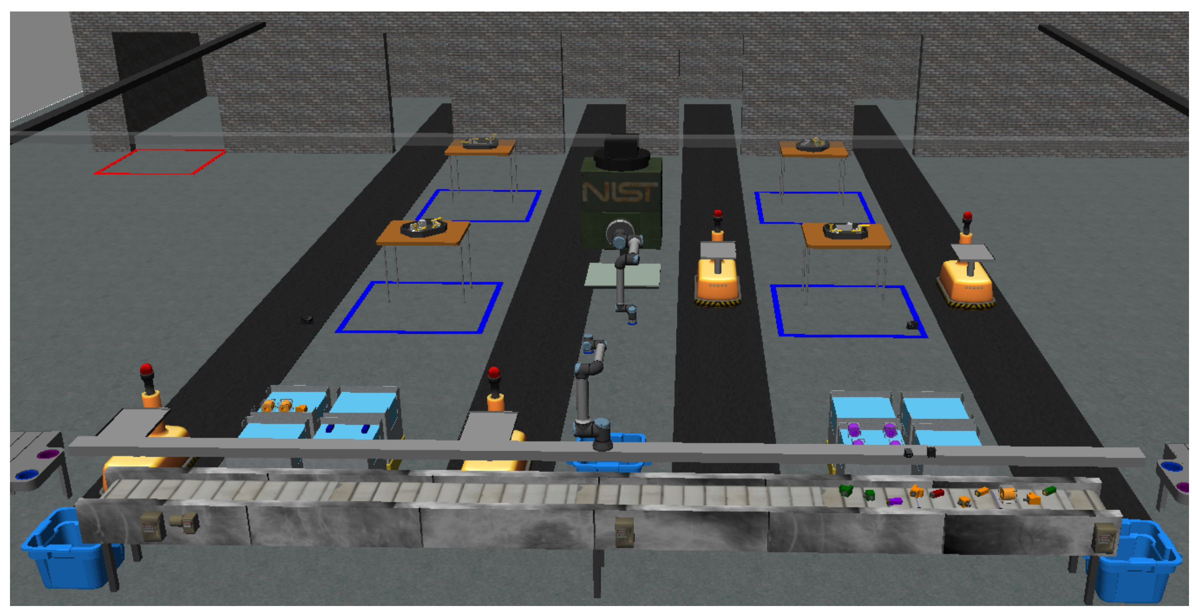
CHALMERS



DEVELOPING A ROS2 INFRASTRUCTURE AND CONTROL SYSTEM

An Approach of Utilizing Operation Runners and Convolutional Neural Networks for Autonomous Robots in a Simulated Factory

Bachelor's thesis in Electrical Engineering

ALI AL-BAYATI, LOUISE HASSLÖF, FREDRIK HÖGSTRÖM, AMANDA LEION, TAMERLAN TARAEV

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

DEGREE PROJECT REPORT 2025

DEVELOPING A ROS2 INFRASTRUCTURE AND CONTROL SYSTEM

An Approach of Utilizing Operation Runners and Convolutional Neural Networks for Autonomous Robots in a Simulated Factory

Ali Al-Bayati

Louise Hasslöf

Fredrik Högström

Amanda Leion

Tamerlan Taraev

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering

CHALMERS UNIVERSITY OF THECNOLOGY

Gothenburg, Sweden 2025

Developing a ROS2 Infrastructure and Control System - An Approach of Utilizing Operation Runners and Convolutional Neural Networks for Autonomous Robots in a Simulated Factory

©Ali Al-Bayati
Louise Hasslöf
Fredrik Högström
Amanda Leion
Tamerlan Taraev, 2025.

Supervisor: Endre Eros, Applied AI group, Chalmers Industriteknik
Examiner: Knut Åkesson, Department of Electrical Engineering, Chalmers University of Technology
Industrial collaborator: Atieh Hanna, Research and Technology Development, Volvo Group
Degree project report 2025

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96, Gothenburg
Telephone +46 31 772 1000

Cover: The Gazebo simulation of the factory the project worked with.
Typeset in LaTeX

Sammandrag

Ökad komplexitet inom industriell automation ställer allt högre krav på flexibla och adaptiva styrsystem som kan hantera uppgifter på ett dynamiskt sätt. Detta kandidatarbete bemöter dessa utmaningar genom att utveckla en ROS2-baserad infrastruktur och styrsystem för autonoma robotar i en simulerad fabriksmiljö, enligt anvisningarna för tävlingen Agile Robotics for Industrial Automation Competition (ARIAC). Den föreslagna lösningen integrerar en s.k. operation runner för uppgiftskoordinering samt ett Convolutional Neural Network (CNN) för realtidsklassificering av komponenter. Målsättningen är att optimera anpassningsförmåga och effektivitet inom ramen för en dynamisk tillverkningsmiljö.

Styrsystemet är uppbyggt kring ROS2:s ramverk och utnyttjar dess kommunikationsstrukturer såsom topics och services, för att möjliggöra integrerad hantering av automatiserade styrbara fordon (Automated Guided Vehicle (AGV):er), robotarmar och tävlingsinfrastruktur inklusive orderhantering. Operation runner:n ansvarar för en dynamisk koordinering av uppgifter genom att utvärdera pre- och postconditions för exekverbara operationer, vilket möjliggör styrning av flera robotenheter parallellt. För objektklassificering så utvecklades ett CNN, vilket var tränat på HSV-maskerad och bearbetad bilddata. Detta CNN uppnår en robust klassificering av komponenter trots variationer i orientering av komponenterna. Operation runnern visade sig effektiv i koordinering av AGV:erna och skalbarhet, medan CNN:et uppnådde god prestanda i realtid. Trots dessa framsteg kunde en fullständig integration mellan styrsystemet och de visuella klassificeringssystemet inte genomföras, huvudsakligen på grund av tidsbegränsningar och tekniska utmaningar.

Abstract

The increasing complexity of industrial automation demands agile and adaptive control systems capable of dynamic task execution. This thesis addresses these challenges by developing a ROS2-based infrastructure and control system for autonomous robots in a simulated factory environment, aligned with the Agile Robotics for Industrial Automation Competition (ARIAC) 2024. The proposed solution integrates an operation runner for task coordination and Convolutional Neural Network (CNN) for real-time part classification, aiming to optimize adaptability and efficiency in a dynamic manufacturing setting.

The control system leverages ROS2's communication framework such as topics and services to manage Automated Guided Vehicle (AGV), robotic arms, and competition infrastructure such as orders. The operation runner dynamically coordinate tasks by evaluating preconditions and postconditions of executable operations, enabling scalable control of multiple robots. A CNN, trained on HSV-masked and augmented image data, achieves robust part classification despite variations in orientation. The operation runner demonstrated success in AGV coordination and scalability, while the CNN demonstrates real-time capabilities with classification tasks. However, the integration of the control system and vision components together and into the ARIAC competition framework was not fully realized, mainly due to time constraints and technical challenges.

Acknowledgements

We are very grateful for the opportunity to work on this project and would like to extend our thanks to all parties involved. A special thanks to our supervisor, Endre Eros, without whom we would not be able to do this project. We would also like to thank Hanna Atieth at CampX, for showing us their testing lab and showing how our work can be applied outside of simulations. We would also like to extend our thanks to our examiner, Knut Åkesson, who has given us valuable feedback during our work to help us progress.

Ali Al-Bayati, Louise Hasslöf, Fredrik Högström
Amanda Leion, Tamerlan Taraev
Gothenburg, May 2025

Acronyms

AGV Automated Guided Vehicle. i, ii, 4–7, 19–21, 28–30, 35, 38, 46

AM Ariac Manager. 6, 15, 21, 42, 43

ARIAC Agile Robotics for Industrial Automation Competition. i, ii, 1, 3–6, 15, 20, 21, 23, 25, 31, 34, 35, 37, 38, 42, 44, 50

CCS Competitor Control System. 1, 6, 7, 15, 20, 28, 29, 42, 43

CNN Convolutional Neural Network. i, ii, 1, 3, 11–15, 21–24, 26–28, 31, 32, 34–36, 38

HSV Hue, Saturation, Value. i, ii, 1, 13, 25, 26, 38

ML Machine Learning. 12, 13

NIST National Institute of Standards and Technology. 3, 5, 6, 15, 51–53

ReLU Rectified linear unit activation function. 11, 27

Contents

Sammandrag i

Abstract ii

Acknowledgements iii

Acronyms iv

1. Introduction 1

1.1 Aim of the Project . . . 1

1.2 Relevance of the Project . . . 2

2. Background 3

2.1 The Agile Robotics for Industrial Automation Competition . . . 3

2.2 Main Software Used for the Project . . . 3

2.2.1 Robot Operating System 2 . . . 4

2.2.2 The ROS2 Message Passing Systems . . . 4

2.2.3 Gazebo . . . 5

2.2.4 The ARIAC Environment . . . 5

2.2.4.1 Control Flow - the Systems at Play in the ARIAC Environment . . . 6

2.2.4.2 Orders . . . 6

2.3 Control System . . . 7

2.3.1 Operations . . . 7

2.3.2 State . . . 10

2.3.3 Operation Runner 10
2.4 Convolutional Neural Network for Classification of Parts 11
2.4.1 Convolutional Neural Networks 11
2.4.2 Data Collection for a Convolutional Neural Network 12
2.4.3 Biases and Fairness in Data Collection and Machine learning 12
2.4.4 HSV Masking 13
2.4.5 Image Standardization: Cropping and Centering 14
2.4.6 Data Augmentation Through Image Transformations 14

3. Method **15**

3.1 Implementation Into the ARIAC Environment 15
3.2 Development of the Control System 15
3.2.1 Robot Instance Methods 15
3.2.2 Developing the Operation Runner 16
3.2.2.1 Emulation of Desired Behaviour 17
3.2.2.2 Simulation of Desired Behaviour 19
3.2.2.3 A Further Development of the Operation Runner 21
3.3 Vision - Enabling the Control System to See the Environment 21
3.3.1 Data Collection for the CNN 22
3.3.2 The Usage of OpenCv for Image Transformations for Expanding our Dataset . . 24
3.3.3 Data Processing for the CNN 25
3.3.4 The Usage of a Convolutional Neural Network 26

4. Results **28**

4.1 The Competitor Control system 28
4.1.1 The Operation Runner 28
4.1.2 State Management 29
4.1.3 Message and Action Handling 30
4.1.4 Order Management and Sequencing 31
4.1.5 Summary of Control System Results 31

4.2 Visual Input - Enabling Perception in the Control System 31
4.2.1 Vision System 31
4.2.1.1 Dataset and Augmentation 31
4.2.1.2 Data Processing Pipeline 32
4.2.1.3 CNN Training and Accuracy 32
4.2.1.4 System Behaviour and Real-Time Integration 32
4.2.1.5 Scalability and Modularity 32
4.2.2 Summary of Vision System Results 33

5. Discussion **34**
5.1 Analysis of the Control System 34
5.1.1 No Planner vs Planner 34
5.1.2 Scalability 35
5.2 Analysis of the Vision Components 35
5.2.1 Integrating the CNN to the ARIAC workspace 35
5.2.2 Optimization 36
5.3 Future Possibilities - Explorations of Unfinished Ideas 36

6. Conclusion **38**

1. Introduction

The future of automated systems is expected to rely heavily on complex resources, such as autonomously roaming robots capable of performing various tasks including material handling [1]. In a factory setting, this may involve industrial robots conducting operations such as kitting or assembly. However, industrial robots still face significant challenges when operating in unstructured and dynamic manufacturing environments, where object locations are not fixed, layouts may change frequently, and unexpected disruptions can occur [2]. These environments demand high levels of flexibility, which many current robotic systems lack. Typical difficulties include recovering from task failures, adapting to unforeseen changes in the environment, and executing fine motor tasks such as grasping with precision, all without external intervention.

Unstructured environments are characterized by frequent and unpredictable changes, for instance; shifting part locations, flexible production layouts, and interactions with human workers. To operate effectively in these settings, robots require more advanced control systems [1]. One common strategy is automated control, where the control system continuously makes decisions based on the current state of the system.

In this project, a combination of automated control and computer vision is used to address these challenges. By using visual sensor input to inform decision-making, the system gains a better understanding of its surroundings and can adapt its behaviour accordingly. Cameras and image processing techniques are used to detect part types, positions, and orientations. The vision system combines Hue, Saturation, Value (HSV)-based colour masking and a Convolutional Neural Network (CNN) to classify and localize parts. This supports both flexibility and autonomy in the control system, enabling the robot to perceive and respond to dynamic changes in real time.

