, Siemens S7-1200 PLC System, a Basler AG Camera, and a C# Programming Environment using visual libraries such as Aforge.net and Accord.net.

These systems were all configured individually to ensure each sub-system was working satisfactorily, followed by combining and configuring all sub-systems to work as a single unit, which allowed the project objective to be carried out and analysed.

3.1 Hardware Systems

As many different hardware platforms was used throughout this system, it is necessary to detail each process individually as well as state each systems contribution to the overall design.

3.1.1 KUKA AGILUS R900 sixx

The KUKA AGILUS R900 sixx is an industrial robot intended for handling tools and fixtures, or for processing or transferring of components or products. The robot compromises an assortment of components. Many underlying components are not always present to the eye but are crucial to the operation of the manipulator.

Figure 3-1 below is a representation of the components and their uses [18].

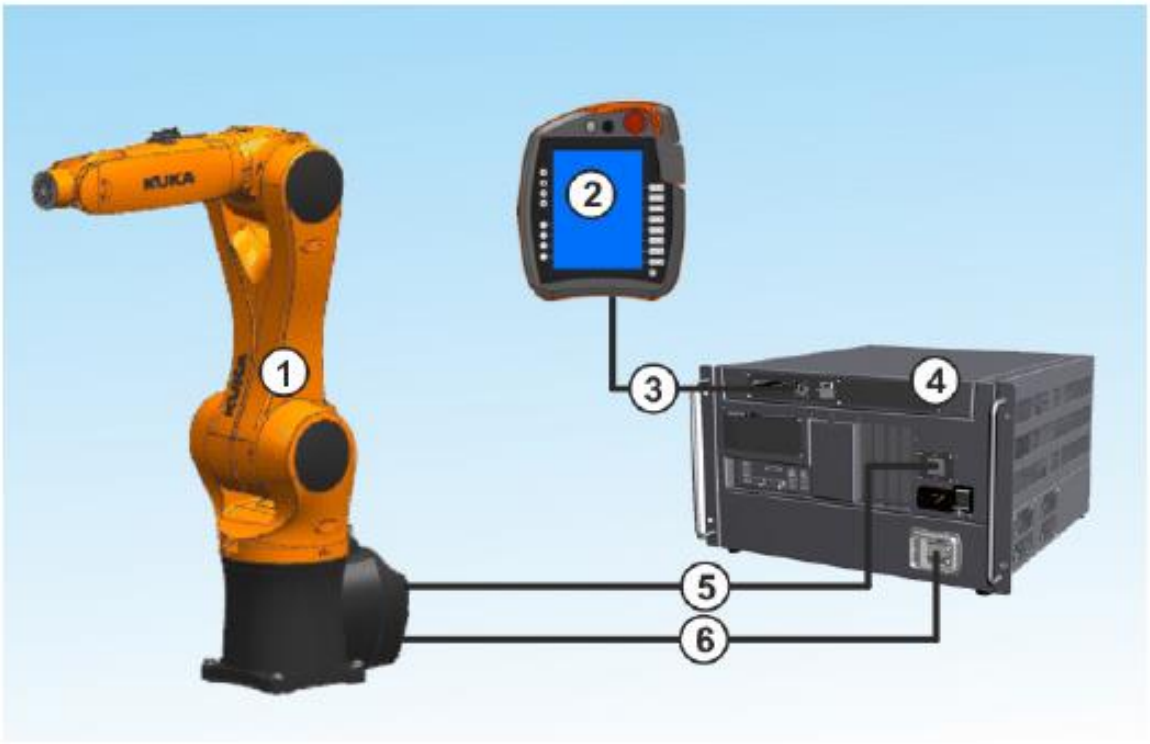


Figure 3-1 Components of an Industrial Robot

From the above figure, we can label each component as:

1. Manipulator
2. smartPAD teach pendant
3. Connecting cable, smartPAD
4. Robot controller
5. Connecting cable, data cable
6. Connecting cable, motor cable

The KUKA AGILUS R900 sixx was selected due to its wide industrial use and ease of integration with external systems. KUKA RSI was fully supported by this robot, which was essential to establish real-time control [18].

3.1.1.1 Tool calibration

Tool calibration is necessary to implement a coordinate system for the end effector. The end effector is mounted on the mounting flange at the head of the KUKA Robot and allows interaction with any objects as needed. The tool coordinate system has its origin at a user-defined point. This is called the TCP (Tool Centre Point). The TCP is generally situated at the working point of the tool.

Within this study, the tool or end effector was a suction gripper, which uses compressed air and a vacuum configuration to pick up items. This tool was selected due to its relative ease of use and ability to pick up lightweight payloads using a suction system.

Advantages of tool calibration include [19]:

- The tool can be moved in a straight line in the tool direction.
- The tool can be rotated about the TCP without changing position of the TCP.
- In program mode: The programmed velocity is maintained at the TCP along the path.

Multiple tool calibration methods are available, namely:

- XYZ 4-point method
- XYZ reference method
- ABC World method
- ABC 2-point method
- Manual Numeric Input

Although the base coordinate system was used throughout the program's main functionality, it was still required to calibrate the tool for testing purposes. The XYZ 4-point method was used for calibration due to undemanding requirements for the process, which can be seen in Figure 3-2 below [19].

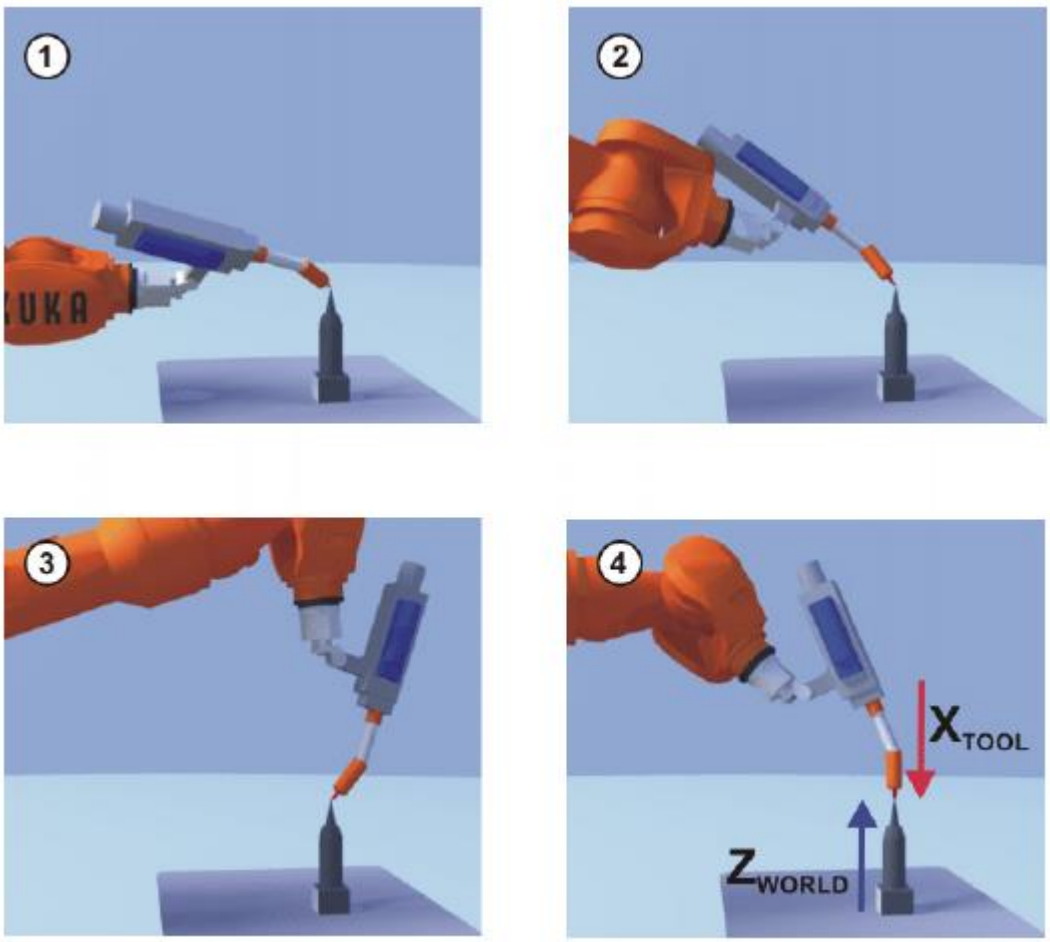


Figure 3-2 XYZ 4-point method example

An object with a sharp point was placed on top of the pallet situated on top of the conveyor. Using the end effector head as reference, the head was then manoeuvred around the point as seen in the steps above to calibrate the tool head to the system. This coordinate system enabled a new offset to be defined using the TCP for testing purposes, relating to the movement of the end effector attached to the KUKA Robot.

3.1.1.2 Robot bases and calibration

Configuring robot bases is essential to the robot's accuracy. Factors that can affect the robot in different working spaces include:

- Alignment of working surfaces.
- Angle of robot base.
- Movement of overall system during operation after fixing positions.

Two separate robot bases were configured for the robot for accuracy considerations on either side of the robot working space. For example, the alignment of floor surface was different to the alignment of the surface on the other end, causing the robot to "dip" into the Z axis when moving the robot on either the X or Y axis. The solution was to configure separate bases for either side of the working space, resulting in correct calibration for each side. The program would then automatically switch between these two separate bases when working on each side respectively.

Robot bases can be calibrated using two different methods namely:

- 3-point method
- Indirect method

The 3-point method was used for calibration of both bases as this was the most reliable option. The robot first moves to the origin of the base as well as two further points. These three points define the new base. An example of the process can be seen below in Figure 3-3, while Figure 3-4 shows the real-world implementation of this process [19].

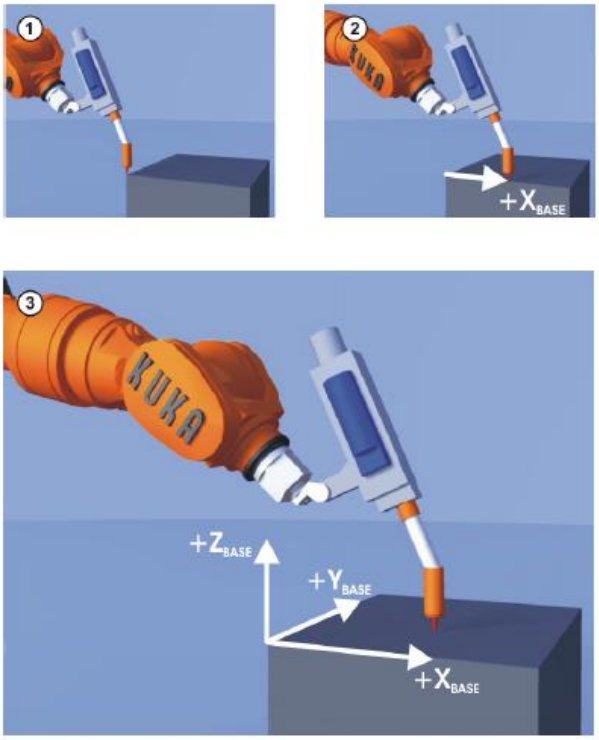


Figure 3-3 A CGI example of the calibration process

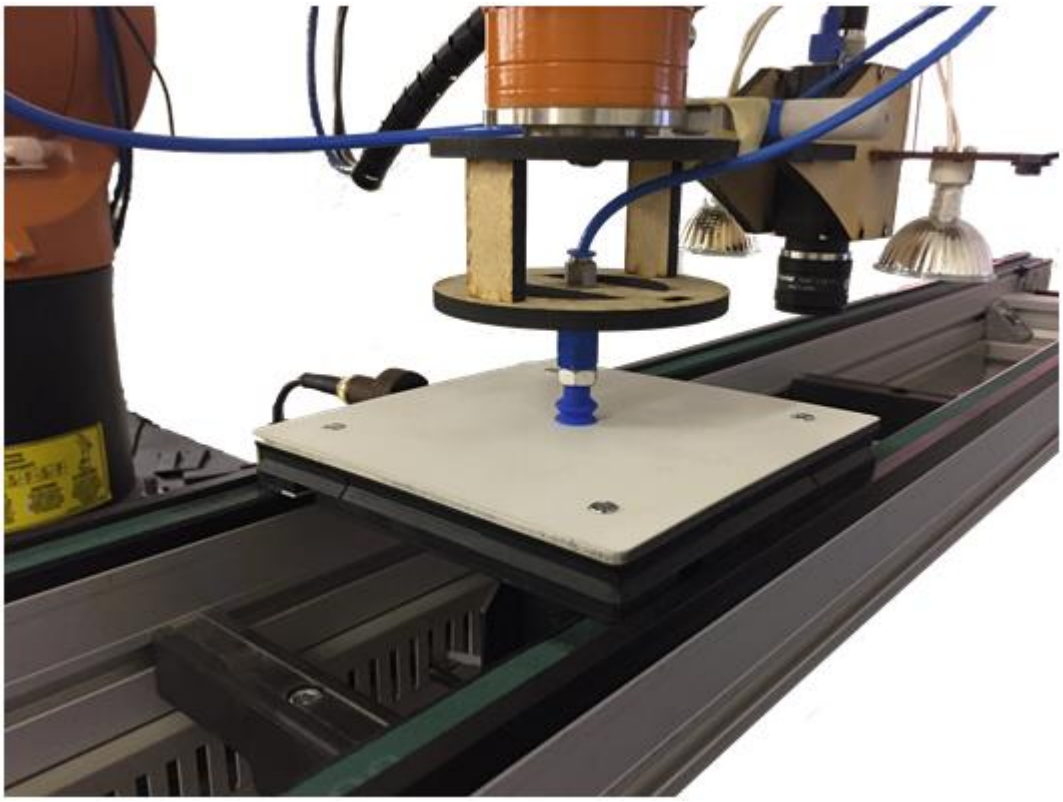


Figure 3-4 Calibration using the system

3.1.2 Siemens PLC

The Siemens S7-1200 PLC is a modern-day industrial controller primarily used within the automation environment. Focused on overall control of an automated system, the Siemens PLC is a cost-effective automated solution.

The PLC is used as an I/O device within this system, controlling conveyors and stop gates while also receiving sensor inputs. The device acted as a communication tool between the KUKA and C# environment, transferring shared variables between the respective systems. The PLC was configured with a raw Transmission Control Protocol/Internet Protocol (TCP/IP) connection configuration, which allowed direct communication via the TCP/IP to the .NET environment. All devices was configured using the standard TCP/IP configuration while transferring data strings that would be parsed into a certain function within the PLC program.

The PLC also contained feedback functionality to alert the program to changes (on top of the shared variables) happening on the system, such as pallets arriving at stop gates that would then trigger the next set of events within the main application. Based on the role that the PLC took in this study, it could be perceived as a partial gateway device to other devices in the system.

3.1.3 Conveyor Transport System

An automated conveyor movement system transported components around a closed loop conveyor, with a parallel conveyor to feed new designs and learn them. The Rexroth conveyor system was selected due to its lightweight payload capabilities and mobility. The system can easily be rearranged into different settings, adding to the flexibility of the system and allowing new configurations to be tested without labour intensive activities.

These conveyors also included stop gates and detection sensors which controlled pallet flow on the system. The detection sensors stopped the pallets containing the components for a picture to be taken and/or for a component to be collected. A secondary stop gate was placed where a queue of components was ready to be scanned and/or collected. This ensured that no secondary pallet made its way into the image processing vicinity while an ongoing operation was present.

The design conveyor also consisted of two stop gates with their respective detection sensors. The first stop gate was the design queuing area while the second was the design build area. If a pallet arrived at the first sensor while no pallet was present at the second sensor, it would automatically be fed through to the first gate. As mentioned above, this ensured no second pallet entered the current pallet's vision processing area.

3.2 Vision System

3.2.1 Basler AG Camera

The Basler AG Camera is an industrial camera that specializes in many application areas requiring cameras such as production, medical, traffic, transportation and retail. The camera acquired for the vision system in this project was specific to the production industry for robotics where detailed inspection is required.

The Basler scA1390-17gm is an Area Scan Camera with the Sony ICX267 CCD sensor that can deliver 17 frames per second at a 1.4 MP resolution. The camera is a GigE, connected via an Ethernet connection.

A 12 mm lens was attached to this camera, which required constant changes to focus and zoom which was very dependent on the KUKA's current location above the pallet. To overcome this, two fixed points were established for the camera positioning (the camera was attached to the KUKA).

3.2.2 Camera Calibration

Camera calibration was done using Basler Pylon, which is used to capture live feeds of the camera. The lens was adjusted using focus as well as light input. Auto adjust was deactivated from the beginning to use manual calibration instead (light was insufficient on both sides of the conveyor, so a constant source was used for both sides). Manual exposure was set on the camera with auto exposure deactivated because of both artificial light (such as ceiling lights) and natural light entering the project area.

Fixed lighting conditions (from the LEDs attached to the system) and fixed positioning of the camera at the system's two image-capturing reference points enabled manual exposure and focus setting. This process was done through the Basler Pylon software while a component was placed on the pallet and optimal exposure and focus was determined and set.

If the lighting environment changed, manual recalibration was required to avoid under or over exposure on the capturing device resulting in extremely inaccurate results when doing image processing on the pallets.

3.2.3 Lighting Environment

A fixed lighting environment (using the LEDs attached to the system) was used due to changing lighting conditions — natural lighting and artificial lighting such as the 50 Hz light flicker found in AC light sources (such as ceiling lights) — within the test area.

A fixed DC-supplied lighting environment was implemented using standard DC halogen lighting. These lights were controlled using the PLC which triggered the lighting when a picture scan was conducted. Once the system successfully initiated the lighting environment, it would proceed to acquire the image requested by the system. On successful return of the image, the lighting was deactivated and system operation would continue.

This fixed lighting environment allowed consistent picture results, removing the need for constant manual camera adjustment in terms of exposure. Figure 3-5 below shows how the lights were attached to the KUKA end effector with the camera situated between the two.

Figure 3-5 Example of the fixed lighting setup attached to the end effector

3.3 Software System

A number of unique software packages and programming languages was used throughout the system to complete the overall objective.

3.3.1 KUKA RSI

KUKA RSI better known as KUKA Robotic Sensor Interface is a real-time communication platform used to communicate directly to the KUKA Robotic System from a remote input through a specific communication channel. The data read by the KUKA Robotic System from the remote data source allows real-time adjustment of the KUKA Robotic Systems position, depending on the data received from this remote module. This process allows a remote system full control of the KUKA Robotic Systems position, which enables a whole new range of applications when using the KUKA Robotic System.

Robot Sensor Interface is an add-on technology with the following functions:

- Data exchange between robot controller and sensor system.
- Data exchange via Ethernet or the I/O system of the robot controller.
- Cyclical signal processing and evaluation at the sensor cycle rate.
- Influence on the robot motion or program execution by processing sensor signals.
- Configuration of the signal flow (RSI Context) with the graphical editor RSI visual.
- Library with RSI objects for configuration of the signal flow (RSI context).
- Online visualization of the RSI signals (RSI monitor) [20].

Data exchange via Ethernet was the selected communication preference.

In Figure 3-6, the fundamental principles regarding Ethernet exchange are depicted [20].

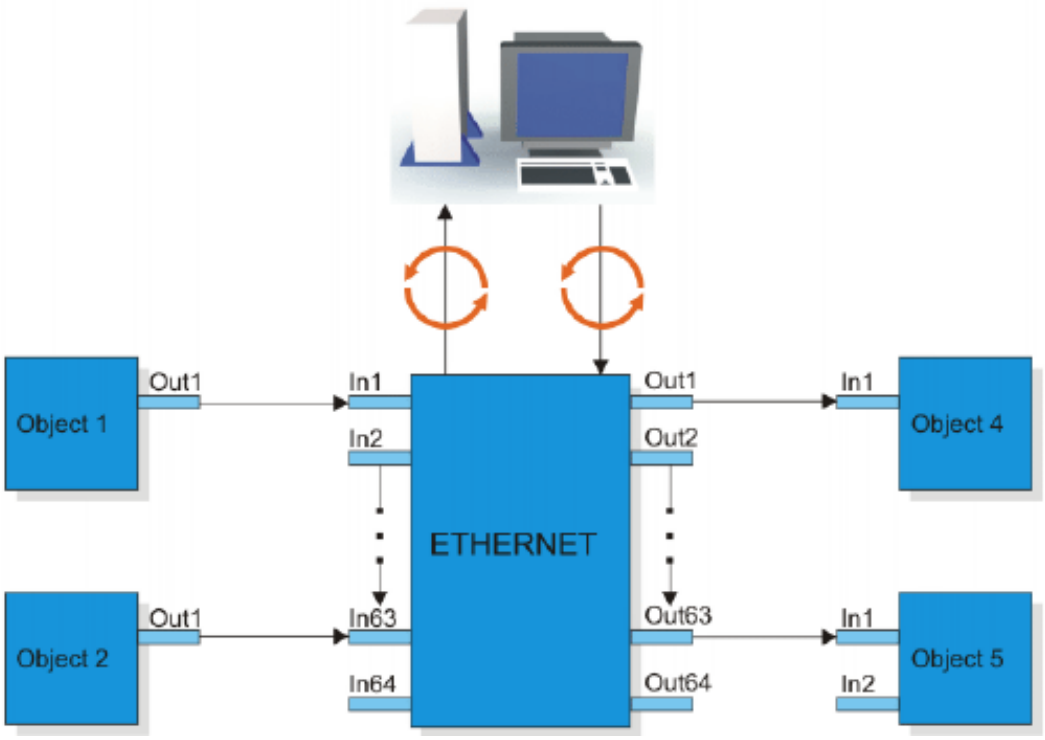


Figure 3-6 Data Exchange via Ethernet

RSI allows for continual influence over the robot motion by means of sensor data.

An object relates to a certain property or variable that will be updated through the RSI system. A certain object will be configured as either an inbound object or an outbound object. This principle is clearly visible under section 3.4.1.3, which illustrates the RSI setup utilizing a visual aid with its inbound and outbound objects.

Different modes of correction can be configured, namely:

1. Motion-suppressed sensor correction
    a. Axis-angle correction, absolute or relative
    b. Cartesian correction, absolute or relative
2. Sensor-guided motion
    a. Axis-angle correction, absolute or relative
    b. Cartesian correction, absolute or relative

3.3.1.1 Sequence of events

The RSI system follows a strict sequence of events that must take place correctly to establish and maintain a successful and consistent connection. This ensures the system and end user's safety during real-time control.

When the RSI system is activated within the KRL, a channel is prepared for sending data to the sensor system via User Datagram Protocol/Internet Protocol (UDP/IP). The robot controller initiates the data exchange with a data packet and transfers data packets to the sensor system at the sensor cycle rate. The sensor system should be constantly listening for a data connection from the robot controller to avoid possible errors on the KRL program. When a packet is received at the sensor system end, the system must respond to the robot controller with its own unique packet. A general overview is seen below in Figure 3-7 [20].

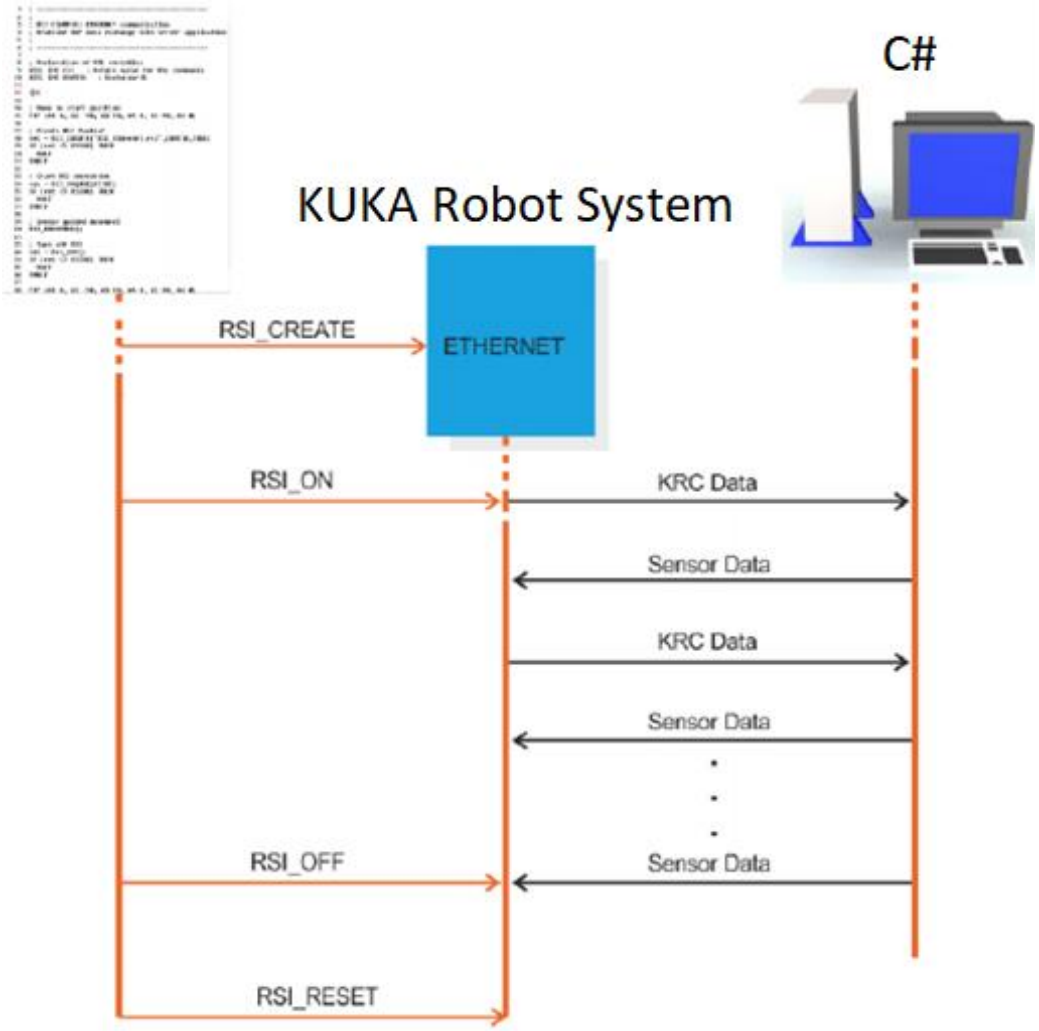


Figure 3-7 Data Exchange via Ethernet (Sequence)

A data packet received by the sensor system must be answered within the sensor cycle rate of the robot controller, meaning a packet must be returned to the robot controller within the same sensor cycle in which it was sent. If any data exchange requirements are not met during the transfer period, the communication channel is closed and will have to be reinitialized.

After the initialization, there is constant packet transfer between the robot controller and the sensor system, even if there is no current correction. The channel remains open until either an error occurs, or a closing packet is conveyed to terminate the connection.

Every data packet transfer is encoded with information as set up in RSIVisualShell interface. RSIVisualShell will be discussed in section 3.4.1.4 in detail regarding its uses and requirements for RSI set up on the KUKA Robotic System.

3.3.1.2 Sensor Cycle Rate

The sensor cycle rate is the rate at which the signal processing is calculated, i.e. the specified cycle time in which all work should be completed to and from the sensor system. The RSI system supports two sensor processing rates, namely Input Processing Output (IPO) mode which sets the processing rate to 12ms. A faster processing rate is available using IPO_FAST mode which fixes the processing rate to 4ms. All data must be processed within this cycle rate for it to be a valid operation.

An IPOC cycle is the overall cycle keyword used within the RSI context to generate a time stamp for both systems (robot controller and end sensor system) to know which cycle is currently being processed. The IPOC time stamp value is checked during every cycle. The data packet is only valid if the time stamp corresponds to the time stamp previously sent. This ensures that the correct operation is carried out by the RSI with regards to the packet received.

3.3.1.3 RSIVisualShell

RSIVisualShell is the programming environment used to configure an RSI connection to an external device such as a sensor or other external input. Programming is done in an offline environment where I/O variables, correction settings and feedback variables are defined. This setup creates various files which must be transferred to the KUKA Robot via the WorkVisual environment as the files are directly embedded into the KUKA file system.

The settings defined within these files must match the setup that will flow in and out of the Ethernet connection to allow error-free operation. Manual manipulation is also required for certain RSI files, which are not part of the automatic processing.

Within RSIVisualShell, different blocks are added from a toolbox. These blocks all have different purposes and create different outcomes.

Firstly, an Ethernet connection must be set up using an Ethernet block. This will enable the Ethernet channel while also allowing inputs to be set up. These inputs will be fed into the system (from an external source) while the outputs will be fed from the Ethernet out to separate blocks to make changes to the KUKA system.

POSCORR can be set up to allow correction on the system using the currently selected coordinate system (BASE).

AXISCORR is more specific and is used to correct a specific axis on the robot such as A6.

Limitations are also set up within this environment to only allow a certain amount of movement at a given time. This is done for safety reasons and prevents over-correction issues.

Figure 3-8 below shows the section that returns data from the robot to C# as well as the Ethernet block to set up Ethernet connectivity to and from the KUKA Robotic System. Basically, it returns the status of internal variables as configured within the KRL program such as current location, motor current, etc.

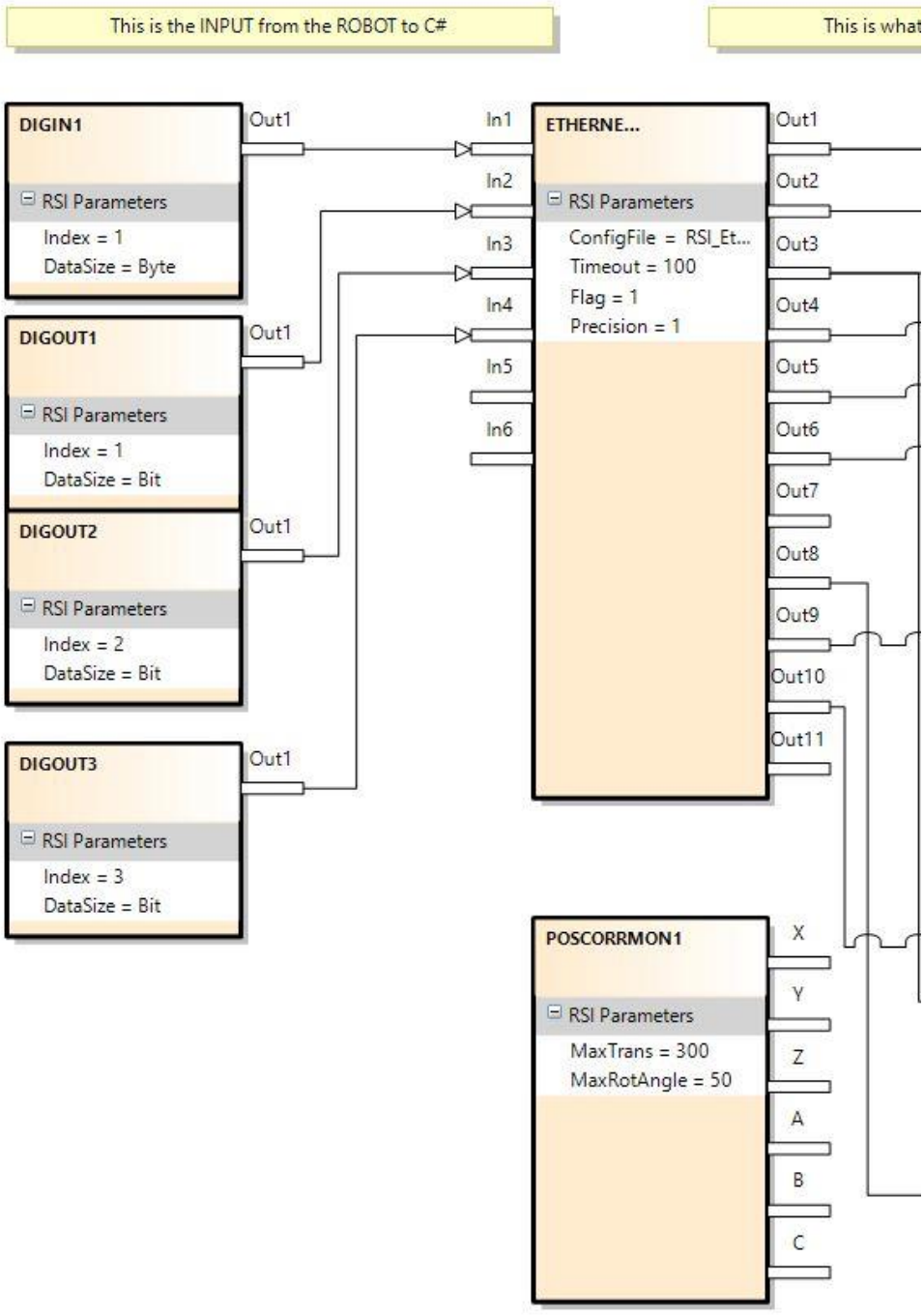


Figure 3-8 RSIVisualShell block diagram section A

As seen above, blocks are created and linked together using virtual wires which mimics the flow of data within the system. Outputs can also be mapped to local

variables on the KUKA Robot System in their respective format, enabling data sharing as well as data correction across these devices.

Figure 3-9 shows the Ethernet connected directly to the POSCORR block which allows for position correction of the KUKA Robotic System (real-time control). This allows for correction on the XYZ coordinate systems.

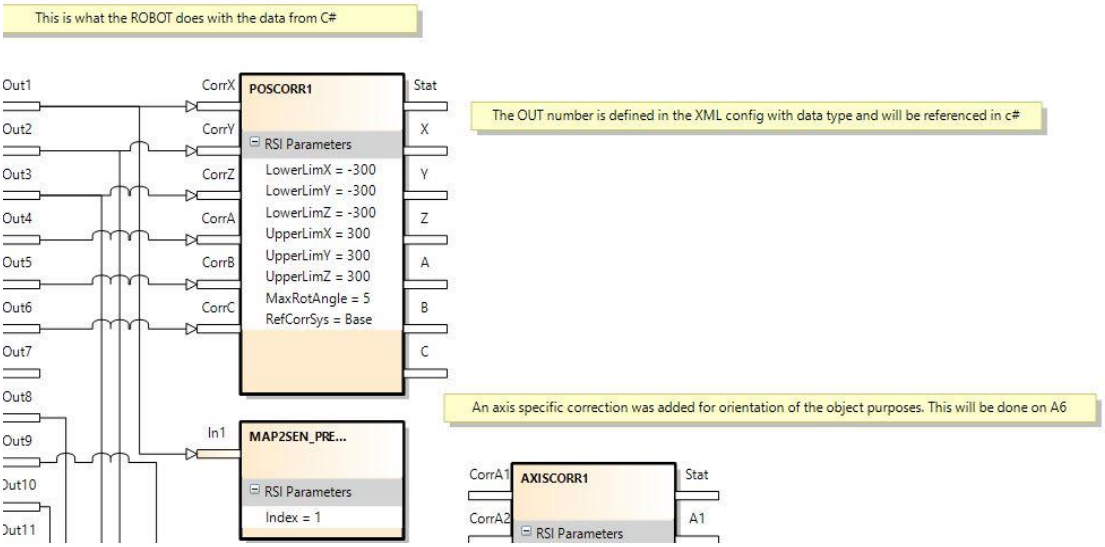


Figure 3-9 RSIVisualShell block diagram section B

A stop block is required to break RSI operation within the KRL program. This is important because if the connection from the server side is interrupted before this block executes, the KRL program (running in the KUKA Robot) will return an error and will have to be re-initialized. This can be seen in Figure 3-10.

As mentioned previously, the AXISCORR is for specific axis correction (which was required within this study). Figure 3-10 depicts this within the RSIVisualShell setup.

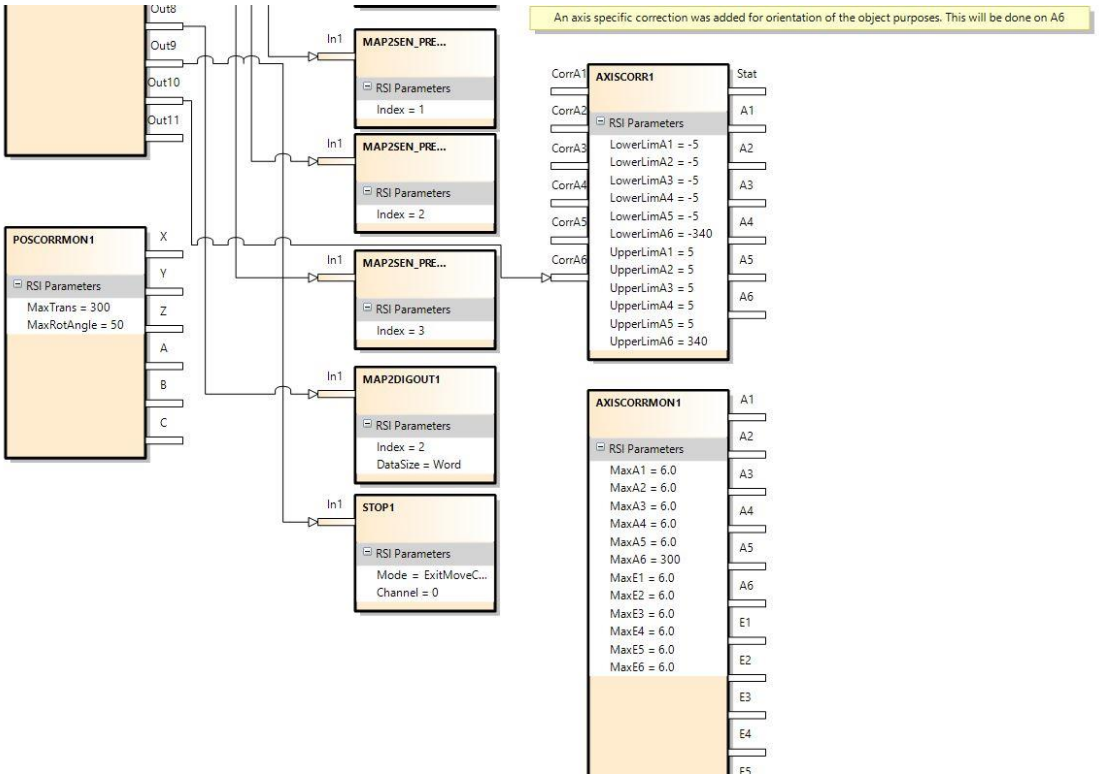


Figure 3-10 RSIVisualShell section C

Manual editing is required for embedded files on the KUKA Robotic system, such as the RSI_EthernetConfig XML file which houses settings such as IP address and PORT number. These settings must correspond to the host settings set up on the PC for the connection to be established.

Figure 3-11 below displays the setup of the XML configuration done on the KUKA Robotic System. This file setup corresponds to the setup within RSIVisualShell which notifies the RSI system which data to expect when sending and receiving data over Ethernet to the sensor system. These elements are basically how the data is “split” and sorted when receiving or sending.

```
<ROOT>
   <CONFIG>
      <IP_NUMBER>192.168.2.11</IP_NUMBER>   <!-- IP-number of the external socket -->
      <PORT>6020</PORT>                  <!-- Port-number of the external socket -->
      <SENTYPE>luketaMasters</SENTYPE>        <!-- The name of your system send in <Sen Type="" > -->
      <ONLYSEND>FALSE</ONLYSEND>       <!-- TRUE means the client don't expect answers. Do not send anything to robot -->
   </CONFIG>
   <!-- RSI Data: TYPE=  "BOOL", "STRING", "LONG", "DOUBLE" -->
   <!-- INDX= "INTERNAL" switch on internal read values. Needed by DEF_... -->
   <!-- INDX= "nmb" Input/Output index of RSI-Object / Maximum of RSI Channels: 64  -->
   <!-- HOLDON="1", set this output index of RSI Object to the last value  -->
   <!-- DEF_Delay count the late packages and send it back to server  -->
   <!-- DEF_Tech: .T = advance .C = main run / .T1 advance set function generator 1 -->

   <SEND>
      <ELEMENTS>
         <ELEMENT TAG="DEF_RIst" TYPE="DOUBLE" INDX="INTERNAL" />
         <ELEMENT TAG="DEF_RSol" TYPE="DOUBLE" INDX="INTERNAL" />
   <ELEMENT TAG="DEF_AIPos" TYPE="DOUBLE" INDX="INTERNAL" />
         <ELEMENT TAG="DEF_ASPos" TYPE="DOUBLE" INDX="INTERNAL" />
         <ELEMENT TAG="DEF_MACur" TYPE="DOUBLE" INDX="INTERNAL" />
         <ELEMENT TAG="DEF_Delay" TYPE="LONG" INDX="INTERNAL" />
         <ELEMENT TAG="DEF_Tech.C1" TYPE="DOUBLE" INDX="INTERNAL" />
         <ELEMENT TAG="DiL" TYPE="LONG" INDX="1" />
         <ELEMENT TAG="Digout.o1" TYPE="BOOL" INDX="2" />
         <ELEMENT TAG="Digout.o2" TYPE="BOOL" INDX="3" />
         <ELEMENT TAG="Digout.o3" TYPE="BOOL" INDX="4" />
      </ELEMENTS>
   </SEND>
   <RECEIVE>
      <ELEMENTS>
         <ELEMENT TAG="DEF_EStr" TYPE="STRING" INDX="INTERNAL" />
         <ELEMENT TAG="DEF_Tech.T2" TYPE="DOUBLE" INDX="INTERNAL" HOLDON="0" />
         <ELEMENT TAG="RKorr.X" TYPE="DOUBLE" INDX="1" HOLDON="1" />
         <ELEMENT TAG="RKorr.Y" TYPE="DOUBLE" INDX="2" HOLDON="1" />
         <ELEMENT TAG="RKorr.Z" TYPE="DOUBLE" INDX="3" HOLDON="1" />
         <ELEMENT TAG="RKorr.A" TYPE="DOUBLE" INDX="4" HOLDON="1" />
         <ELEMENT TAG="RKorr.B" TYPE="DOUBLE" INDX="5" HOLDON="1" />
         <ELEMENT TAG="RKorr.C" TYPE="DOUBLE" INDX="6" HOLDON="1" />
         <ELEMENT TAG="FREE" TYPE="LONG" INDX="7" HOLDON="1" />
         <ELEMENT TAG="DiO" TYPE="LONG" INDX="8" HOLDON="1" />
   <ELEMENT TAG="Stop" TYPE="BOOL" INDX="9" HOLDON="1" />
   <ELEMENT TAG="A6" TYPE="DOUBLE" INDX="10" HOLDON="1" />
      </ELEMENTS>
   </RECEIVE>
</ROOT>
```

Figure 3-11 XML file configuration for RSI communication

The above setup contains non-vital information passed between the RSI system and the sensor system such as motor current, etc. which is only used for display purposes, with the vital information being R.Korr {variable} which is used to implement correction on the KUKA Robotic System [21].

3.3.1.4 Virtual Addressing

Virtual Network Address can be defined as having more than one internet protocol address assigned to the same physical adapter interface. The virtual internet protocol does not actually correspond to a physical interface. This creates two separate networks that can communicate independently from one another over the same physical space.

Virtual Network Addressing is used to establish an RSI Ethernet Connection between the KUKA and remote system. The KUKA Line Interface Port x66 on the KUKA is used as the RSI Interface in addition to the WorkVisual Interface. For both applications to function correctly, virtual addressing must be implemented on both the KUKA and the remote sensor endpoint; in this case, the PC implementing the vision processing techniques.

As seen in Figure 3-12 below, the physical adapter consists of multiple Internet Protocol addresses, these being virtual addresses.

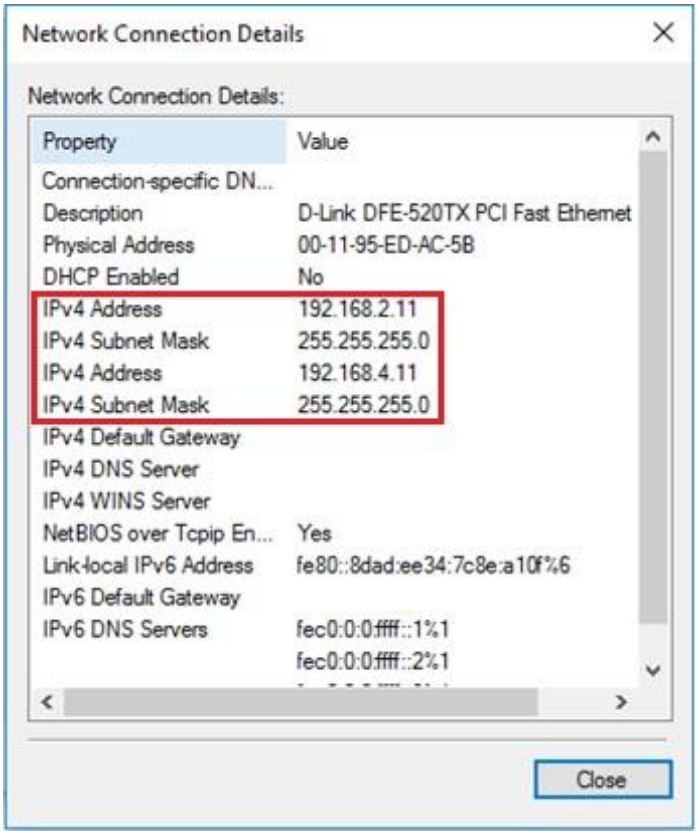


Figure 3-12 Virtual Addresses assigned to one physical adapter

3.3.1.5 XML Packet Transfer

The Extensible Mark-up Language is a standard for creating machine and human readable documents in the form of a specified structure tree. Packet transfer over Ethernet is bi-directional, enabling a user to set up return parameters, such as robot position coordinates or any other returnable parameter. This creates a closed feedback loop where stability of the system is maintained in real-time due to the feedback from the robot controller. These parameters are additionally used to ensure the robot is on the desired path.

Figure 3-13 below shows how data is moved into an XML string, which is then added to the overall XML string packet and sent to the RSI system on the KUKA Robotic System.

```
"<RKorr X=\"{0:F4}\" Y=\"{1:F4}\" Z=\"{2:F4}\" A=\"{3:F4}\" B=\"{4:F4}\" C=\"{5:F4}\" />\r\n"
, new object[] { this.cor.X, this.cor.Y, this.cor.Z, this.cor.A, this.cor.B, this.cor.C });
```

Figure 3-13 XML String creation in C#

In the above snippet, we can see each axis being defined within the XML format (X, Y, Z, A, B, C) while also linking them to a new object, which is their correction amount per IPOC cycle. During every IPOC cycle, a string like the one depicted above will be transferred to the robot controller. Depending on the correction required, the objects referencing each axis will contain a float value, which is the distance the KUKA is required to move that axis within that specific IPOC cycle.

The KUKA Robot can only move a certain distance within one IPOC cycle. These values should be kept low to ensure smooth motion of the KUKA Robot when correction is taking place. If the value is greater than a certain threshold, a jerking motion could result.

3.3.1.6 Integration with C#

Integration with the .NET framework environment was vital for establishing successful communication. The created application would need to respond to the KUKA within a given IPOC cycle while encoding the data for movement requirements as well as decoding the feedback data.

The server was created over the UDP protocol which has no validation whether a packet reaches its intended destination. This can be advantageous if speed is a major factor in the project. In this case, speed is a major factor as the program would need to reply within the IPOC cycle to maintain the connection. It can also be disadvantageous, because if packets are lost en route, there is no way of recovering them or even validating that they were ever received.

Due to the environment and the limited number of switches between the endpoint and the KUKA Robotic System, this problem never occurred. In a more complex network setup where network hardware is under constant strain, this could become a problem. Migrating to the TCP protocol to ensure packet arrival and upgrading network infrastructure would be a feasible solution.

A virtual addressing scheme was created to allow a clean channel for data transfer. This channel is only used by the RSI transfer system while any other system data goes through a different virtual addressing scheme.

On every string received by the KUKA Robot (in the XML scheme), the system first decodes the string to generate an answer. The IPOC string information is first extracted and saved for generating the answer required by the system. Next, the system extracts the feedback data sent to the system regarding the status of each axis (A1-A6). Each axis data is shown in terms of actual position, target position and motor current. This is useful when ensuring that the KUKA Robot's actual position and target position corresponds.

Next, an answer is generated with an updated IPOC inserted into the XML packet and the RSI movement values added if any movement is required during the cycle. The RSI connection can be open, transferring packets with zero correction adjustment, meaning the system will maintain its current position. Other data such as variables can also be transferred with this scheme if it was set up within

RSIVisualShell beforehand. The generated XML packet must match the RSIVisualShell configuration exactly or communication will fail.

Figure 3-14 below shows the process of how the XML strings are handled and generated. Once a newly generated packet is sent, the system then returns to a “waiting” step where it waits to receive a packet from the KUKA Robotic System. This process is then repeated until the real-time correction is completed.

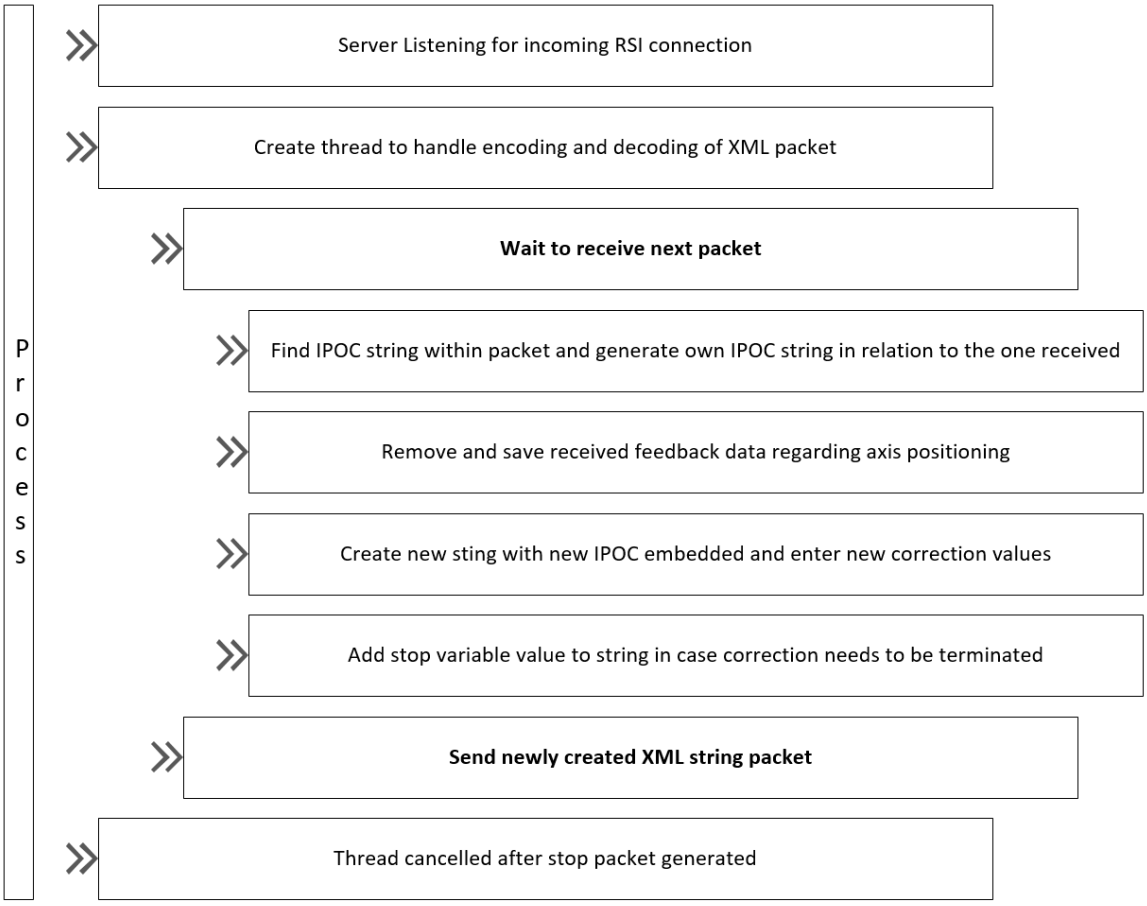


Figure 3-14 XML packet handling

3.3.2 PLC Programming

3.3.2.1 Overview

A PLC controller was integrated into the system to control the conveyors and pallets, to relay sensor data and enable data sharing between the KUKA Robot and the .NET environment. PLCs have multiple programming languages, such as:

- Ladder Diagram (LAD)
- Function Block Diagram (FBD)
- Structured Control Language (SCL)

Ladder Diagram (LAD) was selected as the suitable language to program the PLC due to prior expertise in this language.

The programming was divided into six categories to maintain structure throughout the main block function. These categories were:

- TCP Control Network
- Comparison Network
- Sensor and Air Feedback Network
- KUKA Position Control Network
- TCP Queue System Network
- RSI & Light Control Network

Raw byte data was passed between the two systems with lookup tables on either side used to decode the function required by the main system. The system also feeds data back to the main program to confirm command execution. This ensures proper sync between the actual events and the events recorded on the system.

3.3.2.2 TCP Control Network

A TCP/IP network was set up on the PLC to allow TCP/IP data to be transferred between the PLC and any other TCP/IP capable device. This was cost and time efficient as there was no need to implement a middle man such as an OLE for the Process Control (OPC) server to connect the PLC to environments such as the .NET framework. A simpler system was created instead to enable faster and more reliable data exchange between the two devices.