1.1 Aim of the Project

The primary objective of the project is to develop an infrastructure and control system using ROS2 for a simulated factory in Gazebo. Specifically, the aim is to develop a Competitor Control System (CCS) compatible with theARIAC 2024 competition, focusing on the core of the competition - developing an agile infrastructure and control system for a simulated factory.

The project aims to solve the task by developing and employing an operation runner and a CNN, that together will work as the control system managing the simulated factory.

1.2 Relevance of the Project

The project is relevant due to the rising need and utilization of virtual commissioning and intelligent control algorithm based manufacturing systems. Manufacturing can be complex but utilizing virtual commissioning and intelligent control algorithms is a way to increase productivity. [3]. Virtually commissioning industrial robots is also important since it provides a safety and certainty aspect that the robots will act correctly in different situations - it is a way to ensure no problematic or error filled code will be deployed. A successfully implemented virtual commissioning environment lessens the risk of potentially damaging equipment and humans when taking the step from simulation to the real world [4].

Since 2018 up until 2024, there has been a 12% increase of operational robots and in 2023 the total number surpassed 4 million. [5] Given these numbers, its safe to say that there is an increasing demand for the control of the robots as well.

There is also a cost-aspect that can be said about the simulation of systems in this manner, as it can help reduce waste. For example, one could avoid errors of ordering components that are not ideal, and could also find ways to optimize energy usage before committing to a physical design. [6]

The reasons listed shows that our project is a great way to become more prepared for a future in automation- and production-engineering.

2. Background

The following sections present the technical background required for the project. Firstly, it covers the background necessary to understand the Agile Robotics for Industrial Automation Competition (ARIAC) environment and ROS2, the later sections cover the theories regarding the control system and the vision aspects such as Convolutional Neural Network (CNN) and data collection.

2.1 The Agile Robotics for Industrial Automation Competition

As mentioned, this project aims to develop a control system for ARIAC, which is a competition organized by the National Institute of Standards and Technology (NIST). [7] NIST organizes this competition annually and it is set in a virtual warehouse environment in Gazebo. Robots are to perform series of tasks, such as assembly and pick-and-place operations, while also trying to optimize factors such as efficiency, speed, and adaptability. The competition acts as a way for NIST to gain insight on how to further their research and methods to improve robot adaptability and agility, which means that the core focus of the aim of the competition is just that - improving industrial robot agility.

NIST divides their definition of agility for industrial robots as four main parts, 1) Failure identification and recovery, 2) Automated planing, 3) Not strictly structured environment, which entails automation fixtures and 4) Different types of robots working in a shared environment [8]. This is the definition used for industrial robot agility in this project.

The criteria that is evaluated in the competition are from a list of 8 scenarios, where one, several or none can be used in a trial [7]. The scenarios are detailed in section A.2 of the appendix, while the scoring for the scenarios is described in sections B.1-B.4. The trials may vary from year to year.

2.2 Main Software Used for the Project

The following sections provide a brief background on the main software that was used for the project, namely ROS2 and Gazebo. A brief explanation of the ARIAC environment is also presented.

2.2.1 Robot Operating System 2

ROS2, short for Robotic Operating System 2, is a software development platform for robotic applications, created by Open Robotics [9]. It is open source, which provides freedom and flexility to customize the usage of ROS2 for every specific project. ROS2 also provides tools, libraries and capabilities needed for developing robotics applications, as well as it being easy to integrate with existing software. ROS2 provides a message-passing system, that saves the developer time, as it manages the communication between nodes through an anonymous publisher-subscriber pattern [10], making it a useful tool.

One large advantage of using ROS2 is its big global community of developers - ROS2 is made by developers for developers [9]. Open robotics [10] describes ROS2 as a ”cornucopia of robot software”, meaning that ROS2 has a great supply of building blocks one might need for a robotics project - e.g. everything form drivers and algorithms to user interfaces.

2.2.2 The ROS2 Message Passing Systems

In ROS2, nodes have different ways of communicating with each other, among other things topics, services and actions. These three will briefly be explained in the following paragraphs.

Topics are used for one-way communication, where nodes publish messages with a unidirectional data flow. Any node that requires the information can receive it by subscribing to the topic[11]. The structure of a message sent over a topic follows a predefined format, ensuring that both publishers and subscribers can interpret the data correctly.

ARIAC uses many different topics in the environment to relay important information. One example is the **/ariac/sensors/camera/image** topic, where a camera sensor publishes images. Nodes can subscribe to this topic to receive and process the camera images. Another example is the **/ariac/orders**, where the ARIAC controller node publishes orders. The competitor’s robot logic nodes subscribe to this topic to determine which parts to pick, assemble, or deliver.

Services provide a way for nodes to communicate through a structured request-response mechanism. In contrast to topics, which only supports unidirectional data flow, a service has bidirectional interactions, by allowing a node to send a request and receive a corresponding response [12]. Services are used when a response is required or to trigger an event. Similarly to topics, the request and response must follow a shared message structure to ensure the client and server can communicate correctly.

For instance, in ARIAC framework, there is a service named **/ariac/move_agv**. After loading parts onto an Automated Guided Vehicle (AGV), the robot node calls the **/ariac/move_agv** service with a request to send AGV to the designated assembly station. The system then checks if the AGV is ready, and if so, initiates the movement and responds with a success message.

Actions is another communication type in ROS2 that is specifically designed and intended for long-duration tasks that require extended execution time [13]. Unlike services, actions support cancellations and can optionally provide real-time feedback during the execution of the task, rather than only reporting a final result upon completion. A typical example of an action is moving a robot from one

location to another. When an action is initiated, a goal for the action is communicated, such as for example move Robot A to position 2. Robot A then receive this goal, execute the corresponding movement, and reports back when it has reached the target position.

2.2.3 Gazebo

Gazebo is an open source robot simulator, which with great accuracy and efficiency can simulate different robotic systems and environments. Common systems and environments can for example be autonomous driving systems or industrial factory robots. Gazebo provides advanced robust physics engines, high quality graphical rendering, programmatic interfaces, and, most notably for this project, seamless integration with ROS2 [14].

2.2.4 The ARIAC Environment

NIST provides a simulation environment for the ARIAC competition, which the code of this project is based on. The competition simulates a complex industrial environment comprising of various different sensors, robots, and parts, all of which must be efficiently managed for creating a functional automated warehouse environment. Figure 2.1 illustrates the setup of the warehouse in Gazebo. The different tasks that can be preformed in the simulated factory is kitting and assembly tasks, which are described in detail in section 2.2.4.2. The following sections outlines the key information required for designing the control system. For a comprehensive overview of the components used in the environment, like the sensors, parts, and robots, see section A.3 in Appendix A.

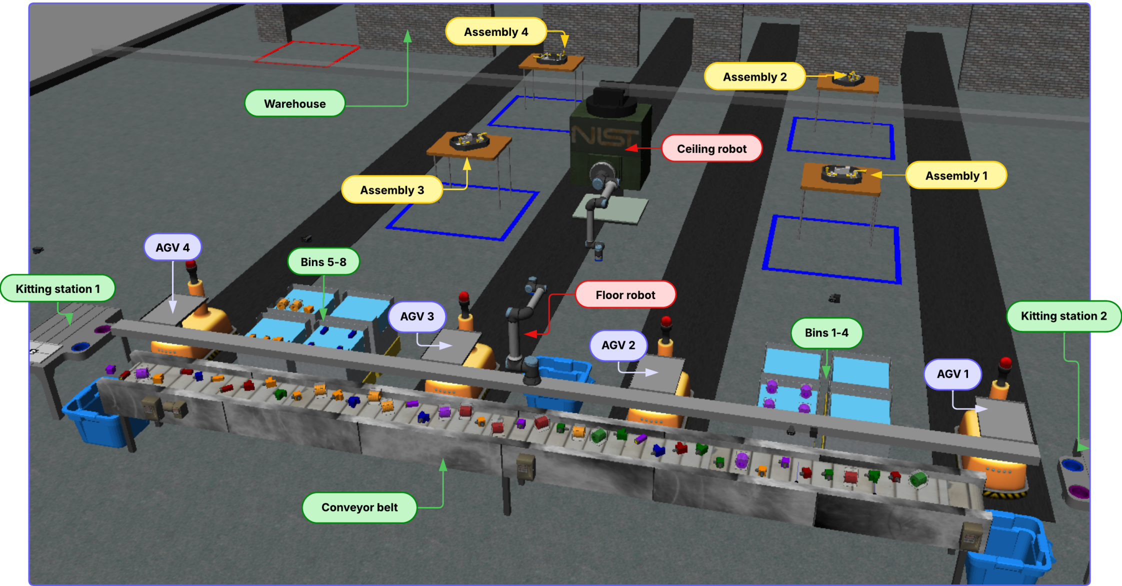


Figure 2.1: Image showing an overview of the ARIAC environment, featuring a conveyor belt filled with parts, a robot arm mounted on a rail parallel to the conveyor belt (floor robot), eight bins (a few of them containing parts), four AGV, a ceiling robot, two kitting stations, and four assembly stations.

2.2.4.1 Control Flow - the Systems at Play in the ARIAC Environment

Within the ARIAC environment, two main systems are at play - the Ariac Manager (AM) and the Competitor Control System (CCS). The AM primarily simulates the environment in which the program will run [15], as well as it manages the ROS2 interface. A flowchart illustrating the logic of the AM, and its interaction with the CCS is provided in Figure A.1 in Appendix A. The AM is not implemented by the competitors and is provided by NIST.

The second main system, as previously mentioned, is the CCS, and the primary function of the CCS is to communicate with the ARIAC system through ROS2 interfaces, like messages, services, and actions, to receive and execute orders from the AM [15]. The CCS await different orders from the AM and once an order has been received, the CCS is responsible for carrying out the specific task required to complete the order. Orders are in this scenario, different pick-and-place tasks, and these are explained more in the next section, 2.2.4.2. The CCS works on completing all the orders it has been sent until all orders are fulfilled, after which the CCS signals the AM to terminate the program.

2.2.4.2 Orders

In the ARIAC environment, an order is a pick-and-place task that includes a set of specific requirements that must be met to complete successfully[16]. There are two main types of pick-and-place tasks; kitting and assembly. Additionally, there is a third type of order, which is a combination of the two aforementioned tasks.

The **kitting task** is a process of gathering different parts and placing them together on a kit tray, see section A.3 for more information about the parts used and A.5 for more information about the kit trays. The role of the CCS in a kitting task is to:

1. Locate the kit tray with the specified ID, on one of the two kitting stations.
2. Pick, place, and lock the tray on the specified AGV.
3. Pick and place desired parts onto the kit tray in the correct, specified quadrant of the tray.
4. Perform a quality check and fix possible issues, like for example checking for broken, missing, or incorrect parts.
5. Direct the AGV to the warehouse.
6. Submit order.

The **Assembly task** is a process of installing parts on a fixed insert - a container that holds parts, see section A.5 for more information. For this task, the parts that will be assembled will be spawned on the AGVs. The role of the CCS in an assembly task is to:

1. Lock the AGV trays.
2. Move the AGV to its designated assembly station.

3. Locate positions of parts on the kit tray.
4. Use the ceiling robot to pick up the parts and install the parts into the insert.
5. Submit order.

As previously mentioned, a **Combined task** is a task that is a combination of a kitting and an assembly task, meaning that the task requires both kitting and assembly. During a combined task, the CCS is supposed to first perform a kitting task and then proceed with an assembly task.

An order includes a few specifications; ID, type, priority, and announcement. These are used by CCS for the completion of the orders. During the execution of orders different challenges can arise, like for example a robot dropping a part, these are specified in section A.2 in appendix A.

2.3 Control System

Modern day automation form the backbone of industrial production, with its use seeing further improvements due to constant innovation, big data and improved efficiency [17]. To manage the complexity of automation, a control system is essential for coordinating tasks.

In the context of agile robotics, the control system is required to coordinate tasks and dynamically respond to changing environments [18]. At the same time the control system is expected to be flexible, scaleable and reconfigurable in order to constantly tackle more complex situations [18]. The following sections describe the theory for the individual parts of the control system developed in this project.

2.3.1 Operations

One of the fundamental components of this project's control system is operations. Our approach are heavily inspired by Bengtsson, Bergagard, Thorstensson, *et al.* [19] and Dahl, Erős, Bengtsson, *et al.* [20] when defining our operations. The following sections describes the essential elements of these operations.

A State can be defined as a set of tuples containing different types of information about a system. It can be described as $S = \{\langle vi, xi \rangle\}$, where vi is a variable with domain Vi and $xi \in Vi$ is a value. It could for instance include the position of an AGV or the status of a robot's grip on an object. For further details, refer to section 2.3.2.

A **guard** is a boolean function over a state. It can be defined as $g : S \rightarrow \{True, False\}$ where S is the state.

An **action** initiates state changes and initializes/resets the resources an operation requires/required to execute. Actions can range from for example modifying a state variable to invoking a ROS2 action to move a robot. It can be described as $a : S \rightarrow S$.

Transitions consists of a guard and an action. As previously mentioned, a guard is a boolean function, meaning evaluating the expression will yield a result that is either true or false. Transitions are enabled

only when the guard evaluates to true, allowing the an action to be triggered. A transition can be used to update a state, for example, $T : S = \{a = 1, b = 0\} \rightarrow S = \{a = 0, b = 1\}$, where T is the transition and S is a state containing two variables, a and b.

Preconditions and **postconditions** are both transitions, meaning each contains a guard and an action. However, they serve distinct purposes; a precondition is for initialising an operation, while a postcondition is used to conclude the operation.

An **operation** represents a task that requires time to execute, and consists of a precondition and a postcondition. When executed, it performs a low-level goal or action. The guard of the precondition evaluates whether an operation can start, while the action updates the state to ensure the operation can be completed without interference from for example other operations. Depending on the guards, multiple operations can be run simultaneously, and it may thereby be necessary to add actions that update state variables containing information about for example whether or not a robot is available or busy. The postcondition's guard is then evaluated to determine if the operation has finished, meaning if the guard yields true, the action belonging to the postcondition will be executed and the operation will be completed. Figure 2.2, illustrates a simplified flowchart of the logic of a precondition and a postcondition working together.

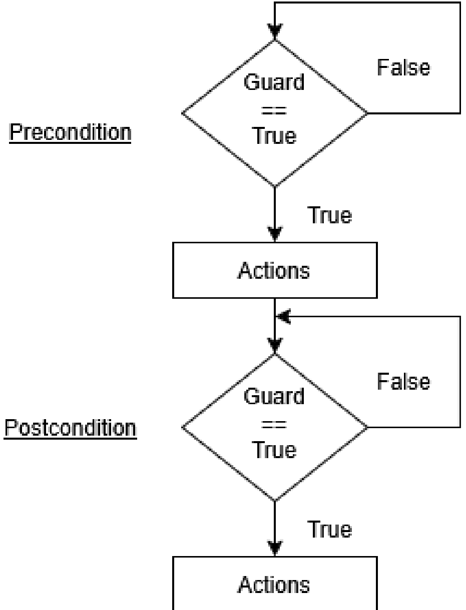


Figure 2.2: A simplified flowchart of the logic for pre- and postconditions, when implemented together.

As previously mentioned, operations usually represent tasks that require time to complete. In contrast to some other ways of modeling, using a postcondition makes it very easy to determine when an operation has finished, since the postcondition has a guard that only evaluates as true when the operation is completed. The actions of the postcondition can be used to update estimated state variables, that cannot be updated during the execution of the precondition. Some state changes should only occur after the operation has been completed, such as for example an estimation of an

object's new position after it has been moved, since the state determines what operations can be executed.

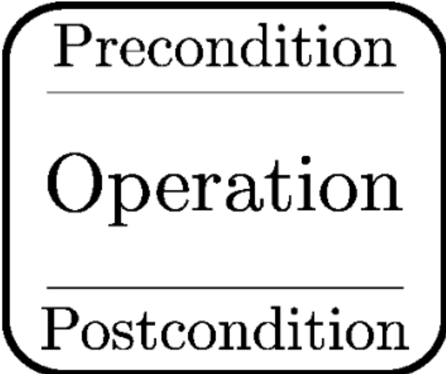


Figure 2.3: An image visualizing an operation, which include two transitions: a precondition and a postcondition, that determines when the operation can start or complete.[19]. Used with permission.

A visualisation of an operation can be seen in Figure 2.3, and an example of how to write and create operations in Python can be seen in Listing 2.1.

Listing 2.1: A code snippet demonstrating how a for-loop can be utilized to create multiple simple operations with minimal lines of code. The operations generated in the code snippet are rudimentary; in practical applications, additional state variables would likely be incorporated into the guards of the transitions. This code is intended solely for visualization puropses.

```
1  for i in range(10):
2      ops[f"moveToPos{i}"] = Operation(
3          name = f"moveToPos{i}",
4          precondition=Transition("pre",
5              #guard:
6              g(f"robotAllowedToMoveLocation && robotPos != {i} "),
7              #actions:
8              a(f"!robotAllowedToMoveLocation, robotMoveTo <- {i}")
9          ),
10         postcondition=Transition("post",
11             #guard:
12             g(f"robotPos == {i}"),
13             #actions:
14             a(f"robotAllowedToMoveLocation, robotMoveTo <- {None}")
15         )
16     )
```

Alternative operation outcomes are feasible with operations through inclusion of multiple postconditions. Multiple postconditions allows an operation to for example have one postcondition for successful task completion and one for failure. Consequently, the structure of operations thereby accommodates the integration of failure handling, which is highly relevant for a control system.

Operations are used as the building blocks which are executed towards reaching a goal. The goal originates from an order within the competition, and to complete the goal, various different operations will be executed until the goal has been reached. According to Bengtsson, Bergagard, Thorstensson, *et al.* [19], operations are self-contained and only encompass what is relevant for the operations' execution, meaning one can implement the usage of operations in multiple ways, such as for example with a sequence planner or an operation runner.

2.3.2 State

In an autonomous environment where numerous variables interact, it is crucial for the system to evaluate conditions that determine when tasks can be coordinated and executed. Rather than explicitly defining discrete states as in a deterministic automaton, the system operates by evaluating predicates (guards) - logical expressions that describe properties of the current situation, such as robot status, task progress or sensor feedback.

These predicates represent state space, conditions under which certain guards are valid. When the current situation satisfies the predicate, i.e. when the guard evaluates to true, the system is considered to be within a certain state space, allowing specific operations to be executed. An operation can only start if its guard condition is satisfied, enabling the system to transition from one state to another through the execution of operations.

A pick-and-place operation for example may have a guard requiring that a part is detected and the gripper is empty. Only when both of these conditions are true can the operation start. If the part is missing or the gripper is already holding an object, the operation will not trigger, allowing the system to wait or attempt alternative actions. This approach enables the system to adapt dynamically to changes and sensor inputs without the need to define every possible explicit state in advance.

2.3.3 Operation Runner

The operation runner is the component of the control system responsible for executing and coordinating operations, which is done in accordance with the needs of the control system. The operation runner executes operations by continuously evaluating all the guards for the different operations. Additionally, it ensures that each operation properly transitions through its starting, executing, and finishing phase while respecting its logical conditions, such as preconditions (guards) and postconditions (secondary guards) as described above. As operations change the state of the system, the runner reacts and updates which operations can run next based on the current state. Theoretically this makes the system flexible and able to handle different tasks without needing fixed sequences, since every operation is being evaluated and the current state is dynamically changed every time an action is executed.

2.4 Convolutional Neural Network for Classification of Parts

These following sections covers a short background on Convolutional Neural Network (CNN)s and the data collection required for it.

2.4.1 Convolutional Neural Networks

The Convolutional Neural Network (CNN) is a method used for image classification. CNNs loosely mimic the way the human brain processes visual information by using layers of interconnected neurons, hence the name 'neural network'. In these networks, neurons are connected across multiple layers, forming a hierarchical structure that enables feature learning from images.

The basic architecture starts with an input layer where image data is received. This is followed by a sequence of hidden layers and ends in an output layer, where each neuron corresponds to a possible class. The values in the output layer represent the probability of the input image belonging to each class. This can be achieved using a Softmax activation function, which transforms the raw output scores into normalized probability values summing to one. During training, the network learns by adjusting the *weights* - numerical values associated with each connection, to minimize the classification error.

Two central types of layers in a CNN are **convolutional layers** and **fully connected (linear) layers**. Convolutional layers apply small, trainable filters that slide across the input image (or feature map) to extract local patterns such as edges or textures. These filters preserve spatial relationships and reduce the number of parameters compared to traditional fully connected layers. Each filter is also known as a *kernel*, which is a small matrix of learnable weights. At each step, the kernel is applied to a region of the input image called a window, and the result of this operation is a single value in the output feature map. As the kernel slides across the image, it learns to detect patterns such as edges, textures, or shapes by adjusting its weights during training. In contrast, fully connected layers called linear layers flatten the data and connect every input neuron to every output neuron, which is typically used at the final stage of the network for classification.

An important part of this process is the use of **activation functions**, which introduce non-linearity. Without them, the network would only be able to learn linear transformations. One of the most commonly used activation functions is the Rectified linear unit activation function (ReLU), which outputs zero for negative inputs and keeps positive values unchanged. Other functions such as Sigmoid and Tanh exist, but ReLU is preferred due to its simplicity and strong performance. [21]

Another key component is **pooling**, especially **max pooling**, which is used to reduce the spatial dimensions of feature maps. Max pooling operates by taking the maximum value in a small region (e.g., 2×2 pixel window), effectively summarizing features while reducing computational complexity. Pooling helps the network generalize better and become more robust to minor changes in the input image.

Figure 2.4: Visualization of the ReLU activation function.

To further promote generalization and prevent overfitting, **dropout** is commonly used. Dropout works by randomly deactivating a subset of neurons during each training iteration. This prevents the network from relying too heavily on specific neurons and forces it to learn more distributed and robust features.

2.4.2 Data Collection for a Convolutional Neural Network

A CNN requires a dataset for learning, which typically are divided into two main subsets; training data and test data [22]. The training data is used to train the model during the model's development, while the test data is employed to evaluate and verify the model after the model has been developed. Additionally, a part of the training data may also be used as validation data during the model development to optimize parameters. This entails that a lot of data is needed for the development of the CNN. The division into the two main subsets occurs during the preprocessing of the data, and is crucial, as validating the model on the same data it has been trained on can mislead the assessment of the effectiveness and accuracy of the model.

2.4.3 Biases and Fairness in Data Collection and Machine learning

Here the term Machine Learning (ML) will be used, not to be confused with CNN; CNN is a subset of ML [23], meaning ML is a broader term that encloses different models, not only CNN.

Data is tightly tied to the functionality of a model, and if the data contains biases, the model will inevitably learn these biases as well [24]. The negative consequences of a biased model can vary in magnitude depending on the application of the model, but regardless, biased data and a biased model should be avoided.

There are numerous types of biases in ML [22], each biases occurring throughout the different stages of the ML life cycle. For instance, learning bias and evaluation bias arise early on in the life cycle, and they are tied to the data being used for the ML. Suresh and Guttag [22] emphasise the importance of understanding the implications of each stage in the data generation process, since this can reveal meaningful ways to prevent and reveal and identify the consequences of biased data. While not every

problem in ML is connected to the data, biased data can and will impact the ML output, making it crucial to consider how the data can introduce bias to the ML and what consequences it can give.

There are many biases to avoid, the following paragraphs briefly describe a few of the biases most relevant to this project.

Representation bias arise during the data collection and data processing phases [22], and it occurs when one or more subgroups are underrepresented. Consequently, a CNN might infer, due to the underrepresentation, that one thing is significantly less common than another, even tough they occur with the same frequency in reality.

One method to counter representation bias is to preform dataset balancing [25], which involves removal of certain data from the dataset. Dataset balancing can enhance accuracy for underrepresented groups within the dataset. However, depending on the method used, varying amounts of data will be removed from the dataset, resulting in a loss of data in the process. Therefore, it is highly beneficial to ensure that no subgroups are underrepresented when collecting data.

Sampling bias occurs when a subgroup or multiple subgroups are not sampled randomly [24]. This results in a model identifying trends in the data that cannot be generalized; the trends fit the sample, but not for a broader population.

After a model has been trained, it is tested on new data - test data. Evaluation bias can occur when the training data is not representative of the population[24] or if the metrics used to measure performance is inappropriately selected. Therefore, it is crucial to carefully consider the test data and performance measurement metrics to avoid biases.