The Ladder Diagram includes TCP\IP function blocks which can implement packet transfer on the Ethernet connector. Figure 3-15 depicts the SEND and RECEIVE blocks used for communication on the PLC.

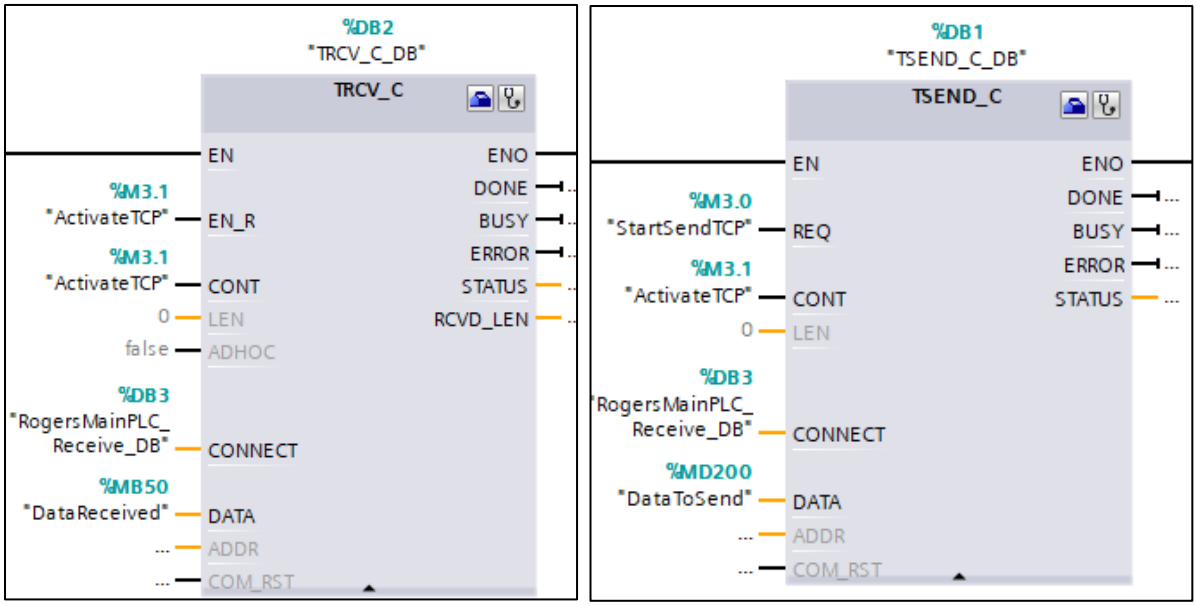


Figure 3-15 Send and Receive Block within the PLC Network

As seen above, separate SEND and RECEIVE blocks are configured independently. These blocks control all direct communication to the .NET platform, which housed the main software for the system.

3.3.2.3 Comparison Network

The comparison network would decode data received through a lookup table. This comparison network acted as the controlling network for most of the devices within the project such as the conveyors, stop gates, KUKA positioning, etc. Once a command was decoded, it would activate the appropriate sub-network to carry out the function required. The system would then provide feedback to the main server, indicating that the process was completed successfully.

Figure 3-16 below is an example of how the data flow progressed. Once the data received was compared to the fixed value, it would activate the subsequent process while also activating the feedback command to confirm it was received and processed.

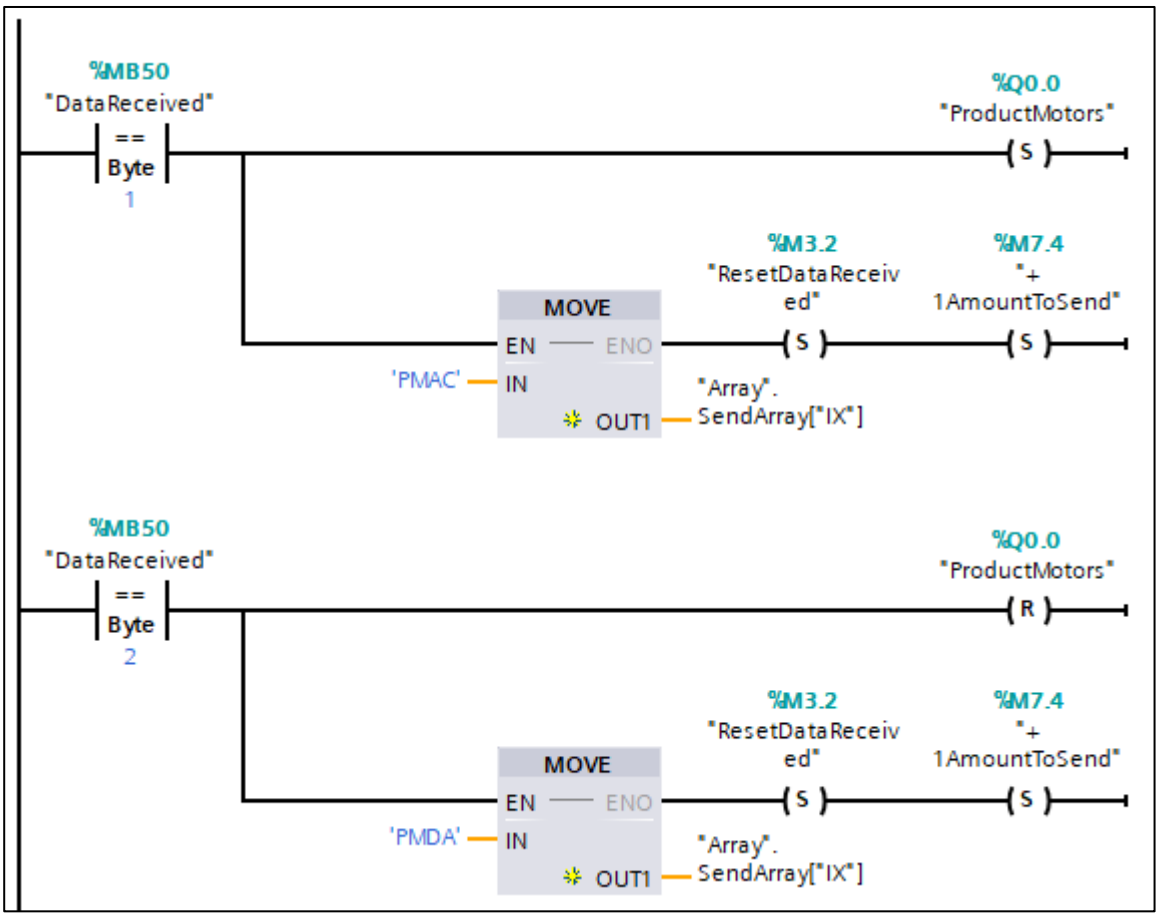


Figure 3-16 Example of how data is decoded for correction function

3.3.2.4 TCP Queue System Network

A TCP queue system was set up on the PLC to ensure no data loss due to simultaneous feedback being relayed to the main program. This occurred when two (or more) pallets arrived at a detection sensor on the conveyor within 50ms of one another. The result was that the data being prepared to be relayed back was overwritten and lost, leading to a discrepancy between actual events on the conveyor system and the events recorded on the main program. This caused errors during flow operations as the system was waiting for a pallet to arrive which had in fact already arrived, although the data had been lost.

The solution to this problem was to set up a TCP queue system where data to be relayed back to the main program would first be added to a queue stack. The waiting data would be added to an array and triggered as soon as the channel became available. If an operation is currently relaying data back, the system would increase a tracking variable indicating that more data is to be relayed once the current operation is complete. The tracking variable would be compared to the current cycle number and if the two variables differed, the system would fetch the new data and re-execute the relaying function. This process ensured all data would successfully reach the main program, allowing identical events to be showcased as they occurred on the conveyor system.

The steps taken to create this system are depicted below in Figure 3-17.

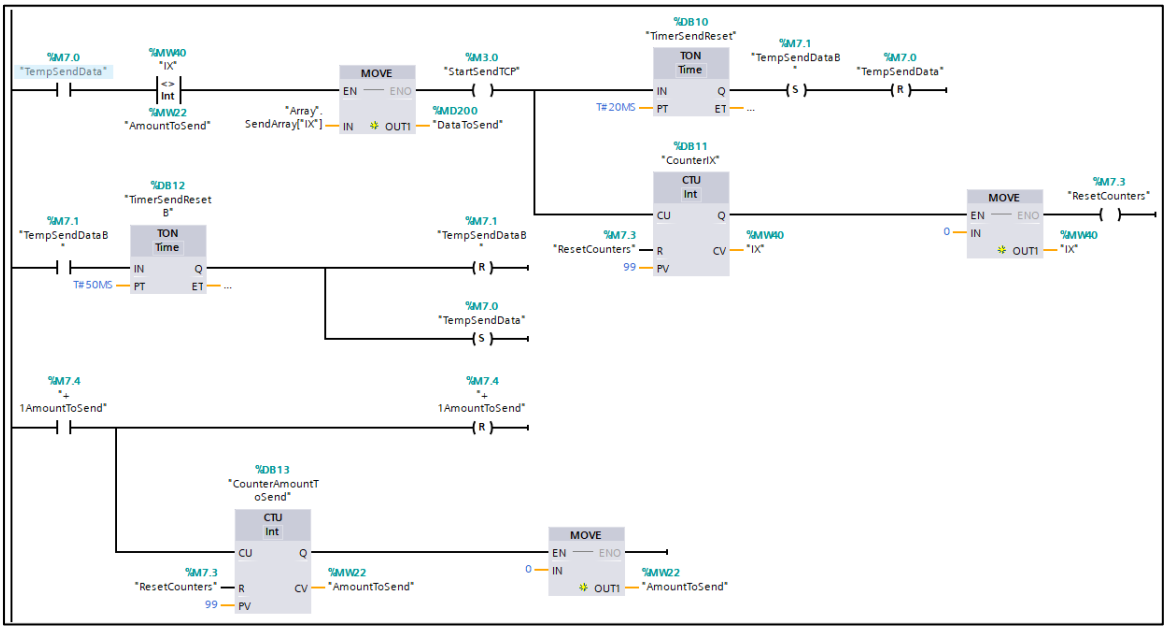


Figure 3-17 TCP queue network system in ladder logic

The system may seem complicated as seen in Figure 3-17, but it is surprisingly simple logic. When data must be sent, it increases the “AmountToSend” after moving the data into the next available position on the data array. At this point, the comparison between IX (the value that keeps track of the data in the array) and the amount to send differ, meaning data has been moved into the array and is waiting to be sent. This value (IX) keeps track of the send queue using a counter that increases the value by one every time the system sends new data. The system will then start the send process and wait 20ms, after which it will set subsequent variables to enable a further delay of 50ms. After the initial 20ms delay, the IX counter is increased and if the value equates to 99, it resets back to 0 and the data [that has already been sent] is overwritten from position 0.

The array will store the previous 99 commands sent back to the main program to streamline troubleshooting. AmountToSend is also increased as the program has cycled the send function an additional time, meaning data was sent back to the main program. The process then repeats itself with the initial IX value being compared again to the AmountToSend variable.

3.3.3 C# Programming

3.3.3.1 Object Orientated Programming

Object Orientated Programming (OOP) is a concept that organises data around objects rather than “actions”. This concept was used programmatically to store all data regarding each “product” and “design” and how they were inherited from one another. The concept makes use of procedures (known as methods) that allows data to be easily modified and linked based on the “product” or “design” at hand. Once a product is identified, an object is created containing all the relevant data. When a new design is studied, the specific product’s object data is collected, and a new object is created based on the previously collected data.

In Figure 3-18, we can see how the Design Object hierarchy is linked to each other.

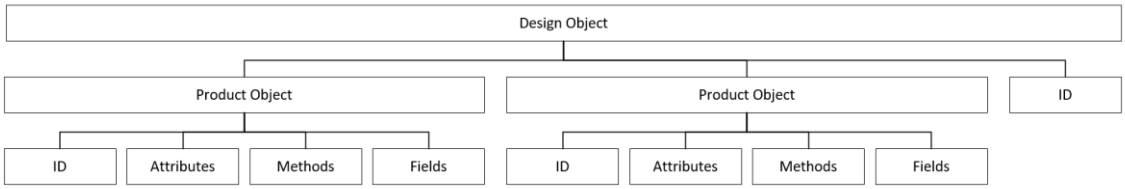


Figure 3-18 A Design Object Hierarchy

The figure above depicts the hierarchy structure of how the data is composed into an object. This process is repeated for each subsequent design learnt and saved for later use.

3.3.3.2 Overall System Flow

3.3.3.2.1 Introduction

Overall System flow was vital for successful operation of the system. This section covers how the system controls the flow of each sub-application to successfully learn a design, followed by rebuilding it using available product components.

3.3.3.2.2 Methods Used

The system design was programmatically split up into five different sections:

- Communication Processing
- Image Processing
- RSI Processing
- Automation Processing
- Overall Processing

Each section contained all methods required to complete the functions specified in the section heading. For example, communication processing would contain all functions and methods required for successful communication to be established with all devices within the project.

Each section directly communicated with each other through global variables within the status section to ensure that no function was carried out without its prerequisite functions being met. This ensured that, for example, the system would not attempt to acquire an image before the KUKA Robotic System had physically moved to its intended destination.

In Figure 3-19, we can see how the communication flow was established while updating status variables concurrently.

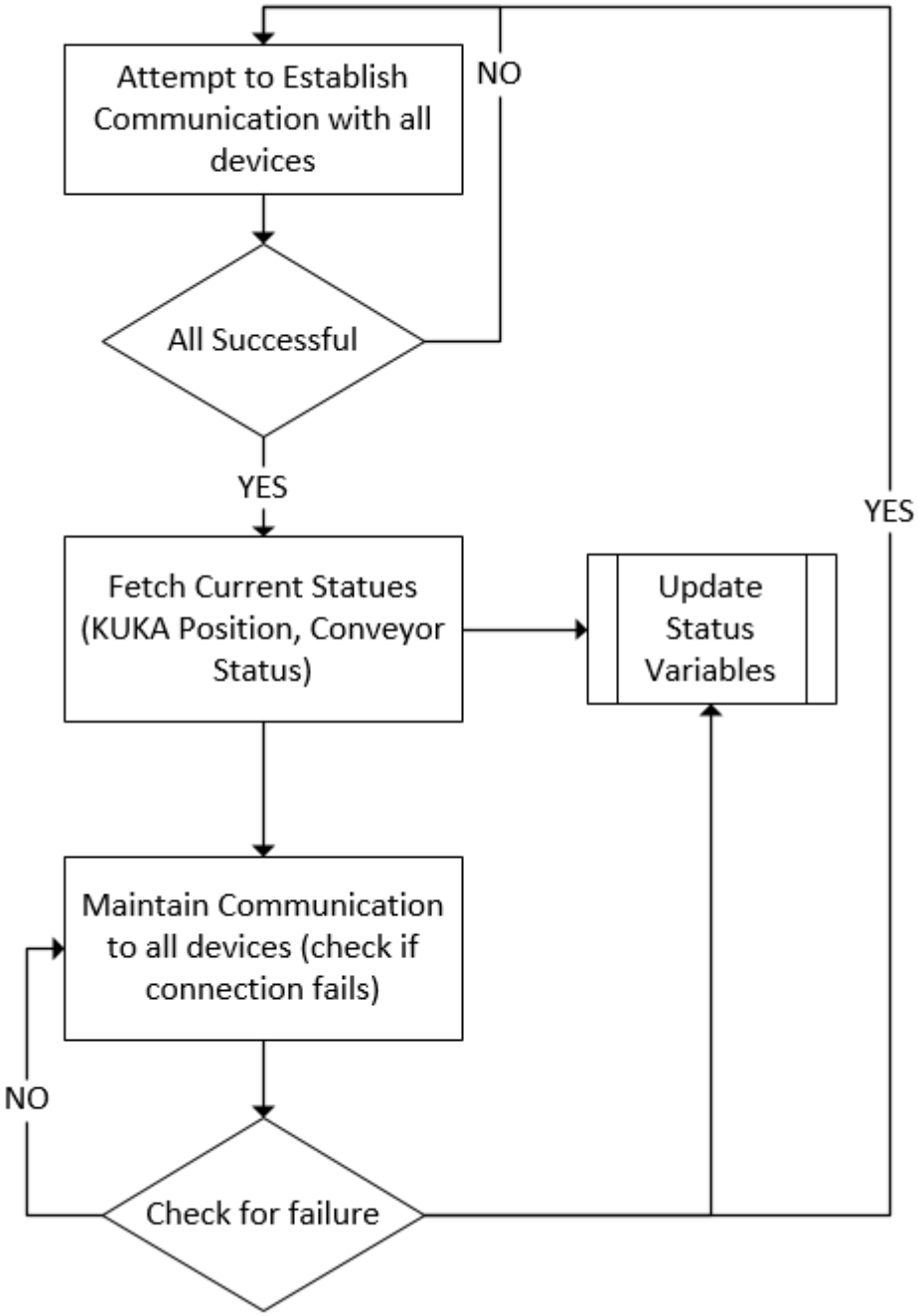


Figure 3-19 Communication Processing Flow Diagram

If at any point the communication between any devices fails, the system comes to a complete emergency stop while the system attempts to re-establish communication with the device.

In Figure 3-20, the image-processing flow diagram is depicted, which returns a processed image from the Basler Camera system on board the end effector.

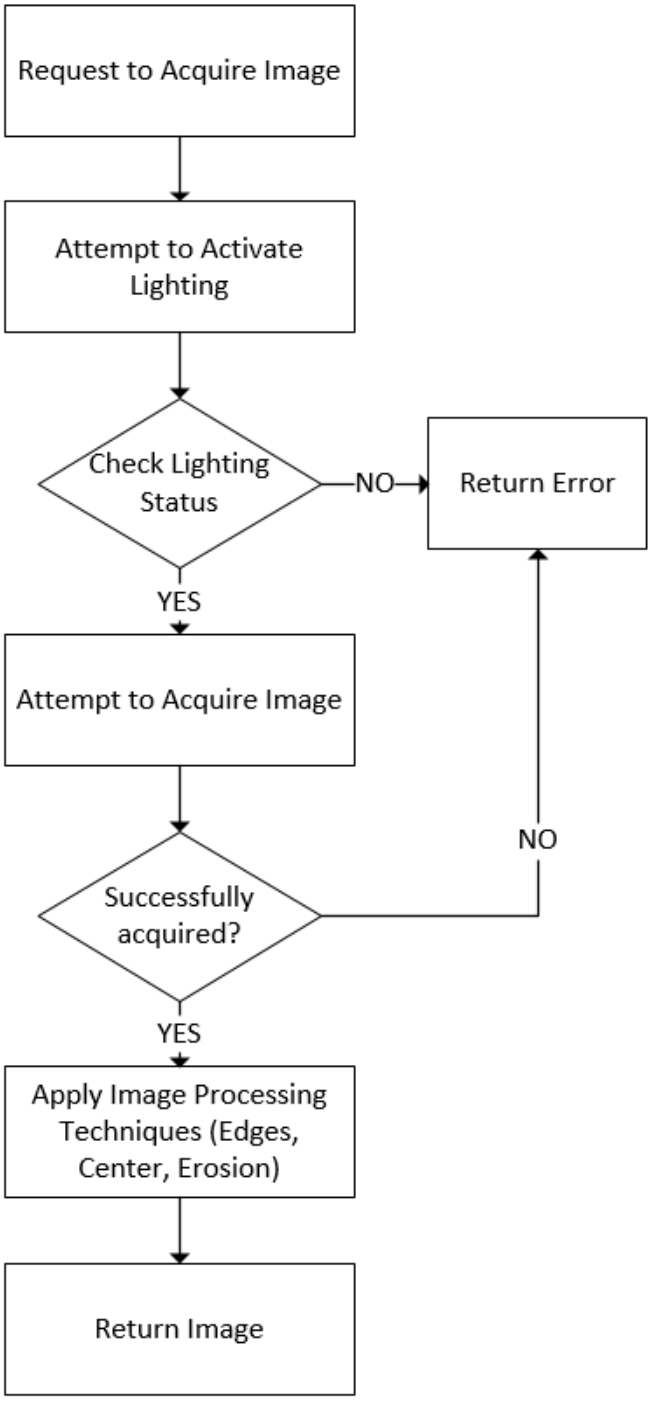


Figure 3-20 Image processing Flow Diagram

Figure 3-21 shows the process that the system follows when the server requests to take over real-time control of the Robotic System.

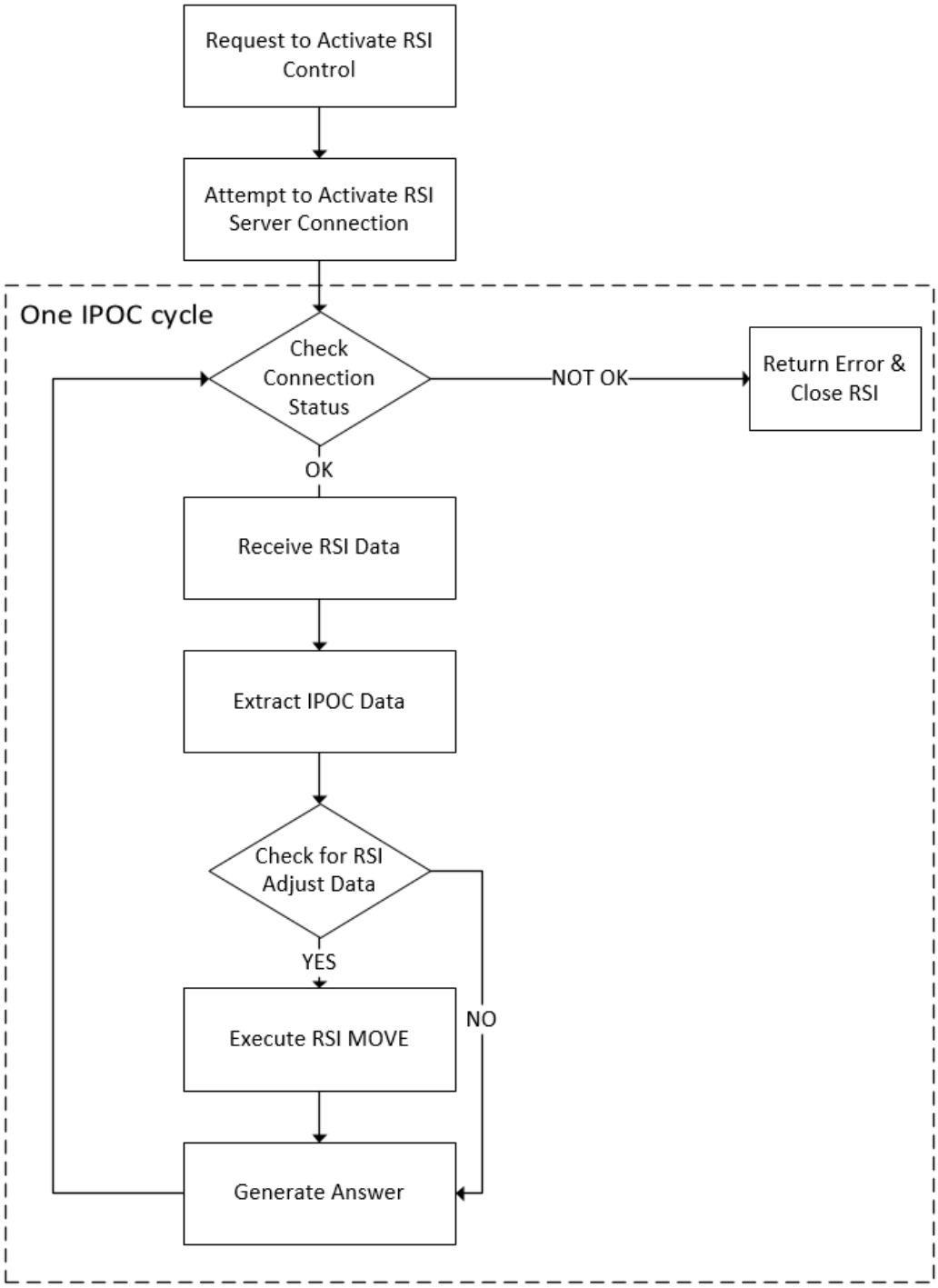


Figure 3-21 RSI Processing Flow Diagram

If any of the systems do not reply within the sensor cycle rate, the connection is automatically closed, and the Robotic System comes to a complete halt.

Figure 3-22 shows the process followed when the system is required to automate a build.

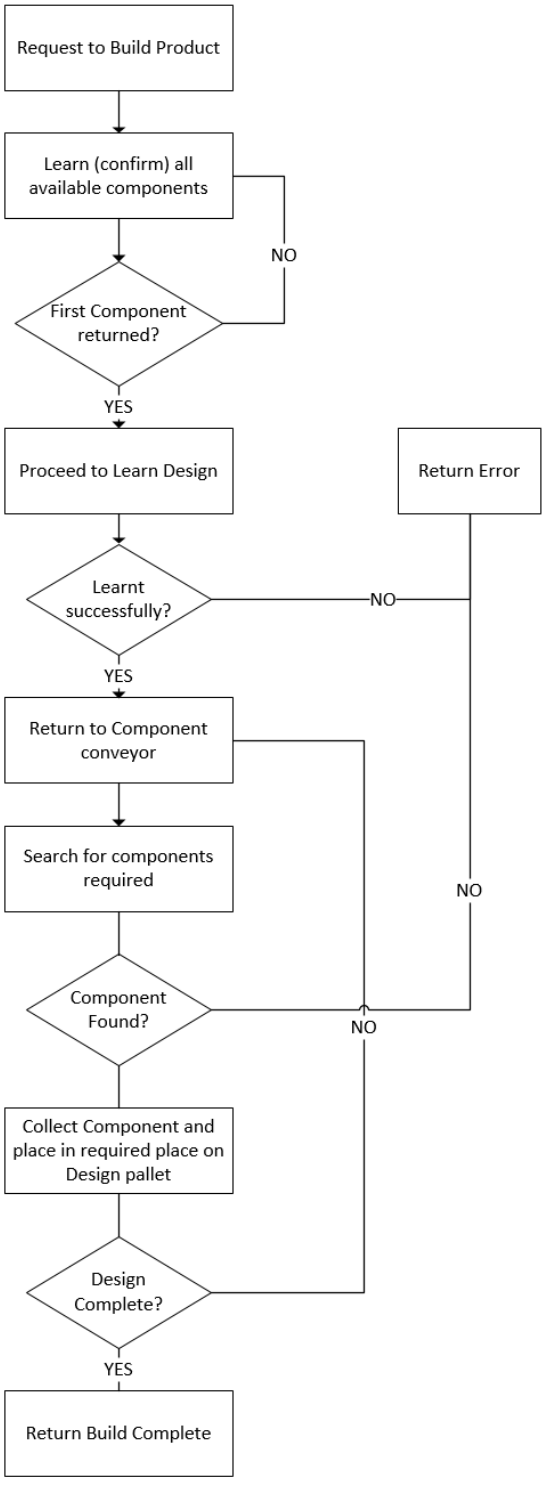


Figure 3-22 Automation Processing Flow Diagram

It should be noted that the above diagram only depicts major processes. Minor processes do exist, such as pallet management using stop gates, sensors, etc.