2.4.4 HSV Masking

A standardized dataset can be created in several ways, a possible technical pre-processing step is colour-based masking in the HSV colour space. HSV, which stands for *Hue*, *Saturation*, and *Value*, is a cylindrical colour representation model that separates image intensity from colour information.

- **Hue** represents the type of colour, measured in degrees around the colour wheel (e.g., red, green, blue).
- **Saturation** describes the intensity or purity of the colour, from dull (grayish) to vivid.
- **Value** indicates the brightness of the colour, from dark to bright.

Unlike the traditional RGB model, which mixes red, green, and blue light values directly, HSV better reflects how humans perceive and describe colour, making it more suitable for tasks like colour segmentation under varying lighting conditions. [26]

An HSV mask is a binary image where each pixel is either white (included) or black (excluded), based on whether it falls within a predefined HSV range. This allows for isolating objects of a specific colour, simplifying subsequent processing steps such as contour detection and cropping.

2.4.5 Image Standardization: Cropping and Centering

In image classification tasks, standardization of input data plays a crucial role in improving model performance. This includes maintaining consistency in aspects such as colour distribution, brightness, orientation, centring, and spatial dimensions.

Two common techniques for spatial standardization are **centring** and **cropping**. The centre can be identified by the primary object in an image using contour detection, a process that locates continuous boundaries in a binary mask. Once the main object is detected, its approximate centre can be computed using image moments - a mathematical approach for calculating the centroid based on pixel intensity distributions.

Cropping is then used to extract a fixed-size region around this centre, ensuring the object is aligned similarly across all training images. If the object is located near the edge of the image, padding techniques can be applied to maintain the desired crop dimensions. These preprocessing steps reduce input variability and improve the stability and accuracy of convolutional neural networks.

2.4.6 Data Augmentation Through Image Transformations

To train a CNN effectively, it is important to ensure that the data is varied, as the visual appearance of objects in real-world conditions will rarely be identical in every instance. One way to introduce such variation is through the use of image transformations that simulate different types of noise. Common examples include blurring techniques, which reduce image sharpness, as well as geometric transformations that alter the shape or position of objects within the image. Here follows a brief explanation of some subtypes that can be used.

- **Geometric transformations:** These include rotation, shearing, translation, and resizing. They simulate changes in viewpoint, object alignment, and spatial shifts that may occur in real-world environments.
- **Blurring techniques:** Gaussian and median blur can be used to simulate motion blur or defocused images, mimicking situations where the camera lens might be dirty or out of focus.
- **Morphological operations:** Dilation and erosion expand or shrink object boundaries, simulating visual noise or partial occlusion. Opening (erosion followed by dilation) and closing (dilation followed by erosion) help in noise removal and shape refinement.

These methods, known collectively as data augmentation, help improve the model's generalization ability and are commonly used to enhance the robustness of machine learning systems - particularly in vision-based tasks like part classification.

3. Method

This chapter describes the steps taken during the course of the project. It covers the applications, choices, and processes used in the development work, as well as how the work was structured and carried out. The aim is to give the reader a clear understanding of how the project was implemented in practice.

3.1 Implementation Into the ARIAC Environment

To run implementations within the Agile Robotics for Industrial Automation Competition (ARIAC) competition and environment, a correctly formatted competitor package was required. This package follows a predefined structure specified by National Institute of Standards and Technology (NIST)[27]. The environment operates through a two-terminal setup. In the first terminal, the ARIAC environment is launched, initializing the simulation infrastructure. In the second terminal, a custom launch file is executed to start the competition as well as anything deemed necessary by the competitors, in this project it is the control system and the Convolutional Neural Network (CNN) with their respective components.

3.2 Development of the Control System

The Competitor Control System (CCS) is implemented in ROS2 using Python and serves as the central coordinator for all robots and sensor actions during the competition. The CCS also functions as a script that manages the competition, it reads incoming orders by subscribing to relevant topics, and starts or ends the competition by publishing the appropriate signals to the Ariac Manager (AM), see section 2.2.4.1 for further reference.

3.2.1 Robot Instance Methods

In order to be able to execute the actions of the different operations, movements for the individual robots and commands for the ARIAC environment need to be defined. This is done by creating a ROS2 interface node that contains all the robot functions defined as instance methods, such as change gripper or move to position as shown in listing 3.1,

Listing 3.1: Instance methods defined for floor robots.

```
    def change_gripper(self, gripper_type: int) -> bool:
        if not self._floor_tool_changer.wait_for_service(timeout_sec=1.0):
            self.get_logger().error("Tool changer service not available")
            return False

        req = ChangeGripper.Request()
        req.gripper_type = gripper_type
        future = self._floor_tool_changer.call_async(req)
        rclpy.spin_until_future_complete(self, future)
        return future.result().success

    def move_to_pose(self, pose: Pose, world_frame: str = "world") -> bool:
        goal = PoseStamped()
        goal.pose = pose
        goal.header.frame_id = world_frame

        self._floor_robot.set_goal(goal)
        plan_result = self._floor_robot.plan()
        if not plan_result.success:
            self.get_logger().error("Planning failed")
            return False

        return self._floor_robot.execute()
```

When an operation is being executed, the instance method required for the operation action is called.

3.2.2 Developing the Operation Runner

The development of the operation runner was carried out in several stages. As mentioned in section 2.3.1 an operation requires certain properties such as actions, guards, a state, and transitions to be defined. These definitions were in the form of Python classes and were used for creating the model containing the operations for the operation runner. Figure 3.1 illustrates the planned steps of development for the operation runner. As shown in the figure, the development began with emulating the desired behaviour of the operation runner, followed by the implementation of a version capable of interacting with the simulated environment.

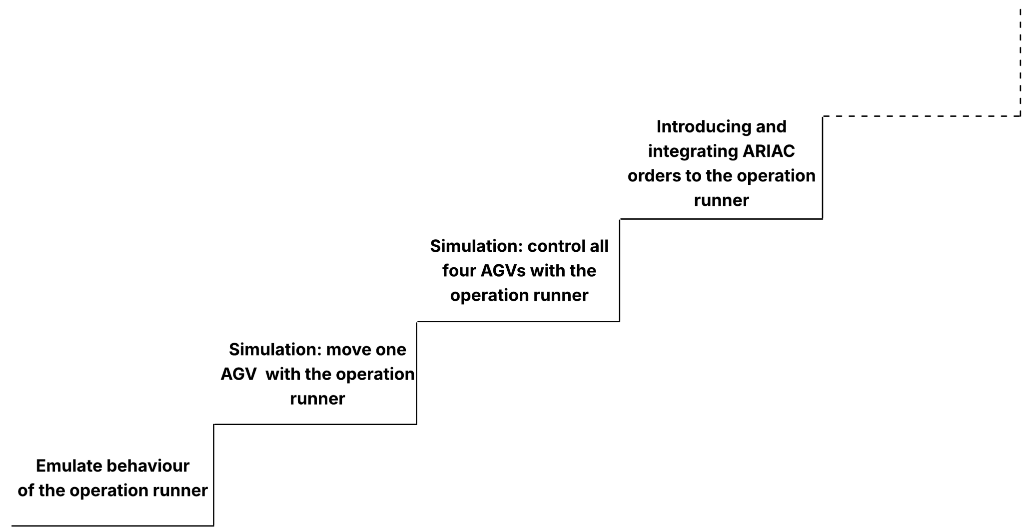


Figure 3.1: Image showing the first four development steps for creating the operation runner.

3.2.2.1 Emulation of Desired Behaviour

The first step of developing the operation runner was, as previously mentioned, to build a program that could emulate the desired behaviour of the operation runner. The emulator was built with the help of ROS2, and more specifically with the communication type of publisher-subscriber. The emulator consisted of two nodes; one control node that contained the operation runner, and one client node that responded to the control node's behaviour. The idea was to have a control node that would be running the operation runner and publish the state as a ROS2 topic, and then to have the client node listen to the state topic and publish another topic with state changes depending on the current state, see Figure 3.2 for a visualisation. The control node would then listen to the state-change topic and update the state accordingly, which in turn would mean operations would be able to start or complete.

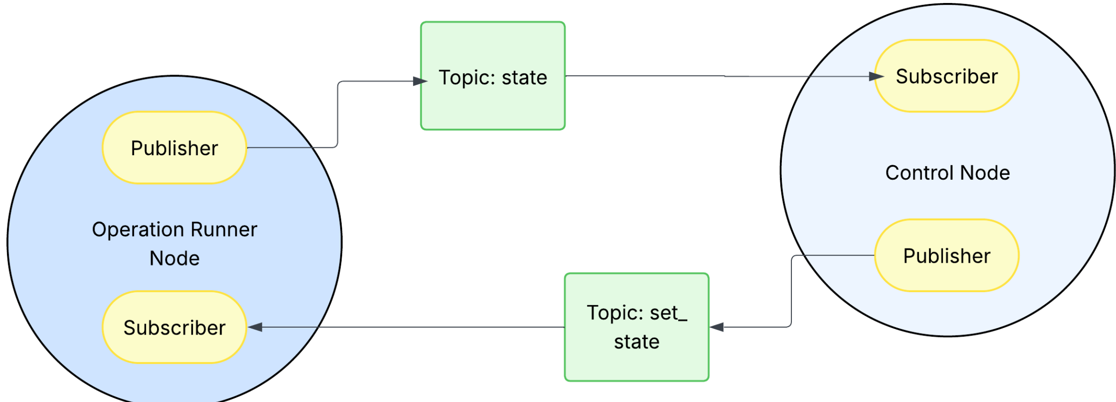


Figure 3.2: An image depicting two nodes and the communication topics between them, illustrating the architecture of the emulator.

As mentioned in section 3.2.2, the model, in which one creates one's operations, requires a few premade classes: actions, guards, state, transitions and operations. These were utilized to make a simple model class containing the state and the operations, which means it was the class in which we defined our state and the operations we wanted. The operations were defined as described in section 2.3.1. Since this program was only intended for emulating the behaviour of the operation runner, it was decided to only make two operations that the operation runner would be able to switch between, since that would be good enough to analyse whether or not the operation runner had the desired behaviour. The two operations used for the emulation can be seen in Listing 3.2.

Listing 3.2: Two operations created with the help of a for-loop in the model class.

```
1  for i, place in enumerate(['Warehouse', 'Pos1',]):
2       ops[f"moveAGV1To{place}"] = Operation(
3              name = f"moveAGV1To{place}",
4              precondition=Transition("pre",
5                  g(f"AGV1_allowedToMoveLocation && AGV1_pos!={i}"),
6                  a(f"!AGV1_allowedToMoveLocation,AGV1_moveto<-{i}")
7              ),
8              postcondition=Transition("post",
9                  g(f"AGV1_pos == {i}"),
10                 a(f"AGV1_allowedToMoveLocation, AGV1_moveto<-{-1}")
11             )
12         )
```

After defining the model, the operation runner could be developed. As detailed in the background in section 2.3.3, the operation runner must iterate through all the operations in the model, evaluate the guards of all the operations and start executing the ones whose guards evaluate to true. Consequently, the operation runner was written as a function, in the control script, that would iterate through all the operations in the model, evaluate their precondition guards and start every operation that it could. The operation runner also had to be able to finish operations, which means the same function would

also iterate through all operations and check the guards of the postcondition to see if there was any operations that could be completed. Lastly, a ticker was created, that then would call the operation runner function every 100 milliseconds, enabling the control node to monitor for changes and thus start and complete operations.

After writing the operation runner function for the control script, the publisher-subscriber communication was implemented. This would enable the controller node to communicate with other nodes. This meant that a publisher was added, as well as a client listener to monitor a topic that regulates the state. This would then enable another node to publish state changes that would affect what operations could be run.

When the control node was done, the client node was written so that it would make state changes, the operation runner in turn would cycle between starting and completing the two operations in the model.

3.2.2.2 Simulation of Desired Behaviour

Once we had a working emulator for the operation runner, the next was to integrate the operation runner into the simulated environment in Gazebo. To simplify this task, our main focus during this step was to only focus on one AGV in the simulation. The goal was to have the AGV move between two locations in the simulated environment, with the help of the operation runner.

When transitioning from working with the emulation to working with the simulation, we switched the ROS2 communication type. Instead of using the publisher-subscriber communication style, like in the emulator, a switch was made to use ROS2 actions instead. As briefly mention in section 2.2.2, actions are good for long running tasks, since they can be cancelled, provide feedback, and report back results when completed. This makes actions highly suitable for tasks such moving a robot in a factory, since the duration of a task can differ due to things like interruptions and blockage, which could affect the expected time of a task. It is critical to know when an operation has finished, which means having an action that reports back a result when a task is completed, is exactly what this type of control system requires.

To test the control system, it was decided to divide the control scripts into multiple scripts - each responsible for one part of the simulated environment, like explained in section 3.2.1. This meant that a control script that had the responsibility for the AGVs was created, as well as scripts for the other robots such as the floor robot or ceiling robot. The AGV control script contained functions that would send service request to another node that managed the simulated environment, to make the AGVs move within the simulation. For the communication with the operation runner node, an action service was implemented to handle requests from the operation runner node. The AGVcontroller script would then listen, and depending on the received action request, it would send a service request to the appropriate AGV. It would then wait until the action had finished, to then report back to the operation runner node on the status of the action, indicating whether it had failed or succeeded. Figure 3.3 depicts an image visualizing the architecture of the communication between the nodes. The figure is simplified to only show the major relevant aspects of the action communication surrounding the operation runner.

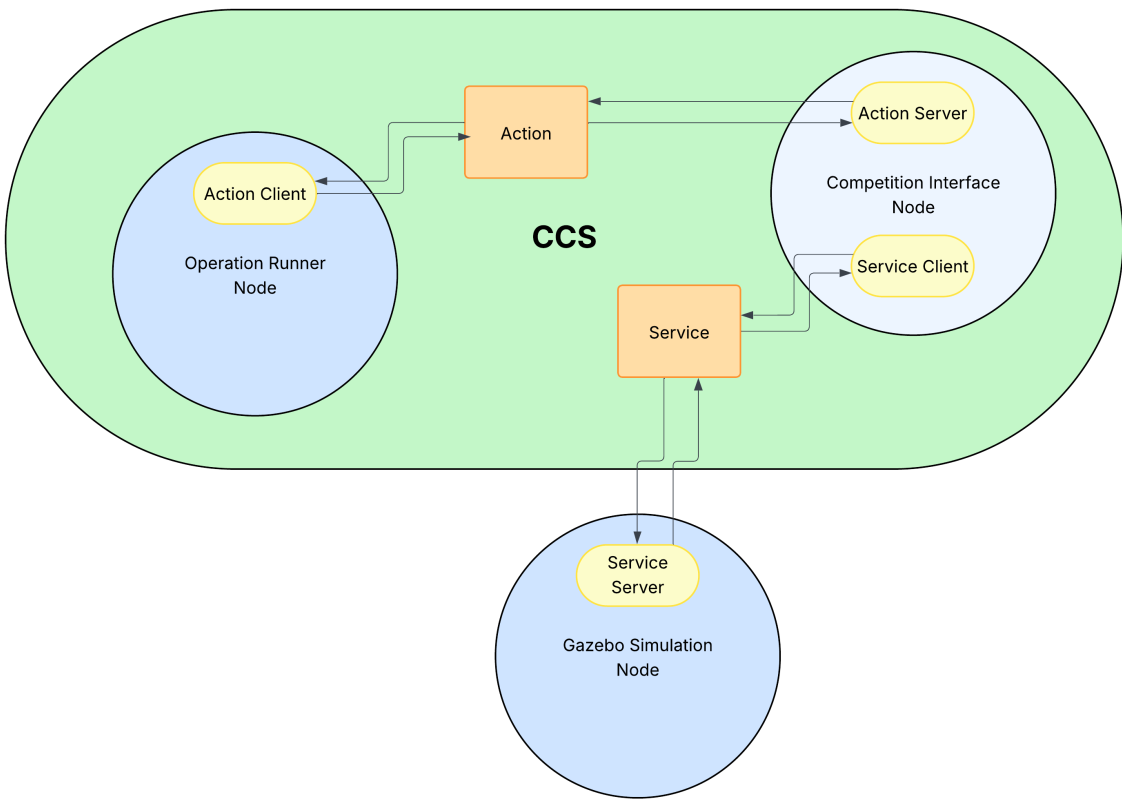


Figure 3.3: An image illustrating the architecture of the interactions of the CCS and the Gazebo simulation.

To be able to implement the action service between the operation runner node and the AGV controller node mentioned in the paragraph above, we wrote our own simple ROS2 action, which can be seen in Listing 3.3. This enabled us to send the exact information that was required back and forth between the nodes. The other actions, that was used to interact with the simulation, was acquired from the ARIAC tutorials, which the competition permitted.

Listing 3.3: The ROS2 action definition that was written and used for the communication between the operation runner node and the AGV controller node.

```
int32 agv_nr
int32 pos
---
int32[] ok
---
int32[] progres_feedback
```

No major changes were made to the operation runner script, compared with the emulation, apart from the change from publisher-subscriber to action server-action client communication.

A decision that had to be made during the development was that the AGV control node would perform

a busy-wait after sending an action request, which means it waited for the action to complete before being able to proceed. Consequently, only one of the four AGVs would be able to execute tasks at a time. This led to an undesired sequential behaviour, but was necessary to further the development of the project, due to the project's time constraints. As previously mentioned, there was going to be one control node for each essential part of the environment, like for example one for the AGVs and one for the floor robot, which means that for example the floor robot will still be able to operate concurrently with the AGVs, since they are two separately threaded nodes, which means the busy-wait is not ideal, but could be more detrimental in a slightly different scenario, where one would only have one control node.

This implementation enabled an AGV to be controlled by the operation runner, as well as allowing us to easily add more operations to the model and thereby expanding what the operation runner could control in the environment.

3.2.2.3 A Further Development of the Operation Runner

After solving the movement of one AGV with the operation runner, further development was done to enable the same function for the remaining three AGVs, according to the plan in Figure 3.1. With this, the ROS2 action definition for the AGVs was also changed to facilitate the new operations. After fully integrating the operation runner with the four AGVs. The following step was to add the remaining AGV functions as operations, which are lock and unlock the AGV trays.

To handle the ARIAC orders, the operation runner subscribes to the topic from the AM which publishes the order. The order is then preprocessed and added to a list that stores the pending orders.

3.3 Vision - Enabling the Control System to See the Environment

Identifying the different parts the robots would work with in the environment was a vital aspect of the project. The necessary attributes the control system would need to identify the parts were colour, type, position, and rotation. These attributes were especially important to be able to do the different ARIAC challenges, see section A.2 for more information, and to achieve a low score in the competition, see appendix B.1-B.4.

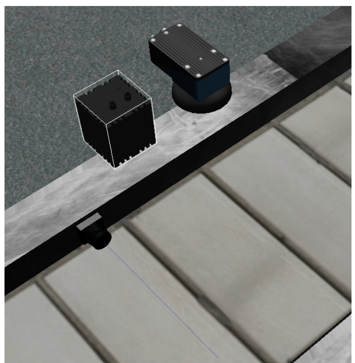
The project's methodology for the identification of parts was to utilize a CNN, which would only require camera images as input. Part of the reasoning behind the approach was that in the ARIAC environment, the developers have access to what is called an *Advanced logical camera* [28], which can report pose, colour, and type of a detected object. This meant that the collection of the required data to train a CNN would be easy to acquire, see section 3.3.1 for more details. For more information about the advanced logical camera and other sensors in the environment, please see section A.4 in appendix A.

The following sections provides a detailed overview of the methodology used for the CNN, along with all related components and processes to it.

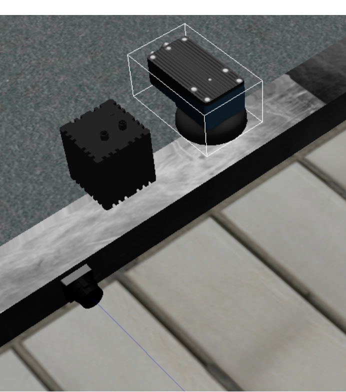
3.3.1 Data Collection for the CNN

As previously explained, the CNN had to be able to identify all the different parts the robots would work with in the simulation, which meant that the data that was to be collected was an RGB image of all the different variations of parts and translations they could have in the environment, as well as the information about what the RGB image showed.

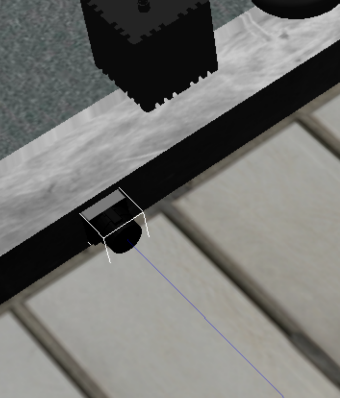
The data collected for the CNN was collected with sensors inside the Gazebo simulation. When collecting the data, the biases explained in section 2.4.3 were kept in mind and avoided to ensure a good model. A RGB camera was used to get the RGB images, while an advanced logical camera was used to get the information about the parts. Both cameras were placed above the conveyor belt, see Figure 3.4a and Figure 3.4b, making it easy for the cameras to take pictures of, and get the information about, the moving parts on the conveyor. A break beam sensor was then placed on the conveyor, see Figure 3.4c, and used to detect when a part was under the RGB and advanced logical cameras, so that we would only save an image of each part once.



(a) Image highlighting the RGB Camera with a small white box.



(b) Image highlighting the advanced logical camera with a small white box.



(c) An image showing the break beam sensor, highlighted with a white box, mounted on the conveyor belt.

Figure 3.4: The setup of the three sensors used for the data collection.

The data was saved as png files and named with the information they contained, since that way it would be easy to keep track of the data.

When the cameras were in place, the work regarding the spawning parts on the conveyor commenced. Parts were randomly spawned, as a way to avoid biases. See Figure 3.5 for an example.

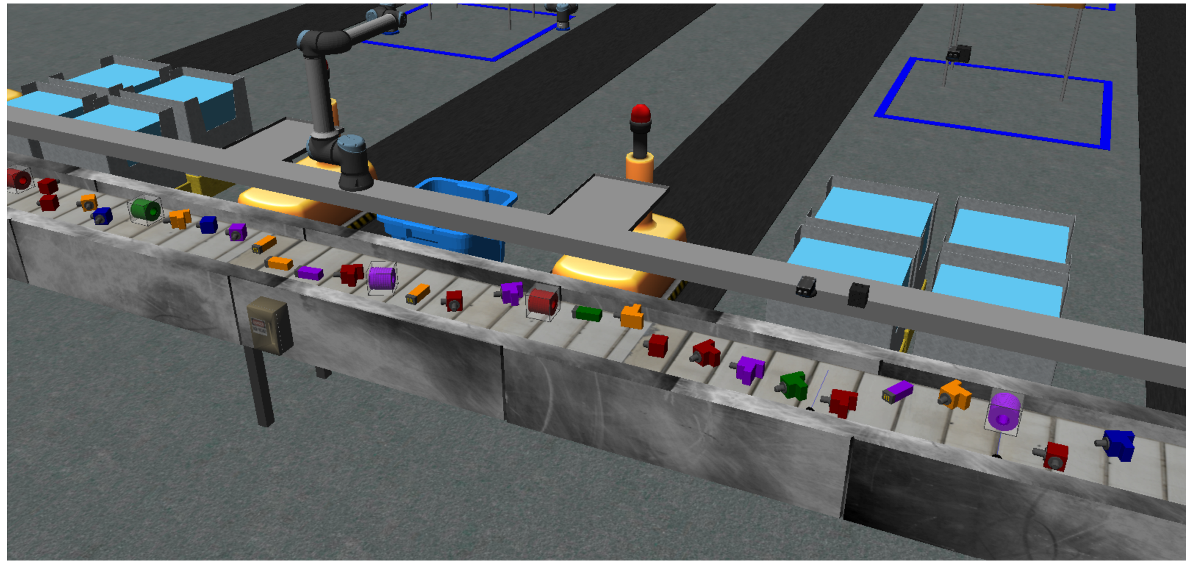


Figure 3.5: Image showing part of the conveyor belt in the Gazebo simulation. On the conveyor, there are different types of parts with different colours, rotations, and offsets. Some parts of the simulated factory can be seen in the background.