Figure 3-23 shows how the software is integrated and cooperates to achieve the final project goal.

Overall Processing Control would submit requests to execute functions while the Status Variables acted as the middle-man for sharing data between each section.

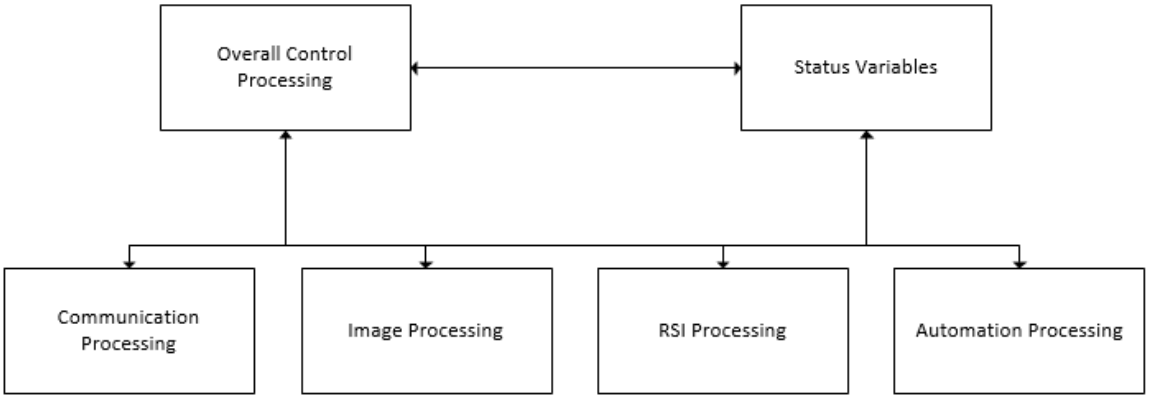


Figure 3-23 Overall Flow within Software

3.3.4 Image Processing Techniques

Various image processing applications and techniques were used throughout the project. This included various open source libraries available for free online as well as software in high-end image processing cameras such as the Basler AG Industrial Camera.

3.3.4.1 Basler Integration

The Basler AG camera is integrated into the C# environment using the Basler Pylon plugin provided by Basler AG. This acquires pixel-based data which is stored in a byte array. It must be manually generated into a static image using a pixel algorithm loop. This is done by taking each pixel and setting the RGB value for each position given by the inputted width and height of the image. Figure 3-24 and 3-25 below shows how the image is stitched together.

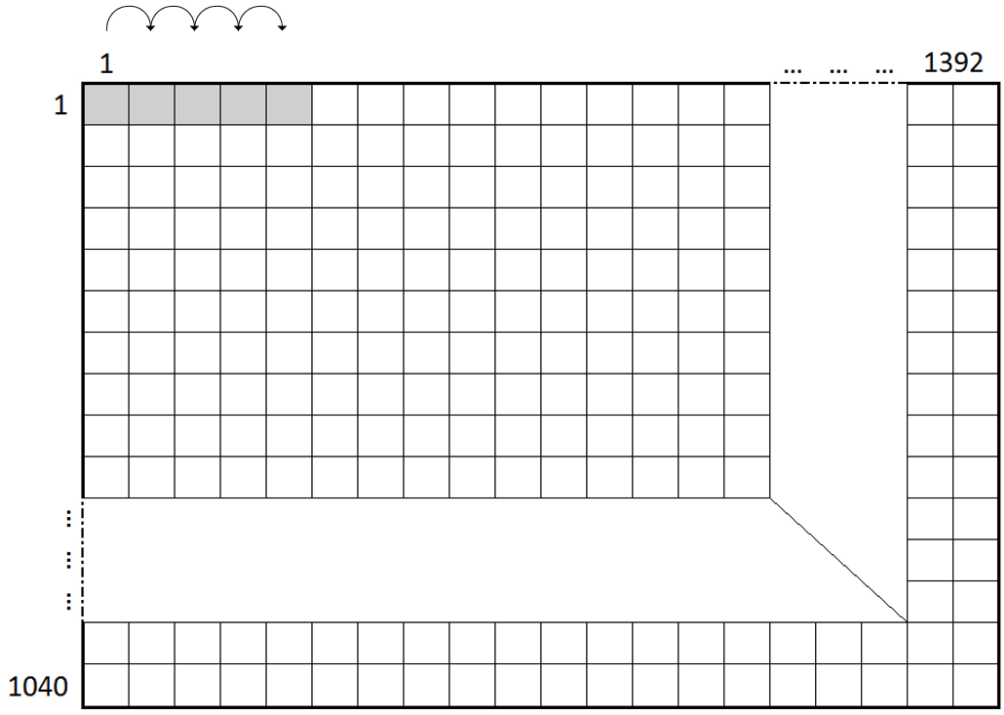


Figure 3-24 Pixel Creation A

The pixel algorithm first places the byte data [RGB] in the specified horizontal manner (as above) and then downwards, layer by layer. For every 1392 interactions of the horizontal placement, the process undergoes one vertical downward shift. The process then repeats itself as seen below.

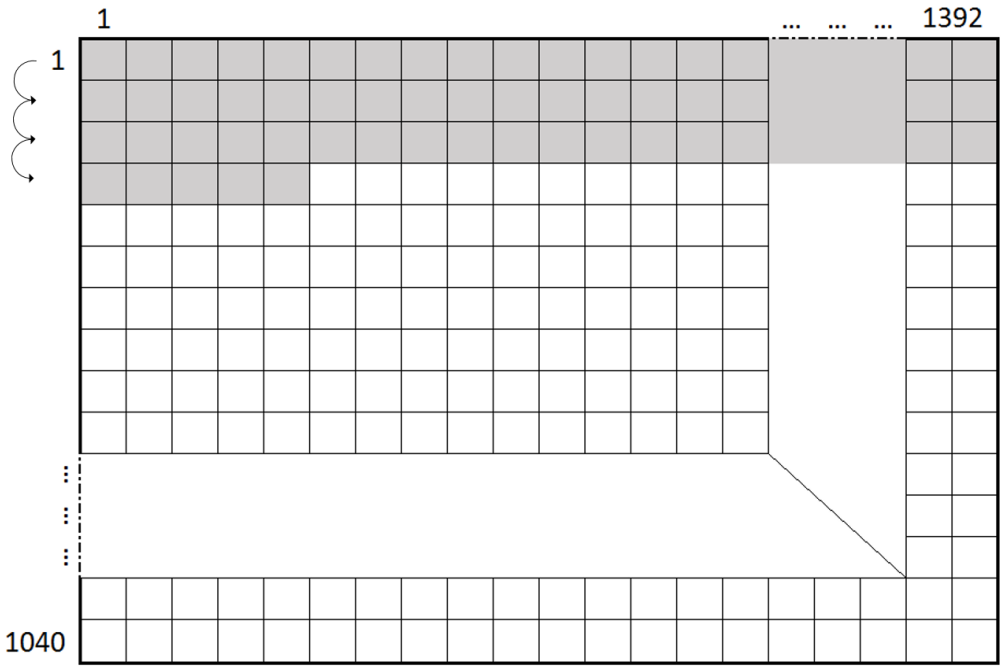


Figure 3-25 Pixel Creation B

Vertical columns are filled until the entire loop has completed the image creation. This process is only used to acquire the image correctly from the Basler camera system, after which it undergoes multiple processes to ensure the picture is processed correctly.

3.3.4.2 Region of Interest

The Region of Interest (ROI) is a selected region within a dataset that is identified for a particular purpose. This area is usually where processing is done, or a specific item is identified. The entire region outside of this specified ROI is completely ignored and unprocessed. Figure 3-26 below shows the ROI used on the image. The red line indicates the ROI which fits directly around the pallet.

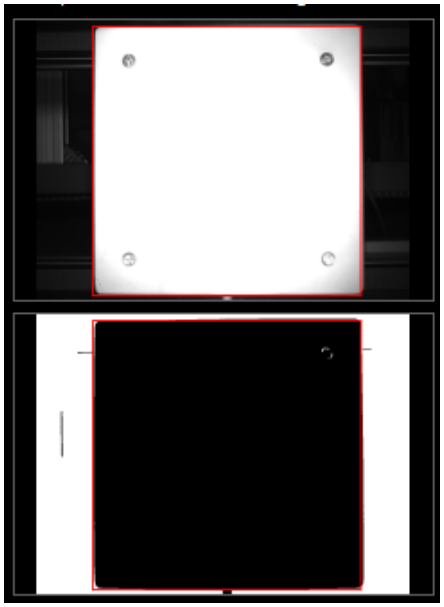


Figure 3-26 Example of the region of interest placed around the pallet within the software.

3.3.4.3 Calibration

The ROI is calibrated carefully in conjunction with the KUKA Robot System alignment above the pallet. Accurate measurements using the visual processing requires a 1000x1000 pixel block that fits perfectly around the pallet. This perfect fit requires the KUKA Robot's end effector to be set at a specific height. Imperfect fits would create errors in distance calculations.

Calibration was done for both the Product pallet and the Design pallet. The pallets would stop at the same spot every time due to the stop gate placement, making calibration a one-time process for a single pallet on each side. All future pallets arriving at these points should maintain the calibration properties.

3.3.4.4 Distance Measurement

Distance measurement techniques were applied on the acquired image to determine placement of the components. All distances were measured to the centre point of the pallet, which had a default coordinate of 0 and 0 on each axis respectively. The pixel-based measurements enabled calculation of the distance to each centre point of gravity using average pixel distance in an image. The region of interest block was fixed at 1000x1000 pixels and a coordinate system was created that could easily be related to real-world distances.

The acquired image was fixed at 1392 x 1040. The pallets measured 16 cmx16 cm and a block sized 1000x1000 pixels could fit perfectly around the edge of the pallet, meaning that each pixel equates to 0.16 mm. These data values and the number of pixels between the centre point of the pallet and the components' centre point of gravity, enabled calculation of the exact distance of the component from the centre of origin.

The height of the camera is crucial to ensure accurate results. The KUKA Robot's position above the pallet requires precise calibration. Figure 3-27 is a representation of pixels correlating to distance when doing distance measurement on an image.

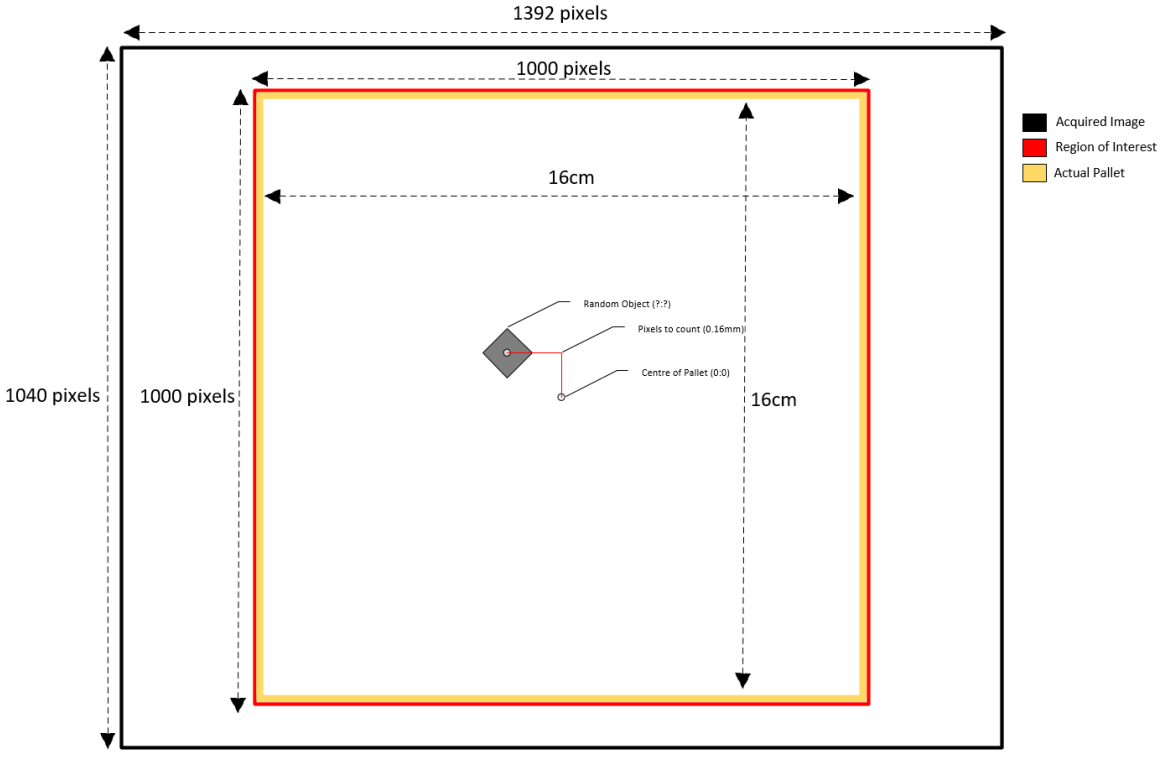


Figure 3-27 Image Distance Measurement Example (Not to scale)

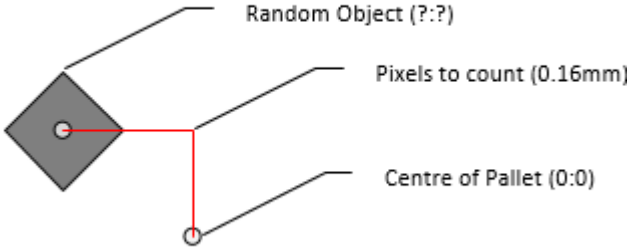


Figure 3-28 Distance Measurement (Close up)

As seen in Figure 3-28 above, the red line depicts the path that the KUKA Robotic System would move to align itself above the centre of gravity of the object. The number of pixels would be counted between these points, which correlates to the distance to be moved in the physical world.

The red line depicted above shows the required movement, purely to calculate the number of pixels. In reality, the KUKA Robotic System would move in a more direct path as seen in Figure 3-29 below. This is due to the X and Y movements happening simultaneously.

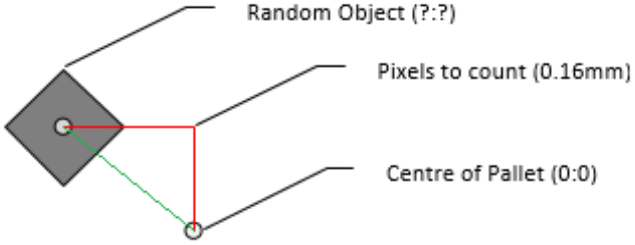


Figure 3-29 Distance Measurement (Direct Path)

3.3.4.5 Aforge.net

Aforge.net is a C# framework designed for developers and researchers in the fields of Computer Vision and Artificial Intelligence. This includes sub-categories such as image processing, neural networks, genetic algorithms, machine learning and robotics. The platform comes with many features in each category to ease the task of developing new innovative systems.

The system was implemented within this project taking up the role of image processing on all the acquired images. Using this diverse tool, many features could be extracted and processed from a single 2D-image. Features in the system included (but were not limited to) centre of gravity calculations, edge detection, template matching, blob detection, erosion filling and subtraction and addition algorithms. General image processing functions are also available in the platform such as invert functions, greyscale functions and thresholding.

The system used blob detection to isolate different items on the pallet that were within the defined threshold, such as size and region of interest.

Aforge.net is free to use and available online, making it a convenient tool for image processing in a budget-critical application.

3.3.4.5.1 Centre of Gravity

Centre of gravity was a key factor when picking up objects. The centre of gravity can be defined as “a point from which the weight of a body or system may be considered to act.” This valuable data was used as the best possible pick-up location on each object, enabling the KUKA Robotic System to maintain stability when moving components.

The vacuum system for acquiring components required even weight distribution of the components to avoid detachment mid-transit. The centre of gravity was acquired using the method seen within section 2.2.5.8.

Distance measurement was calculated to determine the best possible path to this point from the centre point of the pallet (one of the default reference points of the KUKA). Figure 3-30 shows this process taking place.

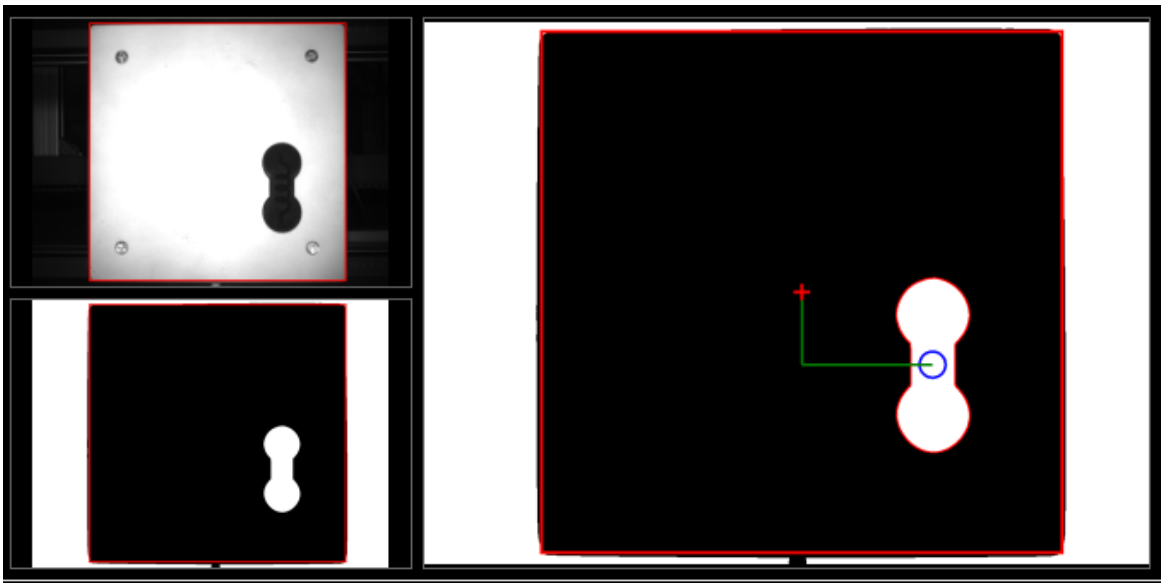


Figure 3-30 Example of the system applying multiple processes to an acquired image

The green line layered over the image indicates the path that the KUKA Robotic System would need to follow to manoeuvre itself above the object to be picked up (or placed). In reality, the X and Y movements are carried out simultaneously, therefore a more direct path the centre of gravity is followed.

The circle representation as seen in Figure 3-31 below was a replica of the sucker size and positioning over the component when collecting the object.

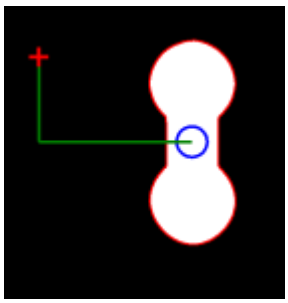


Figure 3-31 Centre of Pallet to Centre of Gravity preview

As seen in Figure 3-32 below, the system returns the measurements from the centre of the pallet to the pickup location on the object which was the centre of gravity of the component. As the starting point for real-time control was the centre of the pallet, the exact distance to be fed into the real-time controller was established using the component's centre of gravity.

```
[01:19:38] Object 1:X: 250,6588 Y: 138,6713
[01:19:38] Object 1:X: 40,105400390625mm || Y: 22,18740234375mm
```

Figure 3-32 Raw data output of object location

3.3.4.5.2 Raw and Central Moments

Moments were used to return vital information about the object(s) in question during image processing on the pallet. This information included the centre point of the object as well as the orientation data.

3.3.4.5.3 Edge Point

Edge point detection was used to outline the perimeter of the component and clearly visualize the component detected. It also clearly distinguished the component from other unique components within the processed image. As the component pallets were using a white base, while the components were black, their edge point perimeter was easily detected by the camera system, assuming the system was adequately lit and correctly calibrated.

3.3.4.5.4 Erosion

Erosion filling techniques were implemented on all image processing to eliminate noise on the acquired image. Several factors caused erosion, such as external light, reflections and calibration issues. These issues appeared as white static on an image as seen in Figure 3-33 below.

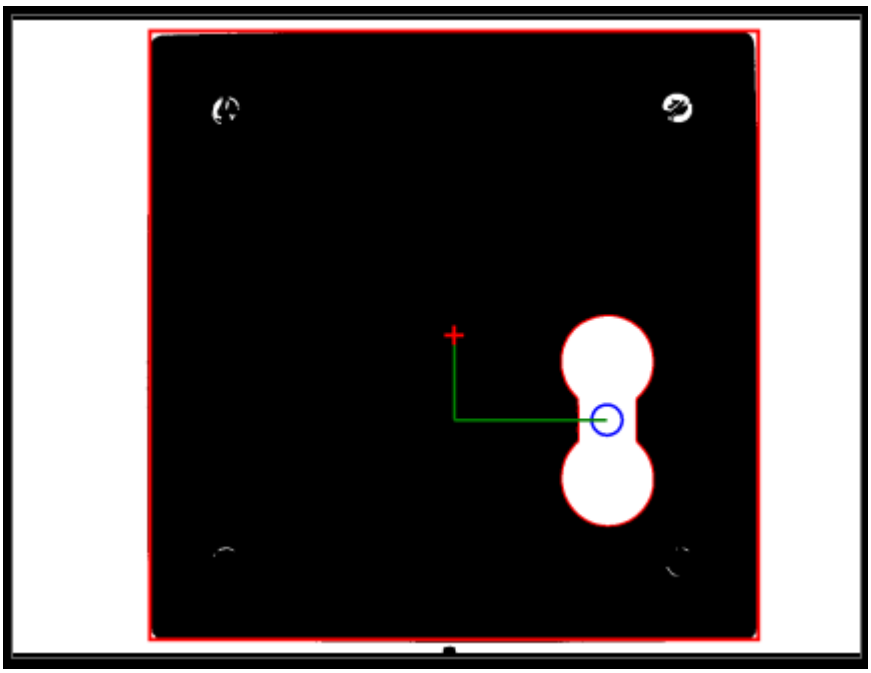


Figure 3-33 Raw Image with noise present

As seen above, the screws are visibly affecting the output of the image due to the reflection of the metallic tip. Multiple erosion filter iterations patched the holes successfully as seen in Figure 3-34. Multiple solutions are available to contain this problem within image processing, but erosion was sufficient in this case.

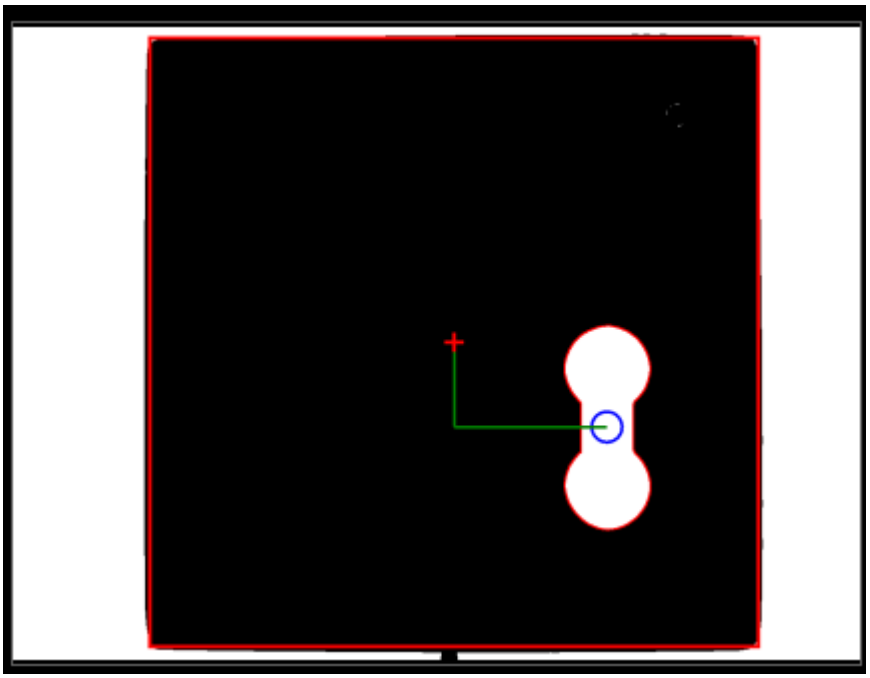


Figure 3-34 Resultant Image after Erosion processing

Almost 99% of the noise was eroded and only the object in question is detected. All subsequent noise was patched, improving the accuracy of the detection system.

This ensured no false positives were detected within the image. To ensure the component itself was not eroded, a minimum threshold was set up in terms of component size. In this case, the metallic tips were much smaller than the components used on the system.

3.3.4.5.5 Template Matching

Template matching was required to compare orientation angles from blobs captured on the design pallet. For the correct template to be matched to the blob, the design blob is first compared to the values within the database such as area, corner features, etc. and when these values correspond to the current blob at hand, the correct template is referenced for calculating the orientation data. This ensured the component could be rotated correctly, as all the components on the component conveyor were placed at a default orientation of 90 degrees.

On the design pallet end, the components were randomly placed, requiring orientation data calculation. The rotation of the actual KUKA Robotic from one conveyor to the next also had to be considered. The two conveyors were parallel to one another so work done on one end would be 180 degrees out of phase compared to work done on the opposite end. When the camera was also attached to the rotating KUKA Robot, this kept the components “in-sync” with the current job. Figure 3-35 shows two template examples created by the system.

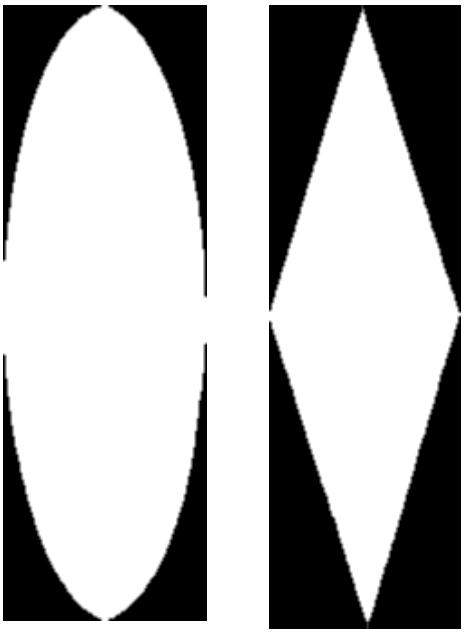


Figure 3-35 Example of two templates created by the system that is compared during operation

3.3.4.6 Accord.net

Accord.net is an open source .NET library written in C# with audio and image processing capabilities. The complete framework contains tools for building production-grade computer vision, computer audition, signal processing and statistics applications including commercial application uses. Accord.net contained vital libraries for orientation calculations needed within the program that did not feature within the Aforge.net library, hence the use of this additional framework. The framework contains imaging functions such as raw and central moments required to impose an orientation on an image (or blob in this case).