Items appear on the conveyor belt in the ARIAC tutorial based on information stored in the tutorial's configuration file. Consequently, modifying this file provides an easy way to control exactly which parts are used. Originally, the tutorial spawned a few selected different parts in a repeating pattern, but we wanted new parts with new parameters every time a part spawned. As previously mentioned in the theory section 2.4.2, the development of the CNN required a substantial amount of data, and manually deciding and writing exactly which parts should spawn, along with their parameters, would be highly inefficient. Therefore, an automated approach was deemed necessary. To address this, a script was developed to generate a list containing a specified number of randomly selected parts which would subsequently be spawned on the conveyor belt at the start and during the competition.

With this, the preparations to collect the data was completed - it was time to spawn the the different parts and begin the data collection. However, it was later decided that using data from the bins would be a better way to collect data for the CNN, as it had a more neutral background. In addition to that, we also had access to a pre-existing dataset portraying the different parts and their different variations. It became evident spawning and capturing new images of all possible variations of parts would be highly time consuming, and consequently, an additional approach was added - applying image transformations to the pre-existing dataset collected with the sensors, thereby augmenting the data and improving efficiency.

3.3.2 The Usage of OpenCv for Image Transformations for Expanding our Dataset

As previously mentioned, to be more efficient, it was decided to add different types of noise to the images through the help of image transformation. By adding noise to some images we can increase variability and help the CNN to still be able to identify parts even if a camera was partially faulty, as for example if the lens would be dirty or out of focus.

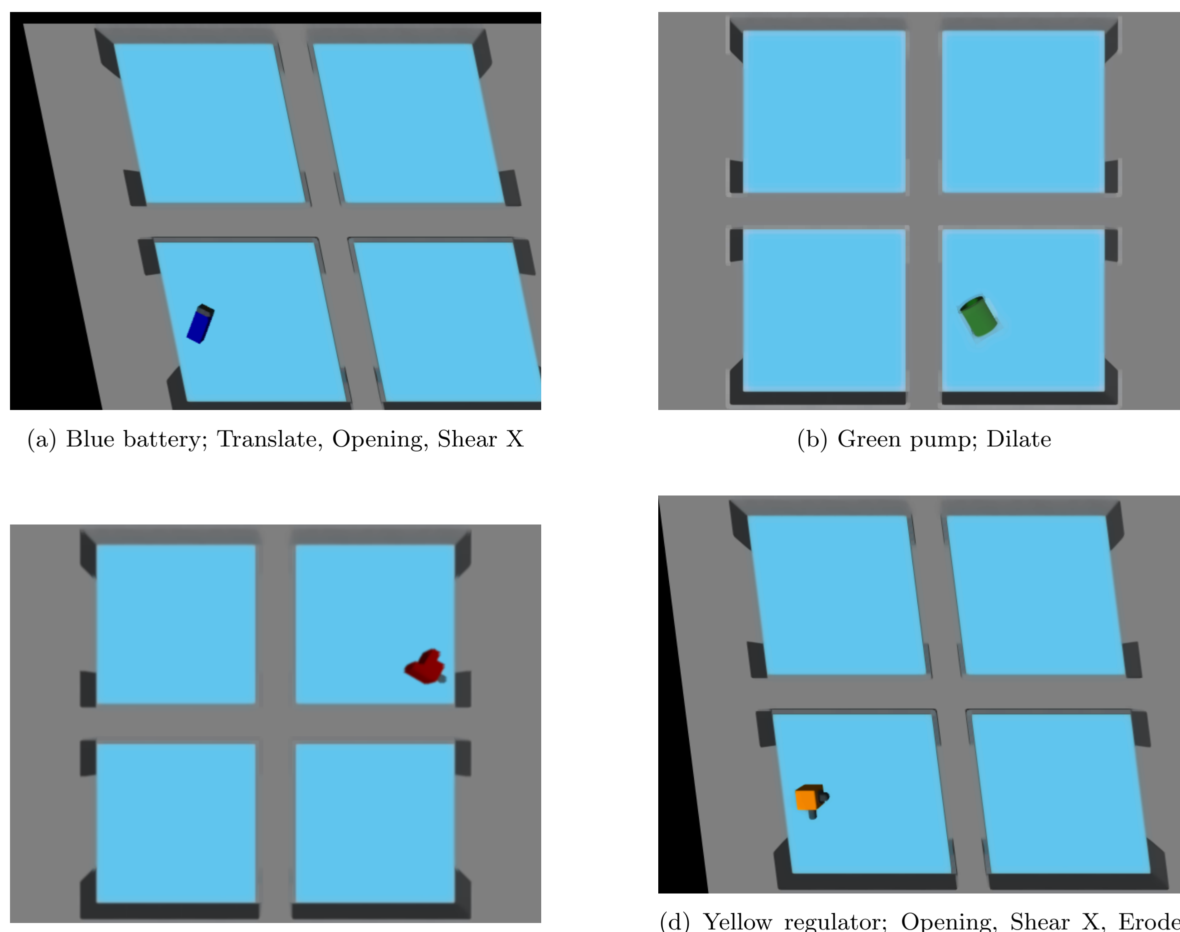
The saved simulation images were then used as a base for the image augmentation by applying one, or multiple, image transformations and saving the results as copies. This approach enabled significant expansion of our dataset without the need of manually capturing each variation. The transformations we decided to use was the following:

- Resizing
- Rotation
- Shearing along X- and Y-axis
- Translations along X- and Y-axis
- Gaussian blur
- Median blur
- Dilation
- Erosion
- Opening (erosion followed by dilation)
- Closing (dilation followed by erosion)

For further context on the purpose of the transformations, see section 2.4.6.

A quick and effective way to do this was through using the free OpenCV library in Python, which offers a wide range of optimized algorithms for computer vision and image processing. Its simplicity and efficiency made it well-suited for performing multiple image transformations programmatically, which meant that utilizing OpenCV allowed us to easily create a program that could take each image of the parts, transform them, and save them as a new images.

The script was configured to apply a random number of transformations, between one and four per image, in order to introduce variation while preserving realism within the simulated environment. The transformed images were saved in a separate directory, with abbreviated labels representing the applied transformations appended to the filenames. Figure 3.6 showcases several examples of the transformations that were applied, illustrating how they distorted the images.



(a) Blue battery; Translate, Opening, Shear X

(b) Green pump; Dilate

(c) Red sensor; Gaussian blur, Dilate, Closing

(d) Yellow regulator; Opening, Shear X, Erode, Dilate

Figure 3.6: Four images displaying different combinations of possible transformations that was applied to the images. Each of the images have one part laying in one of four possible bins.

3.3.3 Data Processing for the CNN

Each raw input image, in RGB PNG format, contains four regions corresponding to simulated bins. Each bin is processed individually. Although the image data is in RGB format, the HSV colour space is used internally to perform colour-based masking. Specifically, the image is converted to HSV (in memory) using OpenCV functions, and then a threshold is applied using predefined HSV ranges to generate a binary mask. These colour ranges were adapted from the ARIAC tutorials and detect the five part colours: red, green, blue, orange, and purple.

Contours are extracted from the HSV mask using `cv2.findContours()`, and the largest contour is selected to exclude background noise or false positives. The center of this contour is calculated using `cv2.moments()`, and the part is cropped using the self made `crop_centered_image()` function. If the crop region extends outside the image bounds, it is padded with black pixels using `cv2.copyMakeBorder()` to ensure a standardized 64×64 output.

After cropping, a second HSV masking step is applied to the cropped image. This additional masking ensures that any remaining background artifacts from the original image are removed, resulting in a clean binary input for classification.

Each part is also assigned to a specific slot within the bin. The part center coordinates `(cx, cy)` are mapped to a 3×3 grid using fixed thresholds for rows and columns. These coordinates are matched to a predefined slot mapping table. Since each part is detected based on contours and assigned independently, overlapping contours are naturally resolved - only one part, the dominant contour, will be assigned a slot.

Colour identification is performed by sampling the HSV pixel values at the center of the largest contour and comparing them to the predefined ranges. This simple method is robust under the controlled lighting conditions used in the setup.

The entire dataset was processed using this pipeline to ensure input consistency. All cropped images were standardized to the same size, centered, and masked before classification.

An example of a masked, cropped training image is shown in Figure 3.7, which visualizes a preprocessed part in standard format.

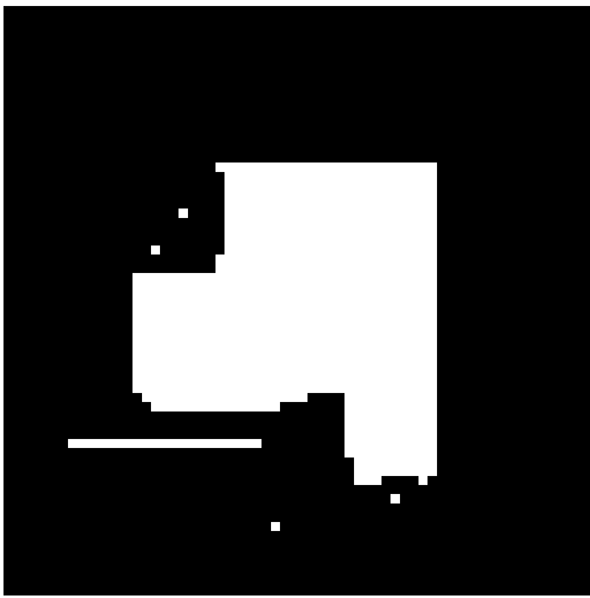


Figure 3.7: An image visualizing an HSV-masked and centered part (purple sensor) in its final cropped format (64×64 pixels).

3.3.4 The Usage of a Convolutional Neural Network

To classify the parts, a CNN was used due to its strong performance in image classification tasks and the availability of well-supported Python libraries such as PyTorch. The goal was to implement a fast, lightweight, and reliable classifier tailored to the constraints of the project.

The classifier architecture used two convolutional layers with Rectified linear unit activation function (ReLU) activations and max pooling. Then a flatten layer followed by two fully connected layers. Dropout regularization was applied to the fully connected layer to reduce overfitting by forcing the network to rely on distributed feature representations. The model was trained for 10 epochs using the Adam optimizer, chosen for its adaptive learning rate and robust performance in prior work. Although using a dynamic learning rate with momentum might have offered improvements, it was deemed unnecessary given the project's scale and time constraints. Lastly, a Softmax activation function was used in the final layer of the network to convert the output scores into class probabilities. The part is classified based on the class with the highest predicted probability. A similar architecture is visualized and can be seen in Figure 3.8.

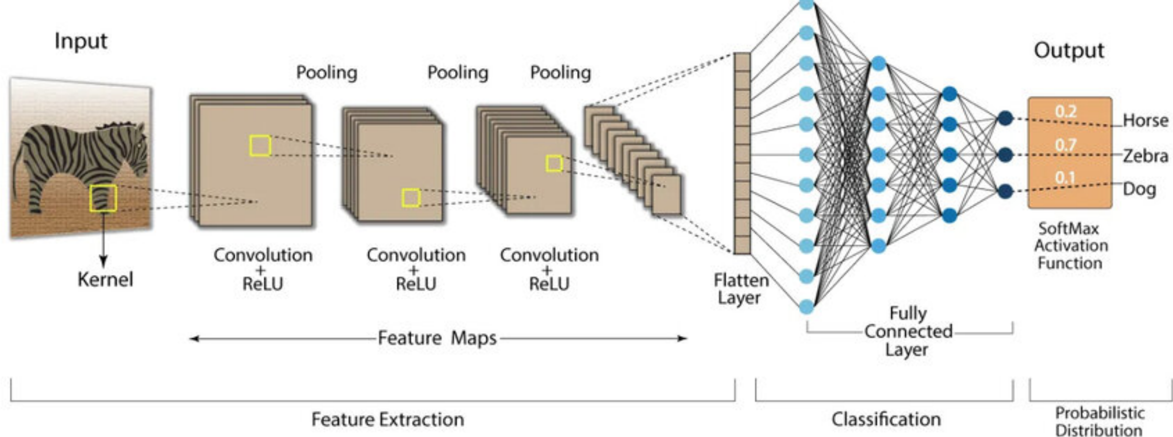


Figure 3.8: An image visualizing a general convolution neural network quite similar to ours [29] . Used with permission.

To prevent the model from learning biases based on data order, the training dataset was randomly shuffled in each epoch. This ensured that different batches were presented in a different sequence every time, improving generalization and stability.

The CNN predicts only part type, not colour or slot location, with colour classification and slot/bin assignment handled separately in the image preprocessor.