3.3.4.6.1 Orientation

Calculating the orientation of a blob was vital for correct placement of the object. Once a blob was extracted from an image, it would need orientation data to accompany it for the component to be rotated correctly at the design pallet section. Orientation calculations were performed on an extracted blob from the design pallet section during the first learning cycle of a new build. This new orientation value joined the data regarding the blob. Note that a blob's orientation is defined within 180-degree cycles. This means that theoretically, if a component was rotated 180 degrees, it would have the same orientation as previously seen.

The solution to this problem was to create an image subtraction function that would take the default template (which is at 0 degrees), offset it by the difference between the blob extracted and the template in question and then perform a subtraction of the two images. If the result of the subtraction was not 0 or the image was not 99% similar (due to noise present at the time of image capturing), it would be obvious that the two objects were 180 degrees out of phase. The simple solution was to either add or subtract an additional 180 degrees dependant on the original angle. This would then allow the component to be rotated correctly as seen in the initial design.

3.3.5 Network System

3.3.5.1 Overall Network

The network system played an important communication role within the project. All systems were connected via Ethernet standard, while implementing the TCP/IP protocol. An industrial Siemens Ethernet switch was used for data switching with all sub-components connecting to this connectivity device.

A general network overview can be seen in Figure 3-36 below, which depicts the network connectivity between all the devices used.

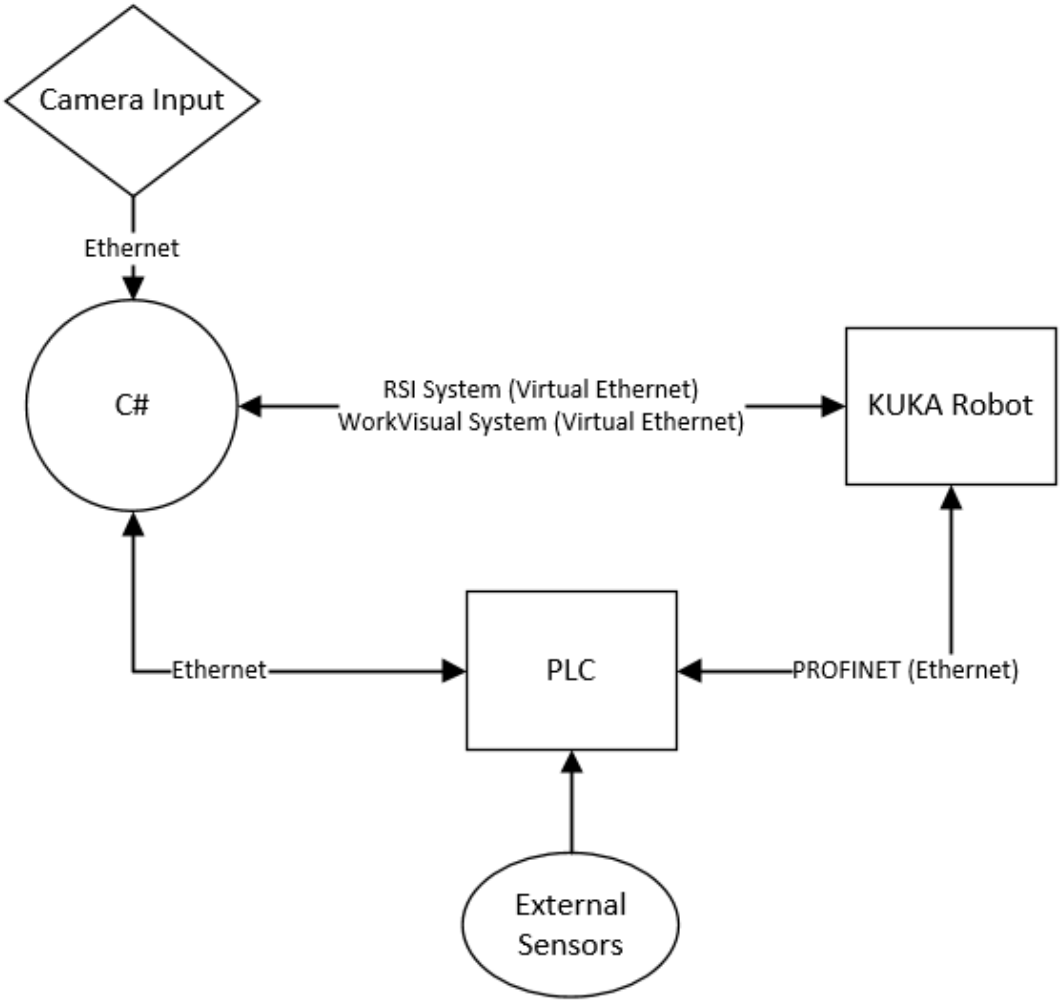


Figure 3-36 Network Overview

3.3.5.2 PROFINET Connection

A PROFINET connection is an industrial Ethernet connection protocol established between two industrial devices. Many services are distributed over a single Ethernet connection including:

- Real-time communication
- Distributed Field Devices
- Drivers & Motion Control
- Distributed Intelligence
- Network Installation
- IT Standards and Security
- Safety
- Automation Processing

PROFINET allows for seamless integration of industrial devices to establish successful communication and setup. PROFINET was set up between the KUKA Robotic and the PLC for the two devices to share processing data with each other. Shared variables were set up across the two devices, each with its own read/write properties to pass on feedback data and control flow of the system.

3.3.5.3 TCP/IP

A static TCP/IP system was implemented to bypass the need for a DHCP server. This allows consistent access to the same equipment via the same unique IP address, eliminating the problem of changing IP addresses due to DHCP leasing.

Table 1 - Internet Protocol Device List

Device	IP Address
Basler AG Camera	192.168.4.7
Siemens S7-1200 PLC	192.168.4.8
KUKA X66 Ethernet Port	192.168.4.10
KUKA X66 Ethernet Port (Virtual)	192.168.2.10
PC Adapter	192.168.4.11
PC Adapter (Virtual)	192.168.2.11

An IP address has two components, the network address and the host address. A subnet mask separates the IP address into the network and host address.

Subnet masks were all set at 255.255.255.0 to ensure the last segment in the IP address was addressed as a host while the first three segments were addressed as the network. This is vital to ensure no cross-communication occurs between the virtual networks.

Firewalls on all the systems were turned off to ensure no port blocking was present during communication. As this was a private network separate from the internet, no repercussions could be suffered from doing so.

3.4 Conclusion

This chapter contained a detailed overview of the techniques and design strategies implemented within the formation of this system. The various subsystems that were deployed to complete the overall objective was also covered, showing each process individually in detail.

4 Chapter 4: Results

This chapter discusses a series of tests that were undertaken on the system to test its capabilities. The system was challenged through various experiments that would require it to build a “Design” in the quickest, most efficient manner possible. The speed and accuracy were recorded under different circumstances such as unique designs, similar components within the designs, etc.

Comparisons were conducted between the manual procedure of configuring the KUKA Robotic System and the Automated Process that this study presents when introducing a new design on an existing assembly system. Factors such as ease of setup, time consumption and reliability were all considered while allowing leeway for the introduced system to thrive.

Although the speed of picking and placing components is greatly reduced due to the real-time control of the KUKA Robotic System, movements between fixed points are still capable of running at 100% speed as these are fixed points within the KRL program on the KUKA Robotic System.

All results are tabulated within this section in terms of speed (time) and accuracy with a discussion following these results.

4.1 Open Pallet Design

Contrary to conventional methods, an open pallet design was used to accommodate any style or “object” on the system. This created a versatile system that could work with any object of any size (within the dimensions of the pallet) placed on the pallet. A fixed (moulded) designed pallet that can fit specific objects is beneficial in many scenarios, but this setup limits the flexibility of the system and therefore an open pallet design was best.

A rubber mask was applied to the top of the pallet to ensure no movement of components during normal conveyor transit (general movement, sliding at stop motion on stop gates). Initially, components would slightly rotate (as all components are defaulted to 90 degrees on the component conveyor) which resulted in a slight offset when components were placed on the design pallet. The rubber mask layer corrected this issue, creating resistance to movement of the components.

Figure 4-1 below shows an example of the open pallet design with a rubber mask layer applied to the surface.

Figure 4-1 Open Pallet Design with Rubber Mask

4.1.1 Components Used

For simplicity and proof-of-concept purposes, the designed components resembled random shapes distinguishable from one another using different surface areas and corner properties. These shapes, designed in CORELDrawX7, were laser cut out of 3 mm wood and referred to as components for the purpose of this study. The component pallets held multiple similar components, while the design pallet contained multiple uniquely designed components.

The components were spray painted with a matt black finish to ensure easy detection on the pallet when the camera captures an image.

Note that the components used were not of fundamental importance within this study. They were merely used as a gateway to prove the real-time concept using KUKA RSI and Image Processing. Figure 4-2 depicts the components used.

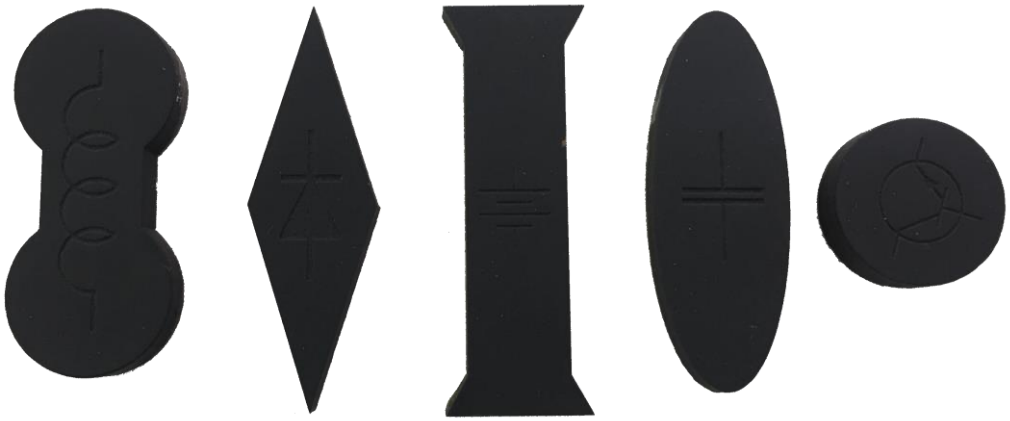


Figure 4-2 Different style components used for testing purposes

4.1.2 Configurations Used

Similar component types were grouped together as a unit on the component pallet conveyor. This ensured that the system learned all the components available to the system in the most efficient manner possible. All components were initially placed at the same angle on the component pallet, from which a template would also be generated by the system and used later when detecting offset of components on the design pallet conveyor. Figure 4-3 depicts how the components were configured on the component pallets. This example only shows one of the

five components' pallets. The remaining components are configured in a similar fashion.

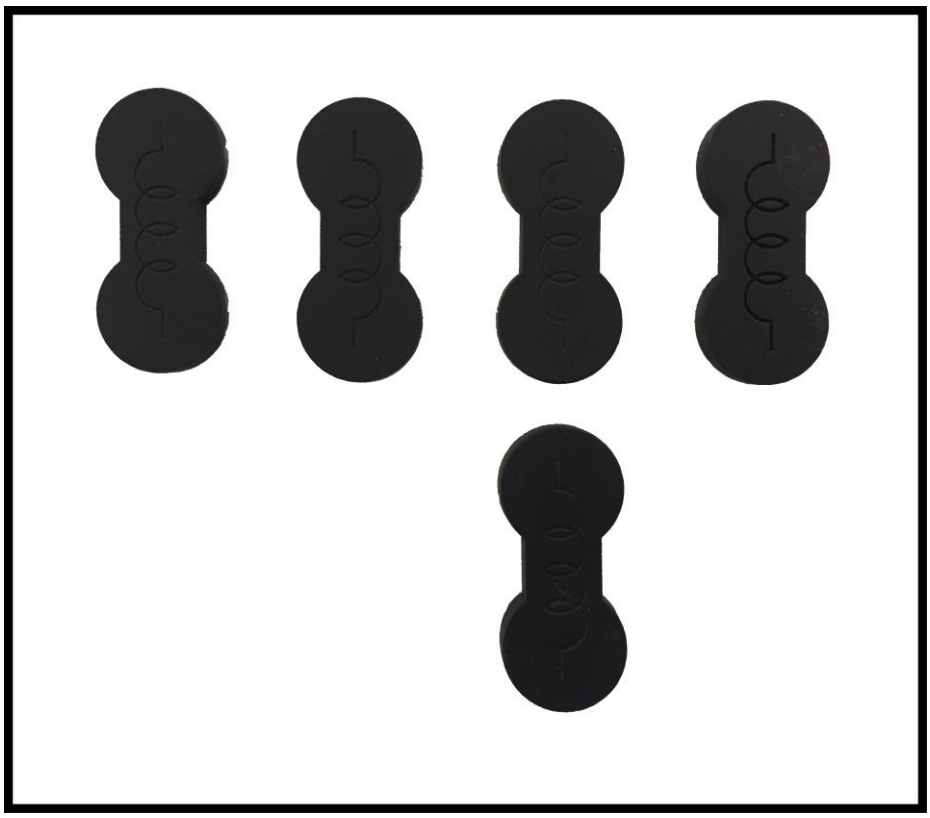


Figure 4-3 Example of component pallet configuration

As seen in Figure 4-3 above, the components are all rotated to the same angle. This is required as the template is created from a single component within the overall image. If certain components were offset by a different angle, the resulting angle on the design pallet when placing the component would be incorrect.

Various configurations of different components were used on the design pallets, ensuring complete randomness was maintained throughout for the system to determine new paths on every iteration of a build required. This allowed speed measurement recording and analysis of system accuracy in different build environments. Figure 4-4 depicts the different configurations used during testing on the design pallet.

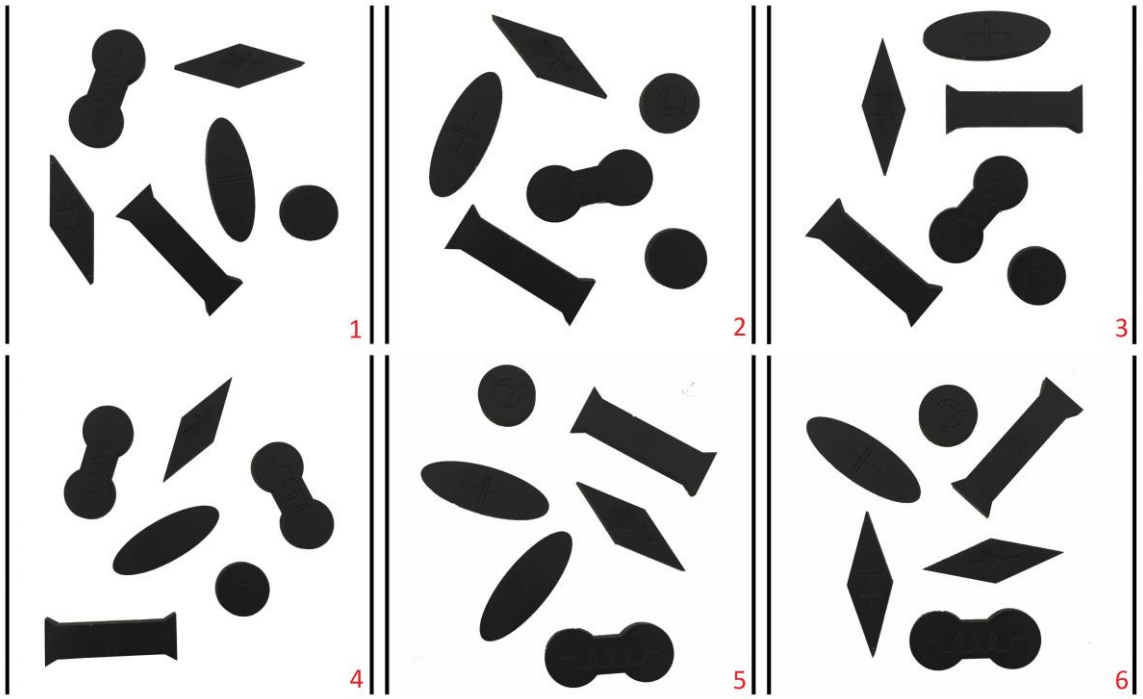


Figure 4-4 Various configurations used one through six

As seen in the figure above, these configurations had no fixed pattern and were set in random configurations for testing purposes. The system had no pre-programmed positions regarding collection or placement of these randomly placed components, therefore requiring the real-time image processing and movement to access these components on the pallets.

4.2 System Learning

4.2.1 Introduction

The system has no knowledge of any components or designs within its repository, therefore some sort of learning is initially required. The system is required to learn all the available components that can be used to build a new design on the component conveyor before starting to build.

The system analyses each component pallet individually and proceeds to save all data regarding the design within the system database. Once a pallet has been scanned and saved, the stop gates trigger sending the pallet forward, allowing the next pallet to enter the scanning area. This process continues until the first component pallet returns to its initial position. This is confirmed by re-scanning the first pallet to confirm that the component type exists within the database.

Figure 4-5 shows an example of where the pallet scan takes place and the movement of the pallets along the component catalogue conveyor.

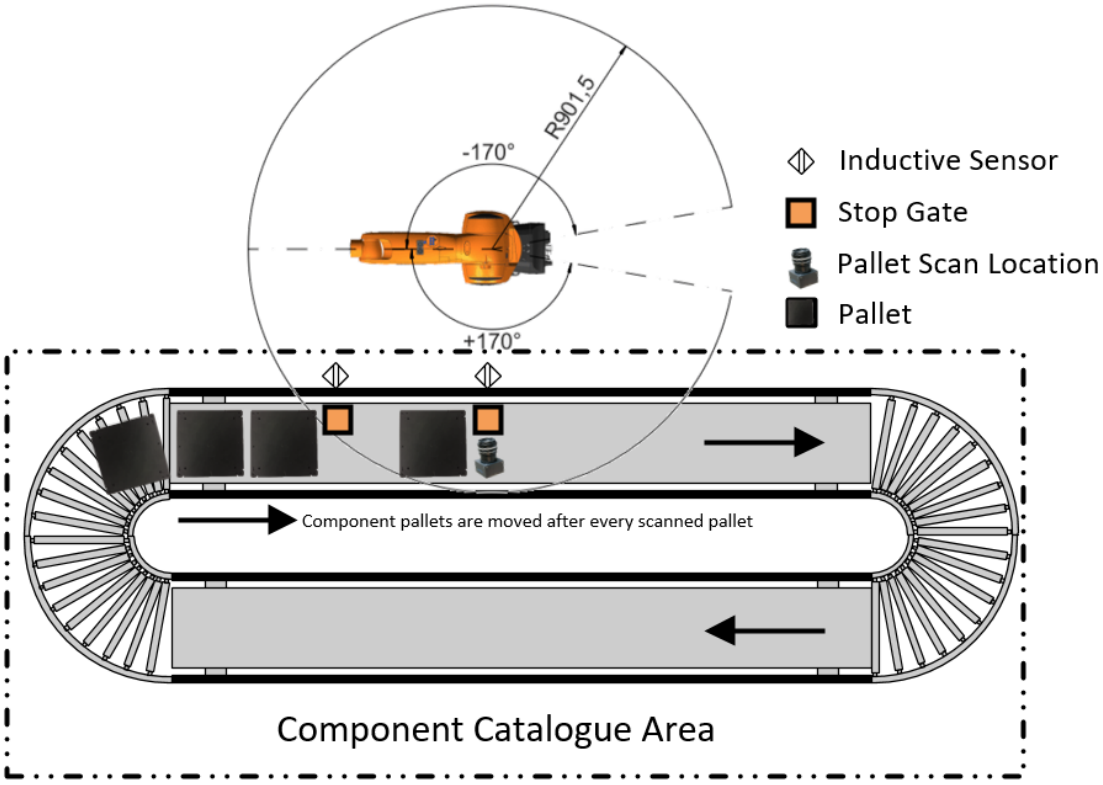


Figure 4-5 Component Catalogue Conveyor

Once all components are mastered on the catalogue conveyor, the system is fully prepared to redesign any design observed on the design conveyor, as long as the design pallet only contains components available within the component catalogue area.

4.2.2 Processing

Once a picture is acquired by the system, the picture is analysed and the resulting data stored within a database object. Only data within the region of interest is analysed and saved. Calculated data includes centre of gravity, number of corners, position relative to the centre of the pallet and area of the object. Each seen component is saved individually, from which an overall component pallet object is then created and identified. Due to lighting, results may differ in terms of area of the objects. When a component pallet is scanned, an average area of all the components is calculated as all component types are the same. This allows more

accurate detection of similar components on the design pallet end when rebuilding a design. The process is repeated for each new component pallet seen. Before adding any pallet to the database object, the system will first compare the incoming data with current data in the database, ensuring no data is duplicated.

The area of the object and corner objects detected are used at certain thresholding levels to detect similar components. This implementation method was used for simplicity purposes and relates to the underlying study as any implementation option could be utilized.

Figure 4-6 and Figure 4-7 depict the results received visually and the raw data when a picture is acquired.

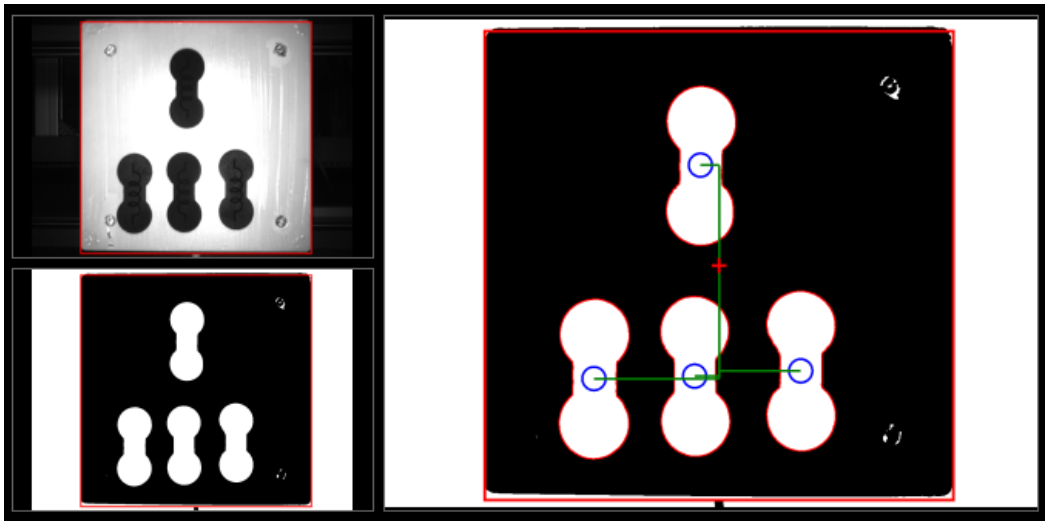


Figure 4-6 Image Processing complete on component pallet with centre points shown

```
[02:11:43] Object 1: POSITION DATA
[02:11:43] Object 1:X: -14,41931 Y: -210,865
[02:11:43] Object 1:X: -2,30708984375mm || Y: -33,7383935546875mm
[02:11:43] Object 1:X: 187,2839 Y: 226,3495
[02:11:43] Object 1:X: 29,9654199218 75mm || Y: 36,215927734375mm
[02:11:43] Object 1:X: -39,22491 Y: 236,7068
[02:11:43] Object 1:X: -6,275986328125mm || Y: 37,8730859375mm
[02:11:43] Object 1:X: -254,1015 Y: 241,2233
[02:11:43] Object 1:X: -40,656240234375mm || Y: 38,59572265625mm
[02:11:43] Object 1: Corners: 799
[02:11:43] Object 1: Area: 39196
[02:11:43] Object 1: Total Objects: 4
[02:11:43] Object 1: Region Data: X: 213  Y:25  Size:1000
[02:11:43] New Component Design -- Adding Pallet...
[02:11:43] Saving Template of component...
```

Figure 4-7 Raw data results from the image processing

As seen in Figure 4-7, the overall average area and corners are returned and saved. Each component's individual distance from the centre point is also returned and used to manoeuvre the KUKA Robotic System in real-time. The KUKA Robotic System then carefully moves to pick up a specified component when required within the design. This process outlined above concludes the processes for creating a component pallet in the system.

A similar process is carried out when learning a new design pallet. Instead of the system learning a unique component individually, a design pallet contains multiple unique components on one pallet that must be compared to the available learnt components outlined in the previous section. For the system to be able to rebuild the seen design, all components on the design pallet must be learnt beforehand on the component conveyor and be available for use.

Once these pre-requisites are met, the system will process the design pallet, creating a data object revolving around this newly learnt design pallet. Each component is compared using area, corners and template matching to the component pallets and when matched, are added to the new design pallet object. If a match is not found for a component within the design, an error is returned, and the process is terminated.

Figures 4-8 and 4-9 show the results after image processing takes place at the design pallet conveyor. Figure 4-8 shows the first configuration for demonstration purposes.

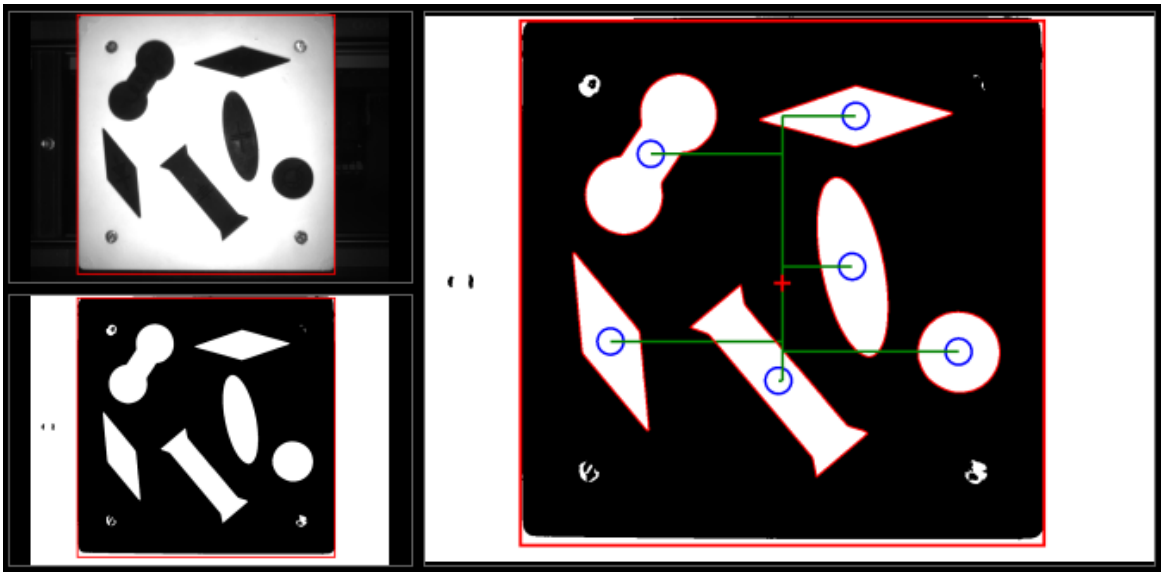


Figure 4-8 Image Processing complete on design pallet with centre points shown

```
[03:17:34] Checking Design...
[03:17:34] Found Component Match! ID: 1
[03:17:34] New Component List for specific component! Adding Data...
[03:17:34] Template Match - Angle Difference: -32.56441
[03:17:34] Found Component Match! ID: 4
[03:17:34] New Component List for specific component! Adding Data...
[03:17:34] Template Match - Angle Difference: -86.12384
[03:17:34] Found Component Match! ID: 2
[03:17:34] New Component List for specific component! Adding Data...
[03:17:34] Template Match - Angle Difference: 11.16994
[03:17:34] Found Component Match! ID: 4
[03:17:34] Template Match - Angle Difference: 24.76661
[03:17:34] Found Previous Component Match! Adding Position...ID: 4
[03:17:34] Found Component Match! ID: 1
[03:17:34] Template Match Error - Adding -180 Degrees for opposite rotation! Actual: 41.44056 Result After: -138.5594
[03:17:34] Found Previous Component Match! Adding Position...ID: 1
[03:17:34] Found Component Match! ID: 0
[03:17:34] New Component List for specific component! Adding Data...
[03:17:34] Template Match - Angle Difference: -10.21719
[03:17:34] Adding Overall Data as New Design...ID: 0
```

Figure 4-9 Results feedback with angle difference shown and component ID

As seen above in Figure 4-9, the system outputs the angle difference compared to the template, which is captured when processing the component conveyor components. A template match error occurs and the system compensates by adding -180 degrees to ensure correct placement on the design conveyor. This is found by rotating the component to the nominal 90-degree template image format and subtracting it from the template. If the result is not a 99% match, a template match error occurs, and 180 or -180 degrees is added to the current angle, as the component is out of phase by 180 degrees. Whether the angle added is positive or negative is based on the initial angle result calculated by the system (offset from 90 degrees).

Once the process is complete, the system saves the data within an overall object and a design ID is assigned. The system now has all the data required to rebuild the seen design.

The data required to build a seen design includes (for each component on the design pallet):

- Component ID (matching component seen on component conveyor)
- Centre of gravity
- Number of corners
- Distance on X and Y axes to centre point of pallet.

This data is captured within an overall design object, referencing the seen object's ID generated by the system when learning the components for the first time.

4.3 System Building

The system is ready to rebuild a seen design once:

- All components have been learnt on the component conveyor.
- The required build has been seen (and learnt) on the design conveyor and the data has been saved on the system.
- An empty pallet is presented to the system on the design conveyor.

The system has the option to either rebuild the previous seen design or build a specific design decided by the end operator (providing that there is more than one seen design).

This process involves searching for the required component on the component conveyor. Once found, the system moves the KUKA Robot to the pickup location where real-time control is activated and the system takes full control of the KUKA Robot's movement. The system manoeuvres the end effector to the component's centre of gravity to safely pick it up. Once above the component (motion complete on the X and Y axes), the system will then move on the Z-axis, 63mm in a downwards motion until the end effector is flush with the required component. The sucker system then activates, picking up the component and attaching it to the KUKA end effector.

Real-time control deactivates at this point and pre-programmed settings move the KUKA Robot over the empty pallet on the design conveyor. Once the KUKA Robot is confirmed in position above the empty pallet, real-time control is reactivated. The system uses data captured during the design pallet learning phase to place the component in the same manner as it was learnt. This involves navigating the KUKA Robot to the centre point of the previously seen component and then rotating it to match the component design offset.

4.3.1 Step-by-step procedure

In the example to follow, the system has no knowledge of any component of design. The following step-by-step procedure will be demonstrated using the first test configuration.

The system will start by learning all the components available on the component conveyor individually, while saving all associated data within the pallet's object. The system moves each conveyor after each pallet completes the learning process.

Figure 4-10 depicts the KUKA Robotic System learning a component pallet. The pallet is bright due to the active lighting system.

Figure 4-10 Component Conveyor - KUKA Robotic scans pallet to learn items

Figure 4-11 shows the data returned via console for every component pallet learnt. All data regarding the components learnt can be seen here.

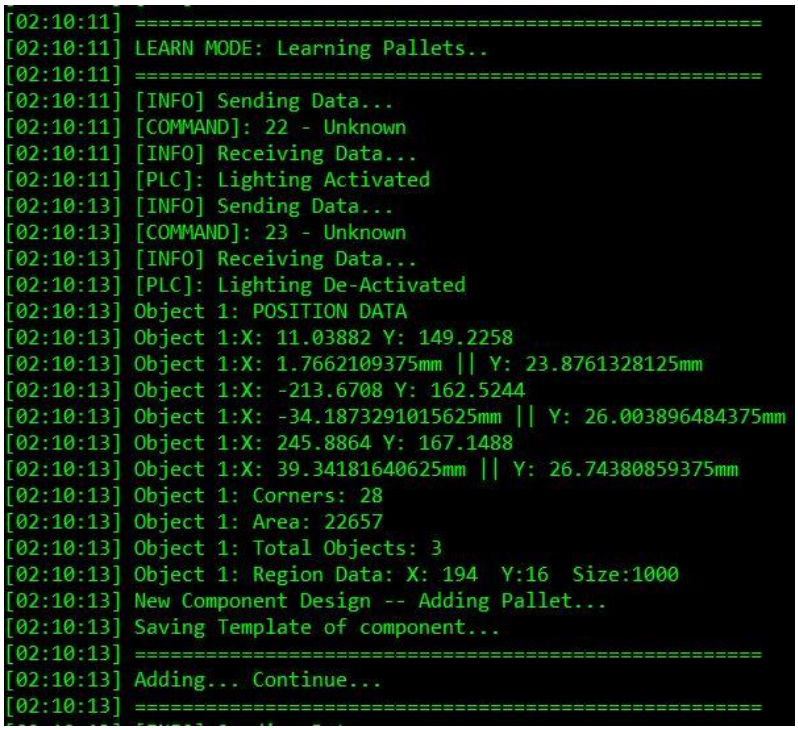


Figure 4-11 Data returned after learning the first component pallet.

As seen in Figure 4-11, each component pallet is learned individually, and all the respective data is presented through the console. The total area of the object, the corners (using Graham Scan, contour calculation) and the total number of objects on the pallet with their respective positions and the distance from the centre of the pallet can be seen. A template is then saved of this component and an ID is assigned. This process continues until all pallets on the conveyor have been scanned.

Figure 4-12 shows the initial component learning process being completed.

```
[02:10:55] Object 6: POSITION DATA
[02:10:55] Object 6:X: 10.86957 Y: 148.9794
[02:10:55] Object 6:X: 1.739130859375mm || Y: 23.83669921875mm
[02:10:55] Object 6:X: -213.6938 Y: 162.4224
[02:10:55] Object 6:X: -34.1910009765625mm || Y: 25.987578125mm
[02:10:55] Object 6:X: 245.7542 Y: 167.1382
[02:10:55] Object 6:X: 39.3206640625mm || Y: 26.742109375mm
[02:10:55] Object 6: Corners: 31
[02:10:55] Object 6: Area: 22930.67
[02:10:55] Object 6: Total Objects: 3
[02:10:55] Object 6: Region Data: X: 194  Y:16  Size:1000
[02:10:55] Seen Component Design -- Updating Pallet...(Pallet Matches Component: 0)
[02:10:55] ========================================================
[02:10:55] First Component is back. All components learnt!
[02:10:55] ========================================================
[02:10:55] ========================================================
[02:10:55] Process Completed Successfully!
[02:10:55] --------------------------------------------------------
```

Figure 4-12 First component pallet arrives back at the scanning point

The system shows that the initial pallet that was scanned has returned, therefore all the pallets have been scanned.

Now that all the components have been learnt, the system is ready to learn its first design. As stated previously, the brightness is due to the active lighting system during image capturing.

Figure 4-13 System learning a new design to rebuild

Figure 4-14 Data returned from learning the design pallet with matches found

Each component match can be seen within the console feedback, with the respective ID match including the angle difference to the template.

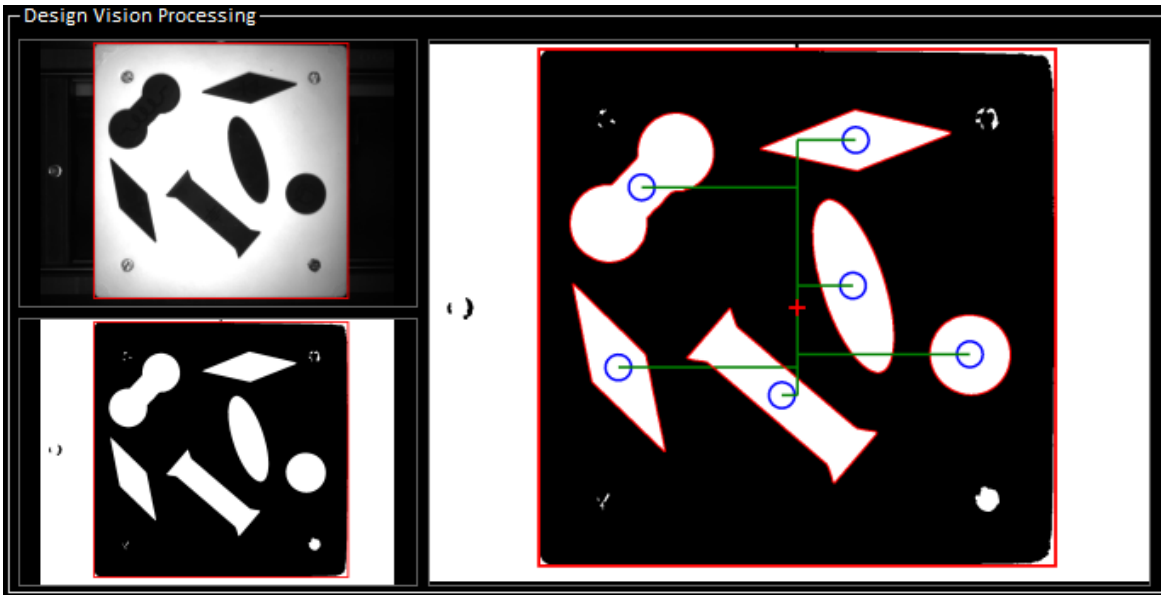


Figure 4-15 First design learnt by the system

Now that the presented design has been learnt successfully, the system is presented with an empty pallet and begins to build the previously seen design. The KUKA Robot will return to the component conveyor to search for the component(s) required to build the design (as the system has confirmed that they exist on the component conveyor).

Figure 4-16 below shows a component compared to the template within the system. As seen in the subtraction, the match is nearly 99% when the subtraction takes place. In the design block, the top component on the design pallet is being compared.

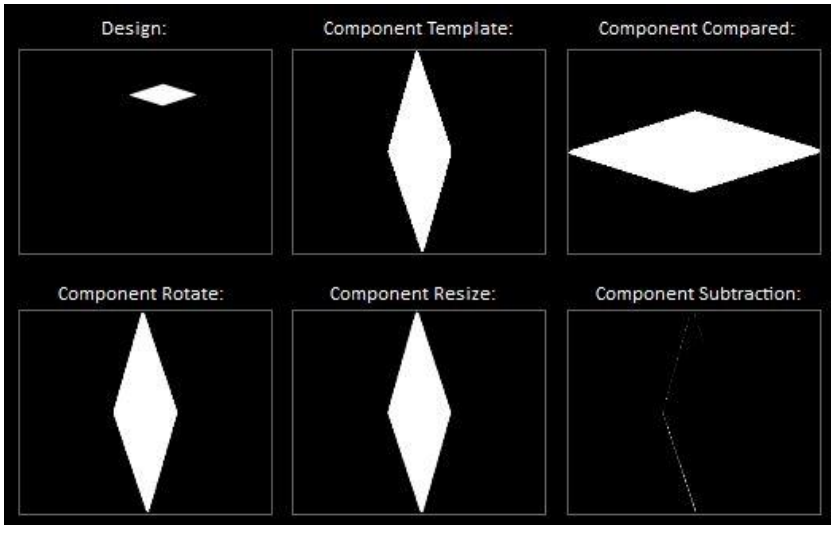


Figure 4-16 Visual Comparison of a design component to the learnt components

Figure 4-17 depicts the system finding the component on the component conveyor and activating real-time control to pick up the specified component.

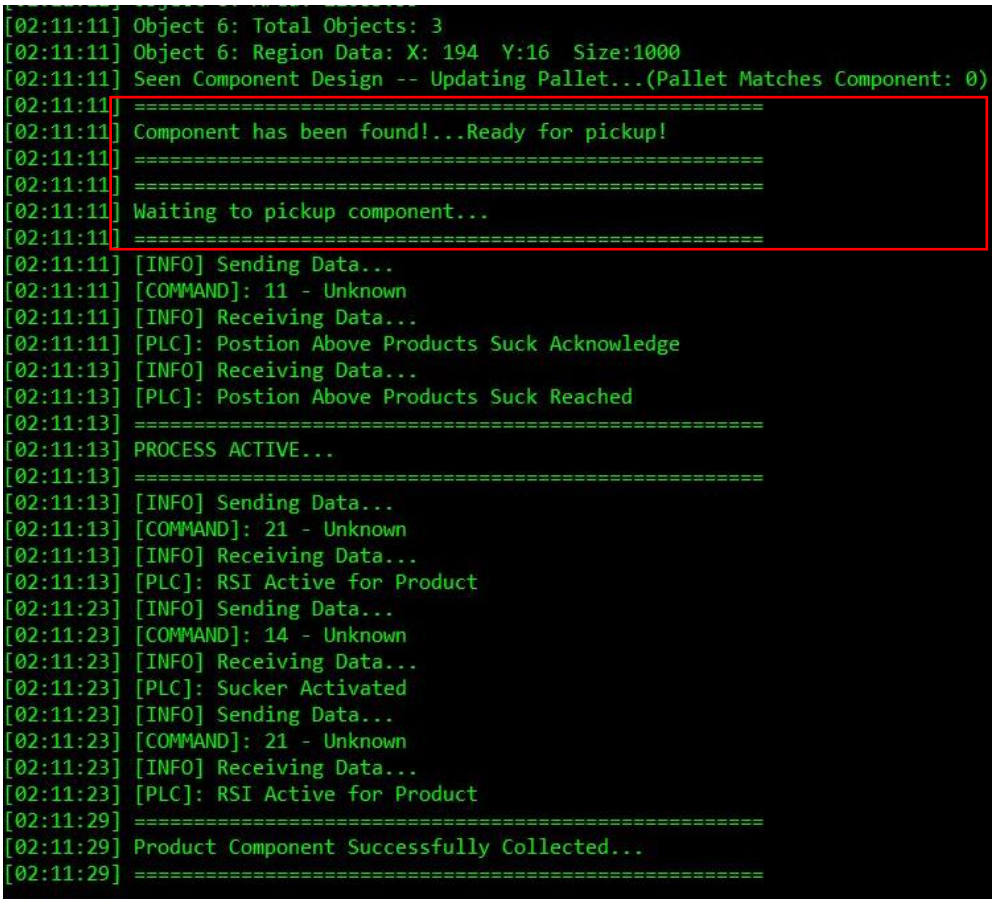


Figure 4-17 Example of the system finding a component and preparing for collection

Now that KUKA RSI is activated, the system has full real-time control over the KUKA Robotic System. The KUKA Status section, as seen in Figure 4-18 below, shows the actual position of the KUKA Robotic System as well as the target location. This is in reference to the Base Coordinate system.

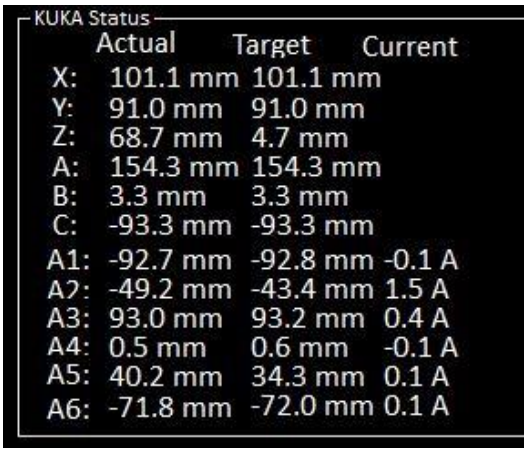


Figure 4-18 KUKA Status, returned by the KUKA RSI System

Figure 4-19 shows the component collected in real-time from the component conveyor (left) and placed on the empty design pallet (right). The component is placed with reference to the design that is seen in Figure 4-15.

Figure 4-19 Example of the first component being picked up and placed on the design conveyor

Figure 4-20 below shows the feedback from console that the component has been successfully placed, along with all the references to the component in the system.

Figure 4-20 Console feedback of the process completing successfully

The same process repeats for each design component. The design pallet is populated with each pick and place iteration. Figures 4-21 and 4-22 show this process step-by-step.

Figure 4-21 Pick and Place of first configuration A

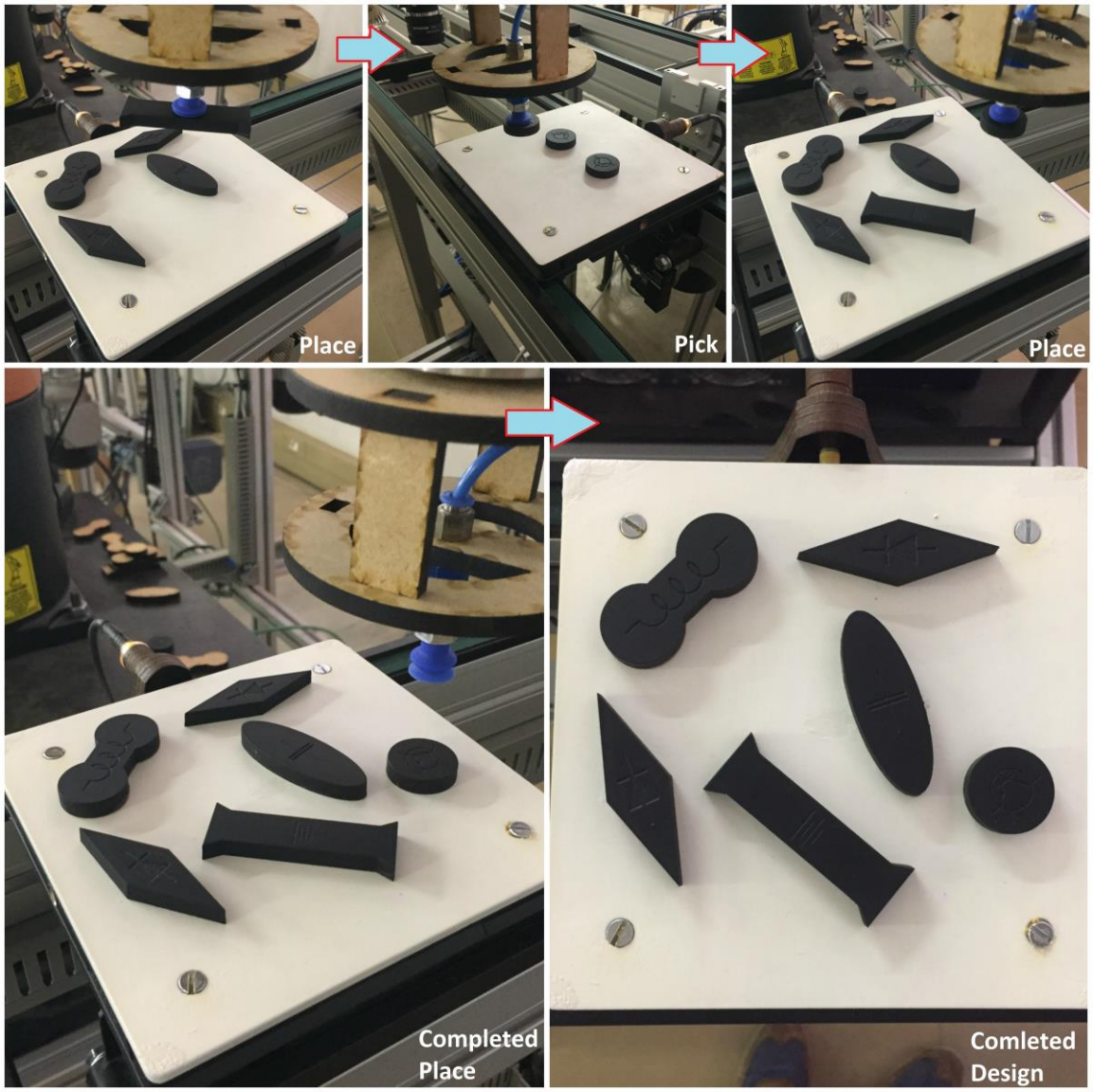


Figure 4-22 Pick and Place of first configuration B

The accuracy and speed of the system is determined by how well the system is calibrated. This is due to initial offsets from where the KUKA Robotic systems pre-programmed locations are and the camera's ability to accurately detect the distances from the centre point of the pallet to the objects placed on the pallet. The KUKA Robot's speed was limited to 50% because of its unstable supporting structure by not being bolted to the floor (limited support was available in test environment). As speed increases, vibrations around the system also increases, which would affect the stability of the system. If the system was bolted to the floor, the speed could be significantly increased.

4.4 Speed and Accuracy

4.4.1 Introduction

This section covers the speed and accuracy of the system with all results tabulated. All results were recorded completely automatically. The system's feedback regarding positions and time were used. Initially, the system returns all the components on the design pallet with their relative positions. Once the system rebuilt the required design, an additional scan was done, with the system returning the new positions (after build) of the newly placed components. These results help determine the difference between before and after and overall accuracy as shown below.

Time was also recorded using the internal timing of the system for accurate results. All components have a "number" and an "ID". The number represents the physical component number (irrespective of design) that the system is working on, while the ID refers to the internal design ID allocated to the design by the system.

In all cases, there would be at least two of the same component design, therefore the same ID number might appear twice. The system assigns IDs as they are learnt, meaning no ID equates to a certain "design of component". It solely depends on the order in which they were learnt. Components that have angles of 0° present in the tables below means the component was a circle, therefore no angle is considered. Placing this component in any direction does not affect the outcome.

4.4.2 Results from the first pallet configuration

Table 2 - Raw Data Results of the first pallet configuration

First Pallet Configuration										
Component		Before (from Camera Scan)			After (after KUKA Place)			Difference		
Number	ID	X(mm)	Y(mm)	Angle(®)	X(mm)	Y(mm)	Angle(®)	X(mm)	Y(mm)	Angle(®)
1	0	18.48589844	-52.65147705	89.70158	19.74491211	-51.9920849	89.20961	1.2590137	0.659392	0.49197
2	0	-50.6486328125	18.00361328	35.16379	-47.65557617	16.08262695	35.99622	2.9930566	1.920986	0.83243
3	1	16.06324219	-8.34162109	15.66263	15.94128906	-8.07501953	15.48299	0.1219531	0.266602	0.17964
4	2	-3.735078125	30.73827148	41.98695	-3.0694140625	29.35798828	41.4755	0.6656641	1.380283	0.51145
5	3	49.89367188	24.00540039	0	48.81323242	25.60262695	0	1.0804395	1.597227	0
6	4	-40.1546337	-36.8570996	-36.33524	-35.5884277	-39.38123	-36.18821	4.566206	2.52413	0.14703
Overall Average								1.7810555	1.391437	0.36042

Time is calculated for pick and place from the moment the KUKA Robotic System rescans the component conveyor pallet (with a positive match) to the point where RSI is deactivated after placing the component (the point where the real-time control ends).

Table 3 - Timing Results for configuration one

Number	ID	Objective	Time
		Learn Design	2.64s
1	0	**Pick* and *Place 1*	60.87s
2	0	**Pick* and *Place 2*	55.71s
3	1	**Pick* and *Place 3*	51.51s
4	2	**Pick* and *Place 4*	55.25s
5	3	**Pick* and *Place 5*	61.91s
6	4	**Pick* and *Place 6*	54.65s

*Recalculation of the component pallet is done before pickup for safety in case objects shift (time includes this).

4.4.3 Results from the second pallet configuration

Table 4 - Raw Data Results of the second pallet configuration

Second Pallet Configuration										
Component		Before			After			Difference		
Number	ID	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)
1	0	3.857792969	4.2646875	-83.3757	5.99578125	3.898447266	-80.96973	2.1379883	0.36624	2.40594
2	1	2.699355469	-47.25211	45.28602	4.283330078	-48.672146	43.88901	1.5839746	1.420036	1.39701
3	2	-46.3059424	-27.7214502	-39.67451	-40.007	-29.91125	-40.8503	6.2989424	2.1898	1.17575
4	3	-39.21145508	37.33771484	40.81068	-37.28689	34.774922	38.83356	1.9245651	2.562793	1.97712
5	4	50.68230469	-23.1707666	0	51.56443359	-23.392612	0	0.8821289	0.221845	0
6	4	44.6116113	36.44303711	0	41.38067383	36.48836914	0	3.2309375	0.045332	0
Overall Average								2.6764228	1.134341	1.159303

Time is calculated for pick and place from the moment the KUKA Robotic System rescans the component conveyor pallet (with a positive match) to the point where RSI is deactivated after placing the component (the point where the real-time control ends).

Table 5 - Timing Results for configuration two

Number	ID	Objective	Time
		Learn Design	2.62s
1	0	**Pick* and *Place 1*	56.45s
2	0	**Pick* and *Place 2*	56.67s
3	1	**Pick* and *Place 3*	56.20s
4	2	**Pick* and *Place 4*	55.40s
5	3	**Pick* and *Place 5*	52.77s
6	4	**Pick* and *Place 6*	60.83s

*Recalculation of the component pallet is done before pickup for safety in case objects shift (time includes this).

4.4.4 Results from the third pallet configuration

Table 6 - Raw Data Results of the third pallet configuration

Third Pallet Configuration										
Component		Before			After			Difference		
Number	ID	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)
1	0	9.93499023	12.4122656	-45.88563	11.2968458	12.237002	-47.20672	1.3618556	0.175264	1.32109
2	1	-27.9602637	-39.126631	-2.693703	-25.400254	-41.527373	-3.691467	2.5600097	2.400742	0.997764
3	2	12.5403418	-62.167528	87.60452	14.6525293	-61.805265	88.27964	2.1121875	0.362263	0.67512
4	3	30.998584	45.239414	0	31.392148	48.899453	0	0.393564	3.660039	0
5	4	24.84095703	-28.48924	-88.46047	25.806142	-30.171392	-87.60473	0.965185	1.682152	0.85574
6	4	-33.1905225	36.660098	37.55062	-32.377139	36.4132129	38.49828	0.8133835	0.246885	0.94766
Overall Average								1.3676975	1.421224	0.799562

Time is calculated for pick and place from the moment the KUKA Robotic System rescans the component conveyor pallet (with a positive match) to the point where RSI is deactivated after placing the component (the point where the real-time control ends).