4. Results

As stated in the section 1.1, the objective of the project was to develop an operation runner and a CNN that together would function as a control system for managing a simulated factory environment in Gazebo. Throughout the project, efforts were consistently directed toward achieving this goal, but unfortunately the objective was not fully realized. The following sections presents the outcomes of the work conducted; first, the results related to the operation runner will be presented, followed by the result regarding the CNN.

4.1 The Competitor Control system

The project's Competitor Control System (CCS) consists of two main parts; a competitor package and an operation runner. As outlined in sections 3.1 and 3.2, the competitor package is what manages the competition state and orders. It also contains the launch file which starts the necessary nodes that move the robots, however, the launch file does not include the operation runner node. According to the specifications described in section 3.1, the competition setup requires two-terminals - one for launching the CCS and one for initializing the Gazebo environment. Although this setup was intended to be followed, it was later deemed more practical to launch competitor package and the operation runner separately during development. As a result, the integration of the operation runner node into the launch file was postponed and ultimately not completed within the project timeframe, meaning ultimately three terminals is utilized to launch all necessary nodes. A functioning operation runner was developed during the project and is detailed in the following section.

4.1.1 The Operation Runner

The work done on this project has resulted in an operation runner that can control certain parts of the simulated environment. As mentioned in the method, section 3.2, and visualized in Figure 3.1 there, the process for developing the operation runner was divided into clear and simplified steps, and has ultimately resulted in an operation runner that can control the AGVs in the environment. Specifically, the operation runner has been implemented with a structure that manages operations as state-based tasks. Each operation, such as moving AGVs or locking trays, is tracked using a state system where the status of operations (initial, executing) is explicitly managed. A dynamic state update mechanism has been integrated, allowing the operation runner to respond to external signals such as AGV position

feedback and order messages. These are handled through topic subscriptions, ensuring that the current state reflect the actual environment in gazebo.

What this entails, is that we have a functioning operation runner that is compatible with the simulated environment in Gazebo, that has the capability to control and interact with the objects inside said simulation.

We have a scalable model containing the operations required for controlling the environment, which easily can be expanded with more operations. Consequently, further development is easily done thanks to the architecture of our control system.

As previously described in section 3.2.2.1, the operation runner node includes a ticker mechanism that iterates through all the operations defined in the model. This ticker is triggered every 100 milliseconds, to ensure that operations are initiated and completed as promptly as possible, thereby enhancing the responsiveness of the control system. The chosen number is arbitrary, and further testing would be necessary do determine a more optimal timing for improved performance and reaction time.

4.1.2 State Management

A state management system was developed to track relevant system variables, including:

- AGV positions
- Tray IDs
- Destination stations
- Operation statuses
- Order-related variables

Examples of state variables used for the AGVs can be seen in Listing 4.1, and state variables for a kitting order can be seen Listing 4.2.

Listing 4.1: Example of initial state variables for AGV1, which is identical to other AGVs.

```
1   # AGV1 state
2   tray_id_on_agv1 = "",
3   agv1_location = 0,
4   agv1_lock_status = "unlocked",
5   agv1_kit_ready = False,
6   agv1_target = None,
7   agv1_task_in_progress = False,
```

The Competitor Control System (CCS) is subscribed to the **/ariac/orders** topic and stores the incoming orders in a list. When a current order no longer is in process, the queued orders are popped and used to update the state with a new order.

Listing 4.2: Variables representing the kitting order variables in the current state environment.

```
1      #Kitting Order State
2      kitting_id = "",
3      kitting_agv_id  = None,
4      kitting_destination = None,
5      kitting_tray_id = "",
6      kitting_parts = [],
```

Each operation has a status: initialized (i) or executing (e). The operations status is automatically added to the initial state when the program is initialized. If the precondition action to an operation evaluates to true, the operation status changes from "i" to "e". This allows the operation runner to not repeat the same operation when it already is executing. Only after the postcondition guard has evaluated to true will the status change back to "i", allowing the operation to be rerun.

4.1.3 Message and Action Handling

The developed control system subscribes to the competition orders via the **/ariac/orders** topic, parsing kitting task information such as agv ID, tray ID, destination, and required parts. Relevant order data are then stored in state variables for operation selection. AGV status is monitored through ROS2 topics for each AGV allowing real time updates of AGV positions. These updates are used to evaluate operation conditions and trigger state transitions.

In the method, section 3.2.2, specifically in Listing 3.3 the ROS2 action was originally intended for a simple move functionality. However, it has since been extended and generalized into a unified action that handles all necessary AGV functions. The updated action can be seen in Listing 4.3. This general action now covers movements, tray locking, and unlocking operations for any AGV, making it a versatile and reusable component within the system.

Listing 4.3: The updated ROS2 action definition for the AGVs.

```
1  #Goal
2  string command          # move_agv, lock_agv_tray, unlock_agv_tray
3  int32 agv_id            # which agv (1-4)
4  int32 destination           # where to move the agv (only for command move)
5
6  ---
7  # Result
8  bool   success
9  int32  location # final position where the agv ended up
10 int32  agv_id
11
12 ---
13 # Feedback
14 string progress_feedback
```

4.1.4 Order Management and Sequencing

An active order tracking mechanism was implemented to ensure only one order is processed at a time. The state variable `kitting_order_in_progress` indicates whether an order is currently being executed.

New incoming orders are either accepted (if idle) or queued in `kitting_pending_orders` for sequential processing. When an active order is completed, the next queued order becomes active until the `kitting_pending_orders` list is empty.

This architecture also allows for easy further enhancements, such as implementing order prioritization - one of the challenges presented in the ARIAC competition, refer to section A.2 for further information. However, due the time constraint, this functionality was not fully implemented within the scope of the project, although the foundations required for it was.

4.1.5 Summary of Control System Results

A scalable and adaptable foundation for a full control system has been developed during the project. The architecture has carefully been designed for easy further development for tackling the the other pick-and-place tasks, utilizing the other robots in the factory, and tackling the ARIAC challenges, described in section A.2. A functioning operation runner has been developed, as well as a functioning competitor package containing necessary nodes and a launch file for controlling the simulated environment.

4.2 Visual Input - Enabling Perception in the Control System

The following sections present the result related to enabling the control system to perceive its environment. Specifically, they detail the result of and around the Convolutional Neural Network (CNN).

4.2.1 Vision System

The vision pipeline developed in this project is responsible for identifying parts using a cnn. The pipeline was divided into four major stages: data collection, data processing, model training, and part classification. Each stage contributed significantly to the overall performance and robustness of the system.

4.2.1.1 Dataset and Augmentation

Initially, the dataset consisted of 3,458 labelled images. To improve the model's ability to generalize to unseen data, data augmentation was applied through flipping and rotation transformations. This increased the dataset size to 6,916 images, effectively doubling its diversity. This augmentation not

only simulates real-world variability in part presentation but also reduces the risk of overfitting during training.

4.2.1.2 Data Processing Pipeline

Each image was processed through the same standardized pipeline. This includes:

- HSV masking to isolate part regions by colour,
- Contour detection to locate and centre the part,
- Cropping and padding to produce a uniform 64×64 image,
- Secondary masking for background cleanup.

This preprocessing ensures that the CNN receives consistently formatted inputs regardless of part position or bin location.

4.2.1.3 CNN Training and Accuracy

The CNN classifier was trained for 10 epochs using the Adam optimizer with a fixed learning rate. The final model consistently achieved an accuracy of approximately **98.7%** on the original data set with a 80/20 train/test data split. This high accuracy indicates the model's strong capability in distinguishing between different part types under a variety of conditions, including flipped and rotated inputs.

4.2.1.4 System Behaviour and Real-Time Integration

The current implementation processes one full image frame containing four bins. For each bin, it classifies all parts present, determines their colour, and returns a structured dictionary specifying each part's type, bin, slot location, and estimated colour.

Although integration into ROS 2 and Gazebo simulation is still pending, the full vision pipeline works independently and is ready for deployment. The average processing time per image (including all four bins) is approximately **8 seconds**. This performance is sufficient for near real-time integration, especially in scenarios with slower robot decision loops or discrete planning steps.

4.2.1.5 Scalability and Modularity

The pipeline has been designed to be modular, making it easily extendable. For example, additional CNN outputs could be introduced to classify part orientation, detect defects, or estimate part confidence thresholds. The clear separation between preprocessing and classification also allows for future upgrades to either stage independently.

4.2.2 Summary of Vision System Results

The vision system reliably identifies part types in a wide variety of input conditions, maintains high classification accuracy (98.7%), and operates fast enough for integration into the broader robotic control system. Its modular architecture allows future development and integration into ROS 2-based automation pipelines.

5. Discussion

The following sections discuss the results of the project, beginning with the methodology and outcomes related to the control system, followed by an analysis of the vision component. While both subsystems, the operation runner and the CNN-based part classification were developed and tested independently, their integration into a unified control framework within the ARIAC competition environment remained incomplete. This gap stemmed from time constraints and unresolved technical challenges, ultimately limiting the system's ability to function cohesively in the simulated factory scenario.

5.1 Analysis of the Control System

This section evaluates the control system developed in the project, focusing on the use of an operation runner in place of a traditional planner. The section covers the implications of this design choice, including benefits and limitations in terms of modularity and scalability. Specifically it explores how the system performs as more agents are introduced to the environment, and outlines future improvements and unrealized ideas that could enhance overall system performance.

5.1.1 No Planner vs Planner

In the current control system, a no-planner approach is used. In this system operations are defined as self contained actions consisting of precondition and a postcondition, which in turn are logical conditions that determine if the operation can begin or end depending on the current state as mentioned in 3.2. Thus the benefits of no-planner control system is relatively simple and modular implementation. To alter the sequence of actions, only the guards need to be modified, without requiring fundamental changes to the operation runner itself. The clear separation of logic into operations enhances maintainability and enables straightforward debugging and testing, as errors can be isolated to individual operations or guards.

The downside of such an operation runner is that its limited to small scale goal directed behaviour, operations are executed depending on the current state without regards to broader overarching plan. Another downside would be the inefficiency with handling complexity for non-nominal behaviour, where flexibility and reactivity beyond local operations are required. As the system scales up in complexity in

regards to non-nominal behaviour, maintaining purely reactive control becomes increasingly difficult.

A future step of improvement, would have been integrating a sequence planner with the control system instead of using the simple operation runner currently in use. A sequence planner would, depending on a goal, have been able to pick out the relevant operations in a working order to complete the goal. This would have enabled overarching goal oriented behaviour, where operations are selected not just based on the immediate state, but with an overall plan in mind. Furthermore the integration of a sequence planner would have aligned more closely with the ARIAC competition's emphasis on industrial robot agility as mentioned in 2.1. For the integration of a sequence planner further inspiration from for example Dahl, Erős, Bengtsson, *et al.* [30] and Dahl, Bengtsson, and Falkman [31] would have been taken.

5.1.2 Scalability

The use of an operation runner as the central mechanism of the control system provides a highly scalable foundation when increasing the amount of robots that function nominally. As outlined in section 2.3.1 operations are defined as discrete, self-contained units with clearly specified preconditions and postconditions. This allows new operations to be added or removed independently, without requiring changes to the underlying control structure, providing immense benefits to systems that evolve over time.

This scalability is highlighted during simulation where we successfully extended the system from controlling one AGV to controlling four, see 3.2.2.3. This is accomplished by merely adding additional operations whose guards the runner goes through during its ticker cycle, see 3.2.2.1. This modification requires no restructuring of the control logic, thereby illustrating the system's capacity to grow linearly without introducing additional architectural complexity.

5.2 Analysis of the Vision Components

The use of different kinds of cameras is very prevalent in the competition, and it proved more difficult than initially thought. Since a lot of out time in the beginning went toward the installations and understanding of the systems, the work on vision was delayed. However, the CNN was still developed to identify the parts with good accuracy, despite not having as much training data as desired.

5.2.1 Integrating the CNN to the ARIAC workspace

The current code was made in mind to be compatible with ROS2 framework. The large usage of Pyhton packages means that integration can be more complicated than intended. However great length have been made to avoid cross calling or sections where ROS2 integration would be necessary. An improvement that could have been made was already devising a small patch of code that would integrate correctly with Python code, making it possible to later seamlessly integrate.