Table 7 - Timing result for configuration three

Number	ID	Objective	Time
		Learn Design	2.34s
1	0	**Pick and Place 1*	53.17s
2	0	**Pick and Place 2*	52.19s
3	1	**Pick and Place 3*	60.88s
4	2	**Pick and Place 4*	61.34s
5	3	**Pick and Place 5*	58.17s
6	4	**Pick and Place 6*	53.91s

*Recalculation of the component pallet is done before pickup for safety in case objects shift (time includes this).

4.4.5 Results from the fourth pallet configuration

Table 8 - Raw Data Results of the fourth pallet configuration

Fourth Pallet Configuration										
Component		Before			After			Difference		
Number	ID	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)
1	0	-39.8054736	44.74375	90.22347	-39.83989	43.752373	89.48555	0.0344164	0.991377	0.73792
2	1	-38.9593799	-37.26028	-25.21707	-34.00688	-34.00688	-25.42191	4.9524999	3.2534	0.20484
3	1	44.68881836	-7.359038	36.5745	44.183671	-5.96332	37.57721	0.5051474	1.395718	1.00271
4	2	12.68790039	-40.91854	-42.26223	14.558877	-42.508708	-44.31962	1.8709766	1.590168	2.05739
5	3	-10.9361035	5.9703613	-60.17891	-9.36958	4.91598632	-59.53703	1.5665235	1.054375	0.64188
6	4	22.60951172	31.030263	0	22.155361	32.0109668	0	0.4541507	0.980704	0
Overall Average								1.5639524	1.54429	0.774123

Time is calculated for pick and place from the moment the KUKA Robotic System rescans the component conveyor pallet (with a positive match) to the point where RSI is deactivated after placing the component (the point where the real-time control ends).

Table 9 - Timing results for configuration four

Number	ID	Objective	Time
		Learn Design	2.71s
1	0	**Pick* and *Place 1*	60.04s
2	0	**Pick* and *Place 2*	53.69s
3	1	**Pick* and *Place 3*	55.43s
4	2	**Pick* and *Place 4*	55.43s
5	3	**Pick* and *Place 5*	54.87s
6	4	**Pick* and *Place 6*	52.43s

*Recalculation of the component pallet is done before pickup for safety in case objects shift (time includes this).

4.4.6 Results from the fifth pallet configuration

Table 10 - Raw Results for the fifth configuration

Fifth Pallet Configuration										
Component		Before			After			Difference		
Number	ID	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)
1	0	-35.9079101	-14.0068994	72.24586	-32.965415	-16.22049	72.5592	2.9424951	2.213591	0.31334
2	0	-26.17765625	24.05897461	-62.10514	-25.078926	22.488701	-60.9802	1.0987303	1.570274	1.12494
3	1	29.11233398	-30.8868212	57.67874	29.564746	-32.49792	56.70877	0.452412	1.611099	0.96997
4	2	11.09193359	48.52538086	-81.56414	12.756025	51.216806	-81.4995	1.6640914	2.691425	0.06464
5	3	26.89583984	7.157392578	41.44555	26.558798	8.5044531	40.07979	0.3370418	1.347061	1.36576
6	4	-22.5604785	-55.539396	0	-18.33593	-56.89244	0	4.2245485	1.353044	0
Overall Average								1.7865532	1.797749	0.639775

Time is calculated for Pick & Place from the moment the KUKA Robotic System re-scans the component conveyor pallet (with a positive match) to the point where RSI is deactivated after placing the component (the point where the real-time control ends).

Table 11 - Timing results for configuration five

Number	ID	Objective	Time
		Learn Design	2.75s
1	0	**Pick* and *Place 1*	57.23s
2	0	**Pick* and *Place 2*	56.63s
3	1	**Pick* and *Place 3*	56.17s
4	2	**Pick* and *Place 4*	59.54s
5	3	**Pick* and *Place 5*	54.53s
6	4	**Pick* and *Place 6*	55.66s

*Recalculation of the component pallet is done before pickup for safety in case objects shift (time includes this).

4.4.7 Results from the sixth pallet configuration

Table 12 - Raw Results for the sixth configuration

Sixth Pallet Configuration										
Component		Before			After			Difference		
Number	ID	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)	X(mm)	Y(mm)	Angle(◦)
1	0	-8.0060839	-51.446206	0	-6.8543848	-53.436206	0	1.1516991	1.99	0
2	1	-37.06017	-23.756538	64.35756	-34.448706	-24.521201	66.09718	2.611464	0.764663	1.73962
3	2	30.322012	-34.50115	-45.0548	30.8072363	-35.558246	-43.77469	0.4852243	1.057096	1.2801
4	3	7.009609	50.475371	-80.26466	7.68599609	50.8198828	-80.31286	0.6763871	0.344512	0.0482
5	4	-39.225122	23.651748	-7.656044	-37.898833	22.8449218	-9.755692	1.326289	0.806826	2.099648
6	4	22.085498	16.824785	-83.3742	23.5709668	17.2879589	-83.99962	1.4854688	0.463174	0.62544
Overall Average								1.289422	0.904378	0.965501

Time is calculated for Pick & Place from the moment the KUKA Robotic System rescans the component conveyor pallet (with a positive match) to the point where RSI is deactivated after placing the component (the point where the real-time control ends).

Table 13 - Timing results for configuration six

Number	ID	Objective	Time
		Learn Design	2.51s
1	0	**Pick* and *Place 1*	56.86s
2	0	**Pick* and *Place 2*	57.08s
3	1	**Pick* and *Place 3*	55.14s
4	2	**Pick* and *Place 4*	59.56s
5	3	**Pick* and *Place 5*	52.69s
6	4	**Pick* and *Place 6*	57.14s

*Recalculation of the component pallet is done before pickup for safety in case objects shift (time includes this).

4.5 Results Discussion

The tables seen above give a clear representation of the system's capability in terms of accuracy and speed. Table 14 and 15 below are a summary of the results above to indicate overall accuracy and time.

Table 14 - Overall average accuracy results based on all builds

Build	X (mm)	Y(mm)	Angle(°)
1	1.7810555	1.391437	0.36042
2	2.6764228	1.134341	1.159303
3	1.3676975	1.421224	0.799562
4	1.5639524	1.54429	0.774123
5	1.7865532	1.797749	0.639775
6	1.289422	0.904378	0.965501
Overall Average	**1.7441839**	**1.365569**	**0.783114**

Table 15 - Overall average time results based on all builds

Build	Time Taken(s)
1	342.54
2	340.94
3	342
4	334.6
5	342.51
6	341.01
Overall Average	**340.6**

The system achieved an average of within 1.74mm of the X-axis and within 1.37mm on the Y-axis. The angle of placement of the components achieved a more desirable average of within 0.78°. The system was calibrated to the best possible level, but theoretically, if an even more accurate calibration is achieved, the system could certainly improve on these averages. Many factors could affect calibration, such as movement of the system, camera accuracy (focus, lens), etc.

The system's speed was also excellent, taking an average of 5.67 minutes to learn a design and rebuild it into the correct configuration. This is beneficial in environments where constant unique pick and placing tasks are required, saving reprogramming time. The system can immediately rebuild the seen design quickly, without needing any changes to the software. This time included the learning of the new design as well as picking and placing six components from the component conveyor over to the design conveyor based on the required configuration.

The KUKA Robotic system was limited to 50% of its maximum capable speed to maintain stability. At speeds greater than 50% in the project environment, vibrations from the quick movement of the KUKA Robotic System would influence calibration settings as the system was not fixed/bolted into the ground (not possible in test environment). Theoretically, the average build time achieved could be reduced if the system is implemented in a more stable environment. This would only increase the speed of movements between the component and design conveyor and not the speed of the real-time control (picking and placing components) as this was already set at the maximum stable speed.

As discussed in Chapter 3, RSI can only move the KUKA Robot a maximum distance within one IPOC cycle, otherwise the KUKA Robot responds with a jerking motion trying to move over a greater distance within a smaller time frame. The most optimal distance was determined using trial and error (to maintain stability at the highest possible moving distance). The optimal distance per IPOC-cycle was **0.05mm** within every **4ms**. This resulted in smooth movement when using real-time control on the system.

Overall, the system performed optimally and completed all given tasks without any issues. The system's accuracy and speed were impressive with "everyday" calibration in a changing environment. If the system could be calibrated to perfection, the results could be even more impressive.

4.6 Conclusion

This chapter discussed a series of tests that were undertaken on the system to test its capabilities. The system was challenged through various experiments that would require it to build a "Design" in the quickest, most efficient manner possible. The speed and accuracy were recorded under different circumstances with all these results being tabulated for an easier overview of the results obtained.

5 Chapter 5: Conclusion

This chapter summarizes the project and revisits the research goals and objectives of the study. This chapter also covers contributions made and implementation options using this system. Future work will also be discussed.

5.1 Summary

Chapter 1 introduced the project and gave background to the study's intentions, including the problem statement, the hypothesis and the specific objectives of the project. The proposed layout and system overview was also covered in this chapter.

Chapter 2 presented a literature study conducted to gain knowledge on flexible manufacturing systems while also reviewing concepts required to carry out the study, including required image processing techniques and hardware.

Chapter 3 covered methods and implementation options available and used within the study to achieve the goal. It covered more detail regarding concepts used and their implementation to create the proposed system.

Chapter 4 shows how the system was tested and how the objectives laid out in the beginning were achieved. Results presented in this chapter indicated how the system performed and the system's accuracy and speed were captured and analysed.

5.2 Research Goals and Objectives

The main aim and goal of the study was to use a visually aided system that can learn new product designs in real-time, allowing the KUKA Robotic System to adjust accordingly without needing manual reconfiguration. Real-time control of the KUKA with the RSI subsystem using data from the visually aided system can increase production output of newly introduced products, while also having alternative implementation options.

5.3 Contributions

The project delivered the following contributions:

5.3.1 Solution to new product introduction

As previously discussed, when new products are introduced on existing assembly lines, a reconfiguration is sometimes necessary to handle the new product's needs. The proposed system could adapt in real-time with image processing and real-time control of the KUKA Robotic System without requiring any manual reconfiguration via human intervention.

5.3.2 Smart Pick and Placer

A smart pick and placer could be implemented on a factory floor where changing items on an existing assembly system need to be relocated from one pallet to another (or to a different assembly line), i.e. there are no "fixed" locations of the incoming product. This could save costs when changing the uses of an existing line as no reconfiguration would be necessary in terms of pallet design or programming. (An open pallet design is used within this study, no fixed positions).

5.3.3 Implementation option for KUKA RSI

KUKA RSI is an extremely useful package as an extra option within a KUKA Robotic System, but unfortunately has had limited amount of studies involving it. This study presents a documented application for usage with this system, allowing future work within this line to be referenced to this study.

5.3.4 Engine for C# - RSI Integration with visual capabilities

During this study, custom software was developed by integrating C# with KUKA RSI as well as C# directly to Siemens S7-1200 PLC. This software also includes Open Source image processing applications such as Aforge.net and Accord.net. This software contains an all-in-one system that can be used for many different applications in the future.

5.4 Future Work

5.4.1 RGB-D Camera

The system currently cannot detect depth data so the Z-axis was fixed a certain distance from the pallet. This limited the system to picking up items of the same height. However, if an RGB-D camera was added for depth detection,, the system would be able to pick up any component of any height, as the system would know exactly how low to manoeuvre the KUKA Robotic System.

5.4.2 Improvement on Accuracy and Speed

As stated before, more accurate calibration would improve accuracy when picking and placing components. Another factor that could've lead to small inaccuracies was the height from which the component was released by the suction gripper. This was due to the entire system not being completely square as the surface of the floor was slightly skew. This resulted in having the component being released flush on one end of the pallet and slightly higher on the other end. Accurate calibration was not always possible (due to environmental conditions) but complete isolation of the project and more controlled factors (such as lighting, fixing the system to a solid surface) could be introduced to improve accuracy and speed and eliminate any differences entirely.

5.5 Conclusion

The aim of the study was to develop a self-learning product assembly system using a visually aided system. The system was required to learn a design on the design conveyor and then rebuild this "seen" design from a component conveyor catalogue. This required a real-time system implementation on the KUKA Robotic System, namely RSI. Digital Image processing was used to detect the design and process all calculations required for movement of the system to rebuild it.

Image processing was first done detecting various objects and calculating all relevant data for collecting and placing objects, such as centres of gravity, erosion techniques and angle of components. Positions were also calculated relative to the centre of the pallets (starting point for real-time control).

Real-time control directly into the KUKA Robotic System from C# was then set up for the system to move independently from the KRL Programming on the KUKA Robotic System.

C#, PLC and KUKA Robotic System communication was then established to enable accurate system flow such as movement of the components around the component conveyor system.

The system was then thoroughly tested for accuracy and speed. Six different pallet configurations were created (at random) for the system to rebuild using the components on the component conveyor. The main prerequisite for the design configurations was that the build design had to be constructed from components available to the system on the component conveyor. These results helped determine the accuracy and speed of the system and in turn, the success of the project.

There is limited to no other previous studies to draw upon with regards to this research project thus comparisons and benchmarks to similar systems could not be carried out.

References

[1] T. Raj, R. Shankar, and M. Suhaib, “A review of some issues and identification of some barriers in the implementation of FMS,” *Int. J. Flex. Manuf. Syst.*, vol. 19, no. 1, pp. 1–40, 2007.

[2] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, “Reconfigurable manufacturing systems: key to future manufacturing,” *J. Intell. Manuf.*, vol. 11, no. 4, pp. 403–419, 2000.

[3] Y. Koren and M. Shpitalni, “Design of reconfigurable manufacturing systems,” *J. Manuf. Syst.*, vol. 29, no. 4, pp. 130–141, Oct. 2010.

[4] M. Kaighobadi and K. Venkatesh, “Flexible Manufacturing Systems: An Overview,” *Int. J. Oper. Prod. Manag.*, vol. 14, no. 4, pp. 26–49, Apr. 1994.

[5] G. Chryssolouris, K. Efthymiou, N. Papakostas, D. Mourtzis, and A. Pagoropoulos, “Flexibility and complexity: is it a trade-off?,” *Int. J. Prod. Res.*, vol. 51, no. 23–24, pp. 6788–6802, Nov. 2013.

[6] R. J. Suey, “The Design and Operation of FMS; Flexible Manufacturing Systems,” *Int. J. Prod. Res.*, vol. 22, no. 4, pp. 725–726, Jul. 1984.

[7] Y. Koren *et al.*, “Reconfigurable Manufacturing Systems,” *CIRP Ann. - Manuf. Technol.*, vol. 48, no. 2, pp. 527–540, 1999.

[8] D. O’Sullivan, “Automated Production Lines.” National University of Ireland Galway, Galway, Ireland, 2011.

[9] Destaco, “End Effectors.” [Online]. Available: http://www.destaco.com/end-effectors.html. [Accessed: 03-Aug-2017].

[10] J. C. Whitaker, *The Electronics Handbook*, 1st Editio. CRC Press, 1996.

[11] P. D.-I. H. Frey, “Machine Vision, Lecture Notes.” Ulm, Germany, p. 1, 2014.

[12] S. Jayaraman, S. Esakkirajan, and T. Veerakumar, *Digital image processing*. New Delhi : Tata McGraw Hill Education, 2009.

[13] Olympus America Inc, “Basic Properties of Digital Images,” *Microscopy Resource Center*, 2012. [Online]. Available: http://www.olympusmicro.com/primer/digitalimaging/digitalimagebasics.html. [Accessed: 07-Aug-2017].

[14] R. Davies, *Computer and Machine Vision, 4th Edition Theory, Algorithms, Practicalities Opsylum*. Royal Holloway, University of London, UK: Academic Press, 2012.

[15] K. Briechle and U. D. Hanebeck, “Template matching using fast normalized cross correlation,” *Proc. SPIE*, vol. 4387, pp. 95–102, 2001.

[16] A. W. and E. W. R. Fisher, S. Perkins, “Erosion,” 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm. [Accessed: 21-Aug-2017].

[17] B. Jähne, *Digital Image Processing*. Springer Berlin Heidelberg, 2013.

[18] K. R. Gmbh, “KR C4 compact.” KUKA, Augsburg, Germany, 2014.

[19] K. R. Gmbh, “KUKA System Software 8.3 Operating Instructions.” KUKA, Augsburg, Germany, 2013.

[20] K. R. Gmbh, “KUKA.RobotSensorInterface 3.3.” KUKA, Augsburg, Germany, 2015.

[21] M. Edberg and P. Nyman, “Implementing Multi-Touch Screen for Real-time Control of a Robotic Cell,” Chalmers University of Technology, Göteborg, Sweden, 2010, 2010.

Scientific Outputs

- “Using a Vision System for Real-Time control of an Automated Adapting Robot System” presented by HJ Vermaak and L Rogers at the World Conference on Robotics and Artificial Intelligence 2018 (WCRAI2018), 26-27 July 2018 in Barcelona, Spain.
- “Automated programming robot component transfer system utilizing machine vision as detection interface” presented and included in proceedings by HJ Vermaak and L Rogers at the International Conference on Materials and Manufacturing (ICOMM 2018), 16-18 July 2018 in Melbourne, Australia.
- “Automated adapting component transfer system using real-time robot control within a KUKA RobotSensorInterface environment”, presented and included in proceedings by L Rogers and HJ Vermaak at the International Conferences on Electrical, Electronic and IT research (AFRICON 2017) conference, 18 – 20 September 2017, Cape Town.

Appendix A – PLC Report

Totally Integrated Automation Portal

MastersControl / RogersMainPLC [CPU 1214C DC/DC/DC] / Program blocks

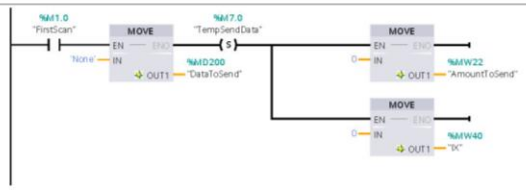
Main [OB1]

Main Properties							
General							
Name	Main	**Number**	1	**Type**	OB	**Language**	LAD
Numbering	automatic						
Information							
Title	"Main Program Sweep (Cycle)"	**Author**		**Comment**		**Family**	
Version	0.1	**User-defined ID**					

Main

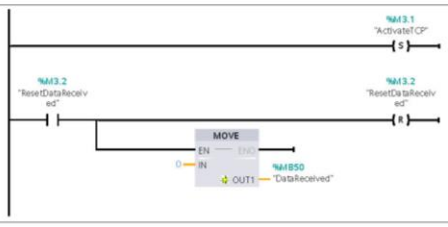
Name	Data type	Default value	Comment
Input			
Initial_Call	Bool		Initial call of this OB
Remanence	Bool		=True, if remanent data are available
Temp			
Constant			

Network 1: TCP Start



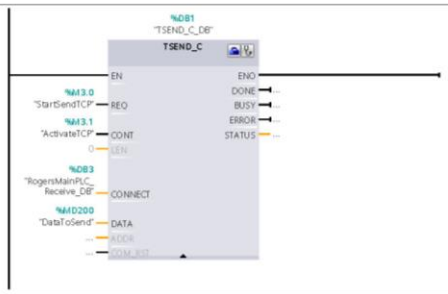
Symbol	Address	Type	Comment
"AmountToSend"	%MW22	Int	
"DataToSend"	%MD200	DWord	
"FirstScan"	%M1.0	Bool	
"IX"	%MW40	Int	
"TempSendData"	%M7.0	Bool	

Network 2: TCP Control Network



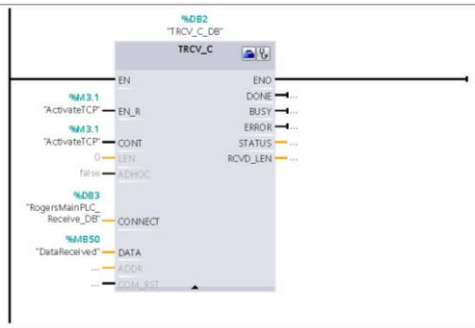
Symbol	Address	Type	Comment
"ActivateTCP"	%M3.1	Bool	
"DataReceived"	%MB50	Byte	
"ResetDataReceived"	%M3.2	Bool	

Network 3: Send Network



Symbol	Address	Type	Comment
"ActivateTCP"	%M3.1	Bool	
"DataToSend"	%MD200	DWord	
"StartSendTCP"	%M3.0	Bool	

Network 4: Receive Network

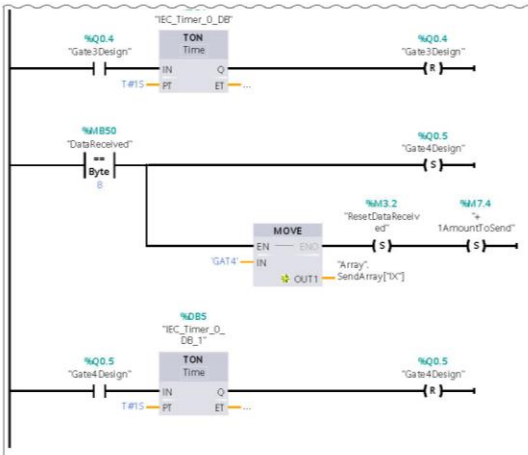


Symbol	Address	Type	Comment
"ActivateTCP"	%M3.1	Bool	
"DataReceived"	%MB50	Byte	

Network 5: Compare Network for Motors and Gates

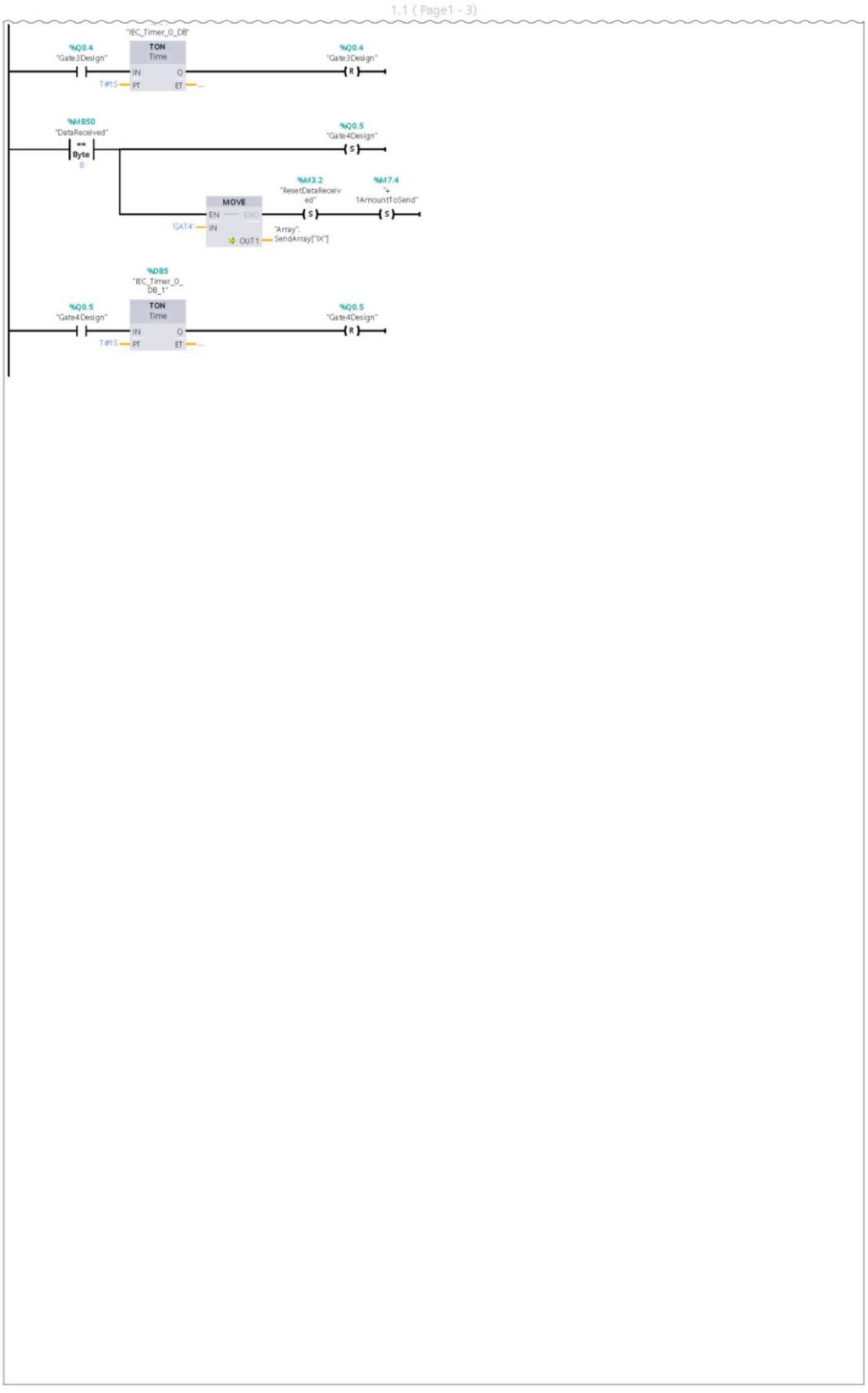
Network 5: Compare Network for Motors and Gates (2.1 / 2.1)

1.1 (Page1 - 3)



Totally Integrated Automation Portal		

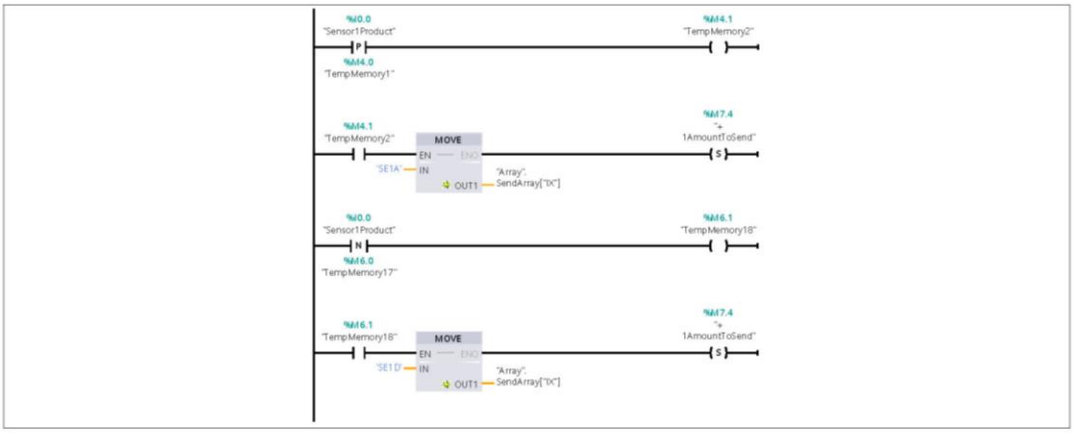
Network 5: Compare Network for Motors and Gates (2.1 / 2.1)



Totally Integrated Automation Portal		

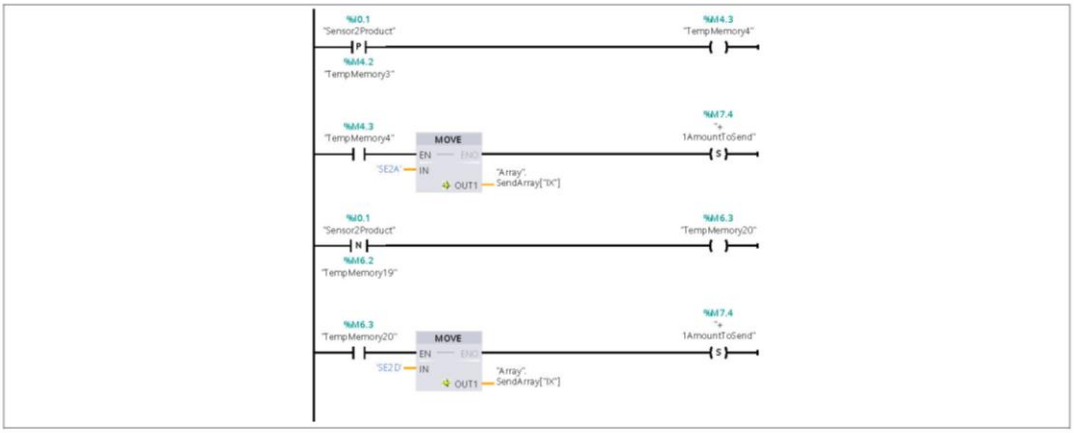
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"DataReceived"	%MB50	Byte	
"DesignMotors"	%Q0.1	Bool	
"Gate1Product"	%Q0.2	Bool	
"Gate2Product"	%Q0.3	Bool	
"Gate3Design"	%Q0.4	Bool	
"Gate4Design"	%Q0.5	Bool	
"IX"	%MW40	Int	
"ProductMotors"	%Q0.0	Bool	
"ResetDataReceived"	%M3.2	Bool	

Network 6: Sensor 1 Network (Product)



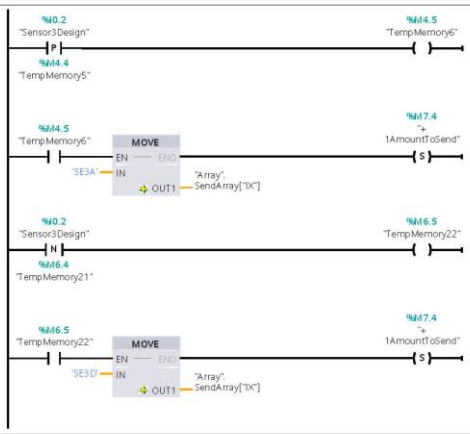
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"Sensor1Product"	%I0.0	Bool	
"TempMemory1"	%M4.0	Bool	
"TempMemory2"	%M4.1	Bool	
"TempMemory17"	%M6.0	Bool	
"TempMemory18"	%M6.1	Bool	

Network 7: Sensor 2 Network (Product)



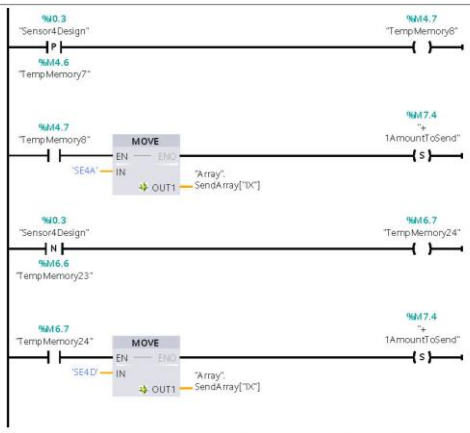
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"Sensor2Product"	%I0.1	Bool	
"TempMemory3"	%M4.2	Bool	
"TempMemory4"	%M4.3	Bool	
"TempMemory19"	%M6.2	Bool	
"TempMemory20"	%M6.3	Bool	

Network 8: Sensor 3 Network (Design)



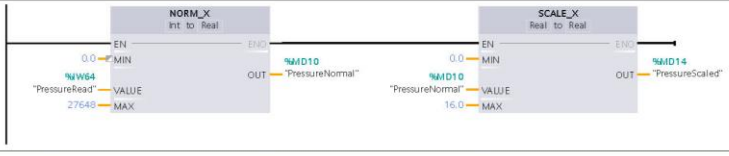
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"Sensor3Design"	%I0.2	Bool	
"TempMemory5"	%M4.4	Bool	
"TempMemory6"	%M4.5	Bool	
"TempMemory21"	%M6.4	Bool	
"TempMemory22"	%M6.5	Bool	

Network 9: Sensor 4 Network (Design)



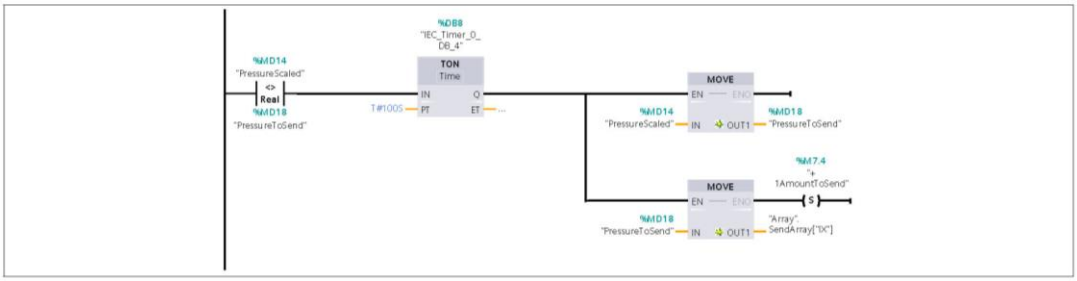
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"Sensor4Design"	%I0.3	Bool	
"TempMemory7"	%M4.6	Bool	
"TempMemory8"	%M4.7	Bool	
"TempMemory23"	%M6.6	Bool	
"TempMemory24"	%M6.7	Bool	

Network 10: Air Pressure Network



Symbol	Address	Type	Comment
"PressureNormal"	%MD10	Real	
"PressureRead"	%MW64	Word	
"PressureScaled"	%MD14	Real	

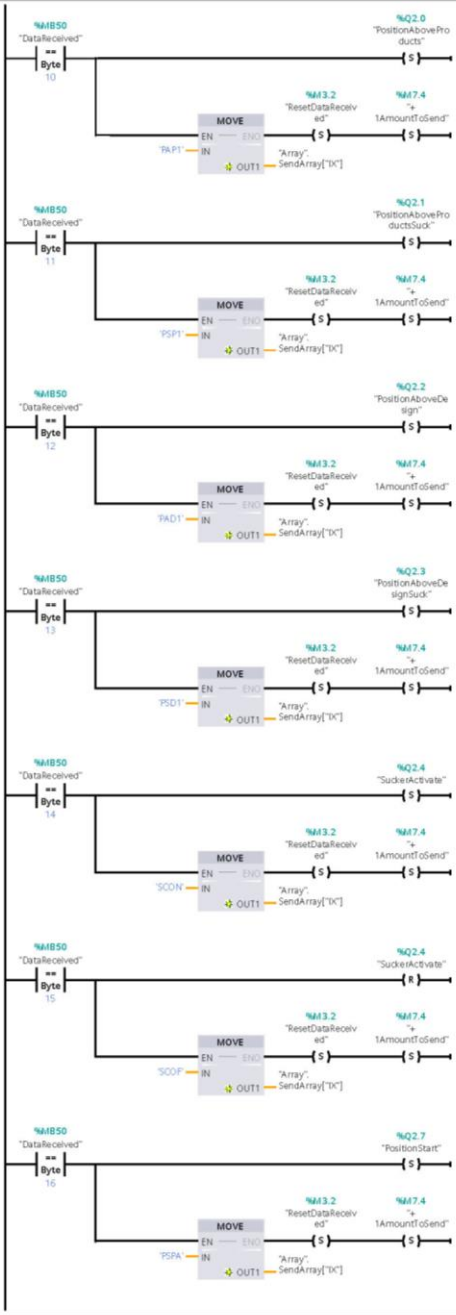
Network 11: Air Pressure Send



Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"PressureScaled"	%MD14	Real	
"PressureToSend"	%MD18	Real	

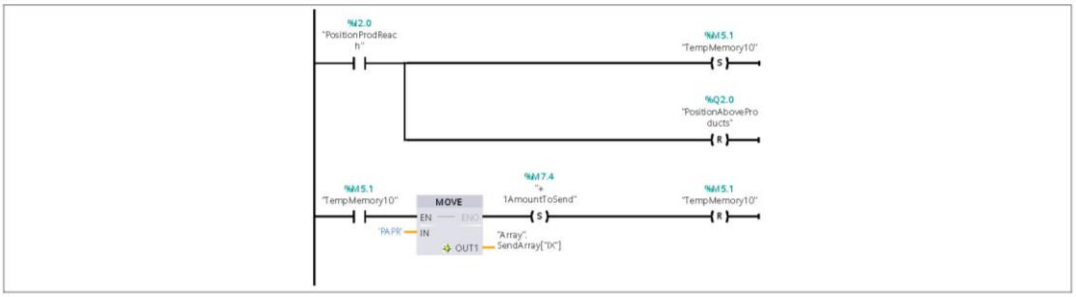
Network 12: KUKA Position Control

Totally Integrated Automation Portal		



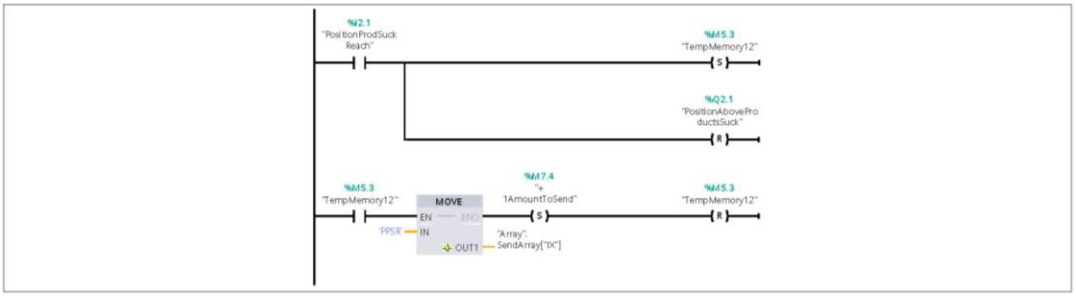
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"DataReceived"	%MB50	Byte	
"IX"	%MW40	Int	
"PositionAboveDesign"	%Q2.2	Bool	
"PositionAboveDesignSuck"	%Q2.3	Bool	
"PositionAboveProducts"	%Q2.0	Bool	
"PositionAboveProductsSuck"	%Q2.1	Bool	
"PositionStart"	%Q2.7	Bool	
"ResetDataReceived"	%M3.2	Bool	
"SuckerActivate"	%Q2.4	Bool	

Network 13: KUKA Position Feedback (Product)



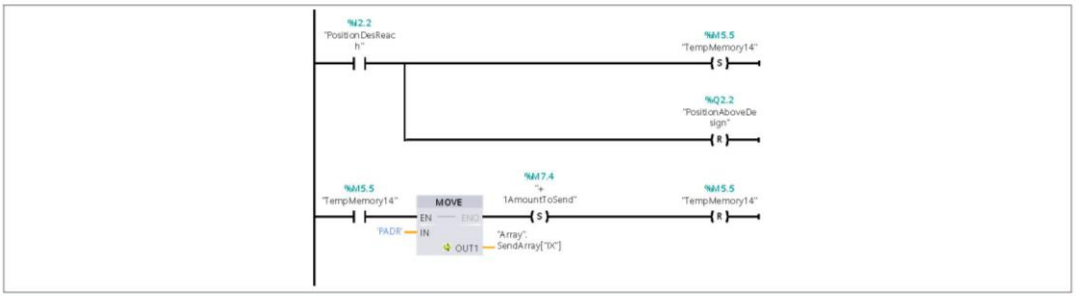
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"PositionAboveProducts"	%Q2.0	Bool	
"PositionProdReach"	%I2.0	Bool	
"TempMemory10"	%M5.1	Bool	

Network 14: KUKA Position Feedback (Product Suck)



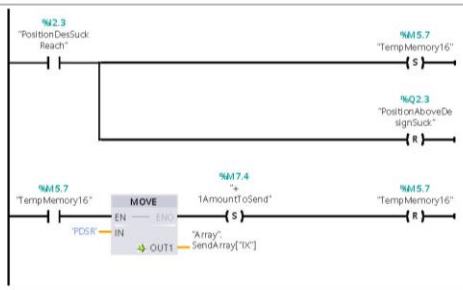
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"PositionAboveProductsSuck"	%Q2.1	Bool	
"PositionProdSuckReach"	%I2.1	Bool	
"TempMemory12"	%M5.3	Bool	

Network 15: KUKA Position Feedback (Design)



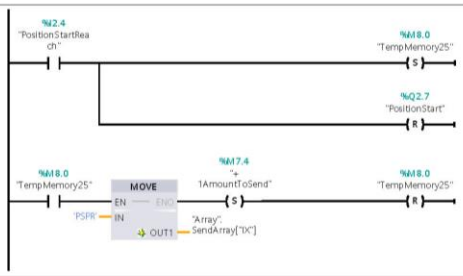
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"PositionAboveDesign"	%Q2.2	Bool	
"PositionDesReach"	%I2.2	Bool	
"TempMemory14"	%M5.5	Bool	

Network 16: KUKA Position Feedback (Design Suck)



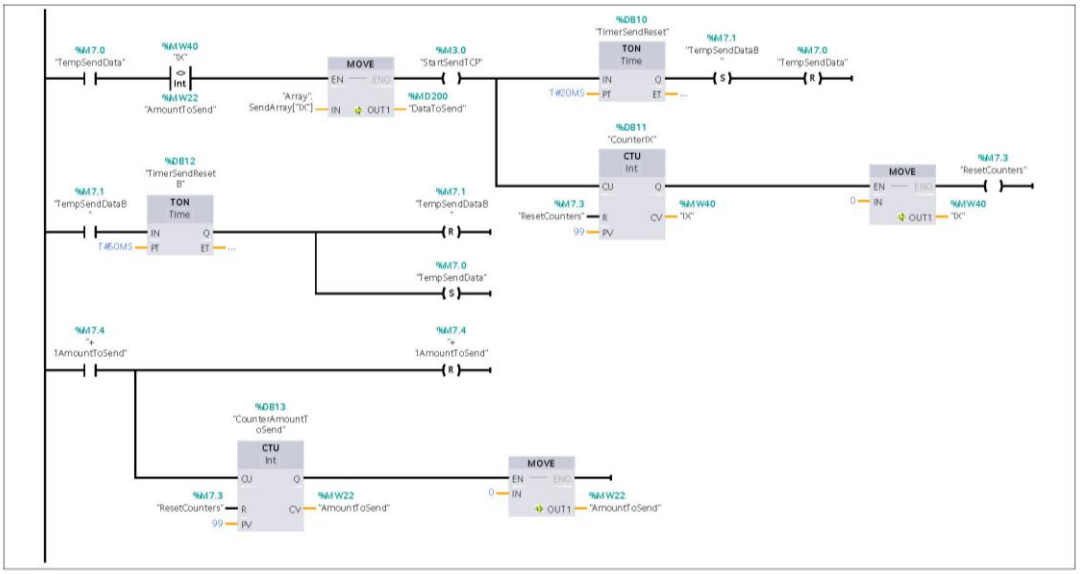
Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"PositionAboveDesignSuck"	%Q2.3	Bool	
"PositionDesSuckReach"	%I2.3	Bool	
"TempMemory16"	%M5.7	Bool	

Network 17: KUKA Position Feedback (Start Position)



Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"IX"	%MW40	Int	
"PositionStart"	%Q2.7	Bool	
"PositionStartReach"	%I2.4	Bool	
"TempMemory25"	%M8.0	Bool	

Network 18: TCP Message Queue Network

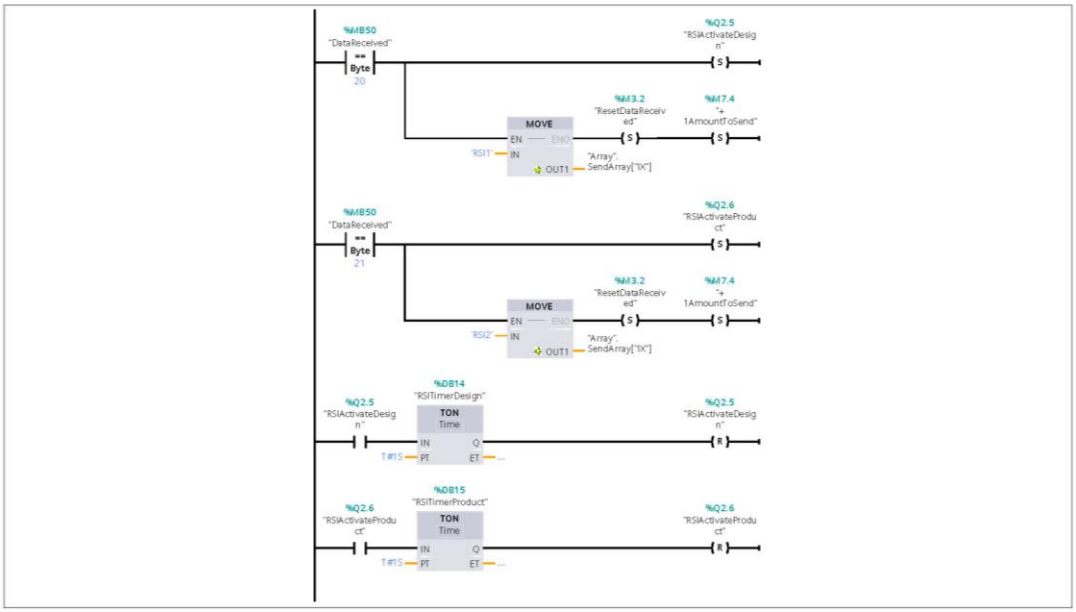


Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"AmountToSend"	%MW22	Int	
"Array".SendArray["IX"]		DWord	
"DataToSend"	%MD200	DWord	

Totally Integrated Automation Portal		

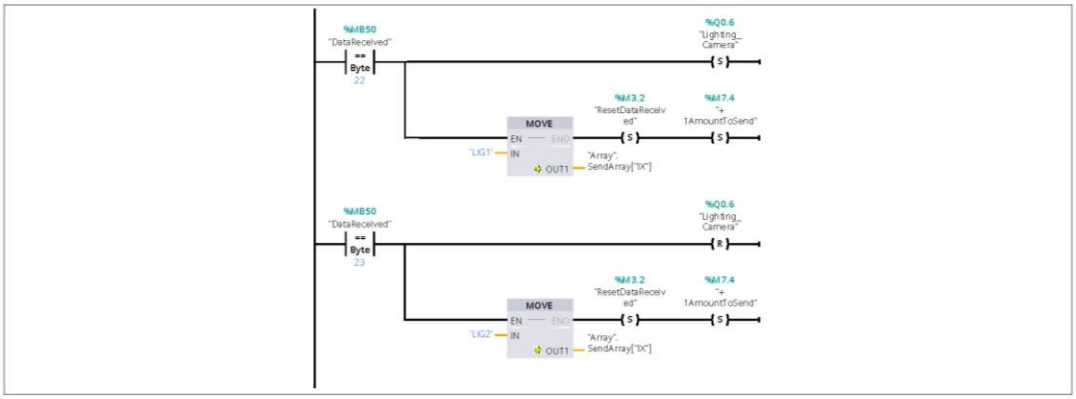
Symbol	Address	Type	Comment
"IX"	%MW40	Int	
"ResetCounters"	%M7.3	Bool	
"StartSendTCP"	%M3.0	Bool	
"TempSendData"	%M7.0	Bool	
"TempSendDataB"	%M7.1	Bool	

Network 19: RSI Control



Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"DataReceived"	%MB50	Byte	
"IX"	%MW40	Int	
"ResetDataReceived"	%M3.2	Bool	
"RSIActivateDesign"	%Q2.5	Bool	
"RSIActivateProduct"	%Q2.6	Bool	

Network 20: Light Control



Symbol	Address	Type	Comment
"+1AmountToSend"	%M7.4	Bool	
"Array".SendArray["IX"]		DWord	
"DataReceived"	%MB50	Byte	
"IX"	%MW40	Int	
"Lighting_Camera"	%Q0.6	Bool	
"ResetDataReceived"	%M3.2	Bool	

Appendix B – KUKA Code

```
DEF  MainMasterRSI ( )

; Declaration of RSI variables
DECL INT ret   ; Return value for RSI commands
DECL INT CONTID   ; ContainerID

  ;FOLD INI;%{PE}
    ;FOLD BASISTECH INI
      GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
      INTERRUPT ON 3
      BAS (#INITMOV,0 )
    ;ENDFOLD (BASISTECH INI)
    ;FOLD USER INI
      ;Make your modifications here

    ;ENDFOLD (USER INI)
  ;ENDFOLD (INI)

  ;FOLD PTP HOME  Vel= 100 % DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:100, 7:DEFAULT
    $BWDSTART = FALSE
    PDAT_ACT=PDEFAULT
    FDAT_ACT=FHOME
    BAS (#PTP_PARAMS,100 )
    $H_POS=XHOME
    PTP  XHOME
  ;ENDFOLD

;FOLD INITIALIZE RSI AND BITS
;=======================
;RESET OUT BITS!
;=======================
$OUT[99] = FALSE
$OUT[100] = FALSE
$OUT[101] = FALSE
$OUT[102] = FALSE
$OUT[103] = FALSE
$OUT[104] = FALSE

;============================
;CREATE RSI CONTEXT TO USE RSI!
;============================
ret = RSI_CREATE("RSI_Ethernet.rsi",CONTID,TRUE)
IF (ret <> RSIOK) THEN
  HALT
```

```
ENDIF
;==========================
;SET DEFAULT BASE AND TOOL
;==========================
$BASE = BASE_DATA[1]
$TOOL = TOOL_DATA[1]

;ENDFOLD

;FOLD START POSITION
;====================
;GO TO Start Position
;====================
StartPos()
;ENDFOLD

;START MAIN LOOP
LOOP

;FOLD Position Control

;FOLD Position PRODUCT
if ($IN[1] == TRUE) THEN
  $BASE = BASE_DATA[2]
  PTP {A1 -73.85, A2 -65.80, A3 75.99, A4 -4.95, A5 74.53, A6 -50.02}
  $OUT[100] = TRUE
  WAIT SEC 0.5
  $OUT[100] = FALSE
ENDIF
;ENDFOLD

;FOLD POSITION PRODUCT SUCK
if ($IN[2] == TRUE) THEN
  $BASE = BASE_DATA[2]
  PTP {A1 -92.47, A2 -51.55, A3 97.82, A4 0.50, A5 37.76, A6 -71.55}
  $OUT[101] = TRUE
  WAIT SEC 0.5
  $OUT[101] = FALSE
ENDIF
;ENDFOLD

;FOLD POSITION DESIGN
if ($IN[3] == TRUE) THEN
  $BASE = BASE_DATA[1]
  PTP {A1 105.70, A2 -69.12, A3 79.25, A4 -3.10, A5 73.54, A6 -50.20}
  $OUT[102] = TRUE
  WAIT SEC 0.5
  $OUT[102] = FALSE
```

```
ENDIF
;ENDFOLD

;FOLD DESIGN SUCK
if ($IN[4] == TRUE) THEN
  $BASE = BASE_DATA[1]
  PTP {A1 87.89, A2 -53.20, A3 100.70, A4 0.11, A5 37.57, A6 -70.51}
  $OUT[103] = TRUE
  WAIT SEC 0.5
  $OUT[103] = FALSE
ENDIF
;ENDFOLD

;FOLD START POS (RESET)
if ($IN[8] == TRUE) THEN
$BASE = BASE_DATA[1]
StartPos()
ENDIF
;ENDFOLD

;ENDFOLD

;FOLD SUCKER ACTIVATE/DEACTIVATE
if ($IN[5] == TRUE) THEN
  $OUT[104] = FALSE
  $OUT[105] = TRUE
  else
  $OUT[104] = TRUE
  $OUT[105] = FALSE
ENDIF
;ENDFOLD

;FOLD RSI Processing

;FOLD RSI for Design
;LET RSI TAKE OVER CONTROL WHEN TRUE (for Design Area) (BASE 1)
if ($IN[6] == TRUE) THEN

  $BASE = BASE_DATA[1]

  ret = RSI_ON(#RELATIVE)
  IF (ret <> RSIOK) THEN
   HALT
  ENDIF

  ; Sensor guided movement
  RSI_MOVECORR()
```

```
  ; Turn off RSI
  ret = RSI_OFF()
  IF (ret <> RSIOK) THEN
   HALT
  ENDIF

ENDIF
;ENDFOLD

;FOLD RSI FOR PRODUCT
;LET RSI TAKE OVER CONTROL WHEN TRUE (for PRODUCT Area) (BASE 2)
if ($IN[7] == TRUE) THEN

  $BASE = BASE_DATA[2]

  ret = RSI_ON(#RELATIVE)
  IF (ret <> RSIOK) THEN
   HALT
  ENDIF

  ; Sensor guided movement
  RSI_MOVECORR()

  ; Turn off RSI
  ret = RSI_OFF()
  IF (ret <> RSIOK) THEN
   HALT
  ENDIF

ENDIF
;ENDFOLD

;ENDFOLD

ENDLOOP


  ;FOLD PTP HOME  Vel= 100 % DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:100, 7:DEFAULT
    $BWDSTART = FALSE
    PDAT_ACT=PDEFAULT
    FDAT_ACT=FHOME
    BAS (#PTP_PARAMS,100 )
    $H_POS=XHOME
    PTP  XHOME
  ;ENDFOLD
END
```

```
DEF StartPos()
PTP {A1 0, A2 -90, A3 90, A4 0, A5 83, A6 -68}
$OUT[99] = TRUE
WAIT SEC 0.5
$OUT[99] = FALSE
END
```

Appendix C – C# Code and Videos

The code for the overall C# project, as well as videos of the project can be found on the CD.