5.2.2 Optimization

The code is written in a simple manner, for clarity and functionality, therefore little optimization work has been done. This is simply because a functional system that has a good framework for optimization is easier to optimize later on, which in this case has not been done. More time could have also been put into researching more appropriate functions or methods to solve our problem. A lot of the choices made during the development were based on previous knowledge. This can result in method solving that is not optimized for our task, even if it still works as intended.

5.3 Future Possibilities - Explorations of Unfinished Ideas

Our results indicate that no finished control system was made. With additional time, the operation runner would have been further developed and been able to handle all the different agile tasks demanded by the environment. The agility challenges mentioned in the background, in section A.2, have barely been considered in the operation runner detailed in the result section. Although, what was considered during development was scalability and adaptability, meaning it was designed so that it would be easy add complexity and expand the scope of the control system's capabilities. Consequently, with additional time, solutions for the agility challenges would have been able to be implemented.

Other aspects for future development would be the classification for parts on conveyor, and their rotation and position. The conveyor is a central part of the simulation. However for this to be done, we would need a program that assigns each part a coordinate location in the simulation in real time. For this, a vector solver would need to be created, and a lower computation time for the classification.

As previously outlined in the method, code for collecting data from the conveyor belt has been developed and is ready for deployment. However, due to time constraints it was not utilized, since the time constraint limited the extent of the work related to collecting the data. Nevertheless, the work conducted enables efficient and straightforward acquisition of images of parts on the conveyor belt, providing a solid foundation for future development.

Another improvement would be the classification of parts rotation and flip. The issue with this is that flip is not necessarily possible with the current camera set up and other solutions might have to be devised. For rotation, this might even be possible in the current training model with some slight modification. However considering the added complexity to the CNN, this was not done previously. A more effective way could possibly be found using the other sensors provided in the competition.

While the applied transformations successfully increased dataset diversity, they may not fully capture the types of variation present in real production environments. The augmented dataset was also not used to retrain the CNN, as the initial training had already been completed. However, there are many possibilities for further development, including applying additional transformations to introduce even more variation. A future evaluation of which specific transformations best reflect real-world conditions would be valuable, as not all types of distortion are representative of realistic scenarios.

Additionally, all data used in this project was collected within a simulated environment, in accordance with the competition framework. For use in a real industrial setting, training the model on real-world

camera images would be preferable in order to ensure robustness and better domain alignment.

While the control system coordinates robot actions, the vision system plays an equally vital role in enabling perception and autonomy. Together, these components form the foundation of a complete intelligent robotic system. The modular design of the vision pipeline also ensures that improvements such as retraining with real images or more advanced classification tasks can be integrated with minimal disruption to the existing framework.

Lastly a focus we unfortunately did not have time for when working on this project was the scoring for the competition. As mentioned, this project originally intended to develop a control system compatible with the ARIAC 2024 competition, meaning one would have to consider the scoring system to be able to place well in the competition. See appendix B.1 for details about the scoring system. When the project began, it was something we thought would be central when developing the control system, but we soon realized that we just simply needed a control system first before we even should consider designing it so it could have a low competition score. A good continuation, would therefore be to consider the scoring system and to further develop the control system and optimize it accordingly, for example the usage of lower-cost sensors, since a low score in the competition would translate to a well-functioning agile system.

6. Conclusion

This thesis developed a ROS2-based infrastructure control system for autonomous robots in a simulated factory environment, aligning with the Agile Robotics for Industrial Automation Competition (ARIAC) framework. The operation runner demonstrated effective control of the Automated Guided Vehicle (AGV)'s by dynamically evaluating preconditions and postconditions of the operations, enabling scalable control of multiple robots. The vision system developed during this project utilized a Convolutional Neural Network (CNN) trained on Hue, Saturation, Value (HSV) and centered images, achieving high classification accuracy. Although the augmented dataset created through transformation has not yet been used for retraining, it significantly expanded the dataset and supports future improvements in model robustness. The classification pipeline, including preprocessing and modular structure, is designed for scalability and future integration with the control system. The modular design also supports potential real time integration and improvements such as the use of simpler sensors to reduce system cost. Together, these results contribute to the broader goal of creating adaptable and efficient control systems for autonomous manufacturing environments. In addition, the division of responsibilities between the control and vision systems reflects a scalable and realistic architecture for autonomous factories. The project demonstrates not only technical feasibility but also the flexibility needed for real world implementation.

Future work should prioritize the full integration of the control system and vision components into the ARIAC competition framework to allow the system components to work in a competitive scenario. Also the various individual parts of the system can be explored further, this includes among other things optimizing the vision system for lower inference times. Additionally, replacing the current operation runner with a goal oriented sequence planner would enable more efficient and intelligent decision making, aligning the system more closely with the dynamic requirements of industrial environments. Expanding support for concurrent operations and addressing bottlenecks in the Automated Guided Vehicle (AGV) controller could also enhance the overall scalability and robustness of the system.

Sources

[1] M. Dahl, C. Larsen, E. Eros, K. Bengtsson, M. Fabian, and P. Falkman, “Interactive formal specification for efficient preparation of intelligent automation systems”, *CIRP Journal of Manufacturing Science and Technology*, vol. 38, pp. 129–138, 2022, ISSN: 1755-5817. DOI: https://doi.org/10.1016/j.cirpj.2022.04.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1755581722000761.

[2] J. Arents and M. Greitans, “Smart industrial robot control trends, challenges and opportunities within manufacturing”, *Applied Sciences*, vol. 12, no. 2, p. 937, 2022. DOI: 10.3390/app12020937.

[3] C. McLean and S. Leong, “The role of simulation in strategic manufacturing”, Gaithersburg, MD: National Institute of Standards and Technology, 2001. [Online]. Available: https://www.nist.gov/msidlibrary/doc/ifip5.pdf.

[4] F. M. Tseeva, M. M. Shogenova, K. M. Senov, K. V. Liana, and A. M. Bozieva, “Comparative analysis of two simulation environments for robots, gazebo, and coppeliasim in the context of their use for teaching students a course in robotic systems modeling”, in *2024 International Conference ”Quality Management, Transport and Information Security, Information Technologies” (QMTISIT)*, 2024, pp. 186–189. DOI: 10.1109/QMTISIT63393.2024.10762908.

[5] C. Müller, *World Robotics 2024 – Industrial Robots*. Frankfurt am Main, Germany: VDMA Services GmbH, IFR Statistical Department, 2024, ISBN: 978-3-8163-0765-5.

[6] A. Lidell, S. Ericson, and A. Ng, “The current and future challenges for virtual commissioning and digital twins of production lines”, Apr. 2022. DOI: 10.3233/ATDE220169.

[7] National Institute of Standards and Technology. “ARIAC Documentation”. [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/. (accessed on: 2025-01-29).

[8] National Institute of Standards and Technology. “What is robot agility?” [Online]. Available: https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation-competition/what-robot. (accessed on: 2025-01-29).

[9] Open robotics. “Why ros?” [Online]. Available: https://www.ros.org/blog/why-ros/. (Accessed: 2025-02-09).

[10] Open robotics. “The ros ecosystem”. [Online]. Available: https://www.ros.org/blog/ecosystem/. (Accessed: 2025-02-09).

[11] Open robotics. “Understanding topics”. [Online]. Available: https://docs.ros.org/en/iron/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html. (Accessed: 2025-04-28).

[12] Open robotics. "Understanding services". [Online]. Available: https://docs.ros.org/en/iron/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html. (Accessed: 2025-04-28).

[13] Open robotics. "Understanding actions". [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html. (Accessed: 2025-04-16).

[14] Open Robotics. "Open platforms for robotics". [Online]. Available: https://www.openrobotics.org/. (Accessed: 2025-02-09).

[15] National Institute of Standards and Technology. "Terminology". [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/competition/terminology.html. (accessed on: 2025-02-05).

[16] National Institute of Standards and Technology. "Orders". [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/competition/orders.html. (accessed on: 2025-02-06).

[17] E. Zhang, "The application of industrial automation computer control system", in *2022 International Conference on Computing, Robotics and System Sciences (ICRSS)*, 2022, pp. 20–23. DOI: 10.1109/ICRSS57469.2022.00015.

[18] Z. Kootbally, "Industrial robot capability models for agile manufacturing", *The Industrial robot*, vol. 43, pp. 481–494, 2016. DOI: 10.1108/IR-02-2016-0071.

[19] K. Bengtsson, P. Bergagard, C. Thorstensson, *et al.*, "Sequence planning using multiple and coordinated sequences of operations", *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 308–319, 2012. DOI: 10.1109/TASE.2011.2178068.

[20] M. Dahl, E. Erős, K. Bengtsson, M. Fabian, and P. Falkman, "Sequence planner: A framework for control of intelligent automation systems", *Applied Sciences*, vol. 12, no. 11, 2022, ISSN: 2076-3417. DOI: 10.3390/app12115433. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5433.

[21] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022, Pre-publication version. Available for personal use only. Published by Cambridge University Press. Version dated July 8, 2022. [Online]. Available: https://smlbook.org/book/sml-book-draft-latest.pdf.

[22] H. Suresh and J. V. Guttag, "A framework for understanding unintended consequences of machine learning", *CoRR*, vol. abs/1901.10002, 2019. arXiv: 1901.10002. [Online]. Available: http://arxiv.org/abs/1901.10002.

[23] IBM. "AI vs. machine learning vs. deep learning vs. neural networks: What's the difference?" [Online]. Available: https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks. (accessed on: 2025-03-23).

[24] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning", *CoRR*, vol. abs/1908.09635, 2019. arXiv: 1908.09635. [Online]. Available: http://arxiv.org/abs/1908.09635.

[25] S. Jain, K. Hamidieh, K. Georgiev, A. Ilyas, M. Ghassemi, and A. Madry, *Data debiasing with datamodels (d3m): Improving subgroup robustness via data selection*, 2024. arXiv: 2406.16846 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2406.16846.

[26] M. Saqib, M. A. Mehmood, N. Tahir, and Y. Hafeez, "Towards deep learning based smart farming for intelligent weeds management in crops", *Frontiers in Plant Science*, vol. 14, Jul. 2023. DOI: 10.3389/fpls.2023.1211235.

[27] National Institute of Standards and Technology. "Competition overview". [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/competition/overview.html. (accessed on: 2025-02-12).

[28] National Institute of Standards and Technology. "Sensors". [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/competition/sensors.html. (accessed on: 2025-02-12).

[29] ResearchGate Contributors, *Performance of lip-sync instead of speech imagery, new combination signals, supplement bond graph classifier and deep formula detection as confidents extraction and roots detection classifier for eeg and bci - scientific figure*, https://www.researchgate.net/figure/General-model-of-Convolution-Neural-Network-52_fig2_372967405, Accessed: May 13, 2025, 2025.

[30] M. Dahl, E. Erős, K. Bengtsson, M. Fabian, and P. Falkman, "Sequence planner: A framework for control of intelligent automation systems", *Applied Sciences*, vol. 12, no. 11, 2022, ISSN: 2076-3417. DOI: 10.3390/app12115433. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5433.

[31] M. Dahl, K. Bengtsson, and P. Falkman, "Application of the sequence planner control framework to an intelligent automation system with a focus on error handling", *Machines*, vol. 9, no. 3, 2021, ISSN: 2075-1702. DOI: 10.3390/machines9030059. [Online]. Available: https://www.mdpi.com/2075-1702/9/3/59.

[32] National Institute of Standards and Technology. "Agility challanges". [Online]. Available: https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation-competition/about-ariac. (accessed on: 2025-01-30).

[33] S. W. Feng, T. Guo, K. E. Bekris, and J. Yu, "Team rubot's experiences and lessons from the ariac", *Computers in Industry*, vol. 129, p. 103 413, 2021. DOI: 10.1016/j.compind.2021.103413.

[34] National Institute of Standards and Technology. "Scoring". [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/evaluation/scoring.html. (accessed on: 2025-02-05).

[35] National Institute of Standards and Technology. "Environment". [Online]. Available: https://pages.nist.gov/ARIAC_docs/en/latest/competition/environment.html. (accessed on: 2025-02-05).

Appendix A - ARIAC and the ARIAC Environment

The following section outlines key aspects of the of the ARIAC environment that are essential for a successful execution of this project.

A.1 Interactions between the ARIAC Manager and Competitor Control System

The following image is a flowchart depicting the interactions and workings of the Ariac Manager (AM) and the Competitor Control System (CCS). For more information about the AM and CCS see section2.2.4.1.

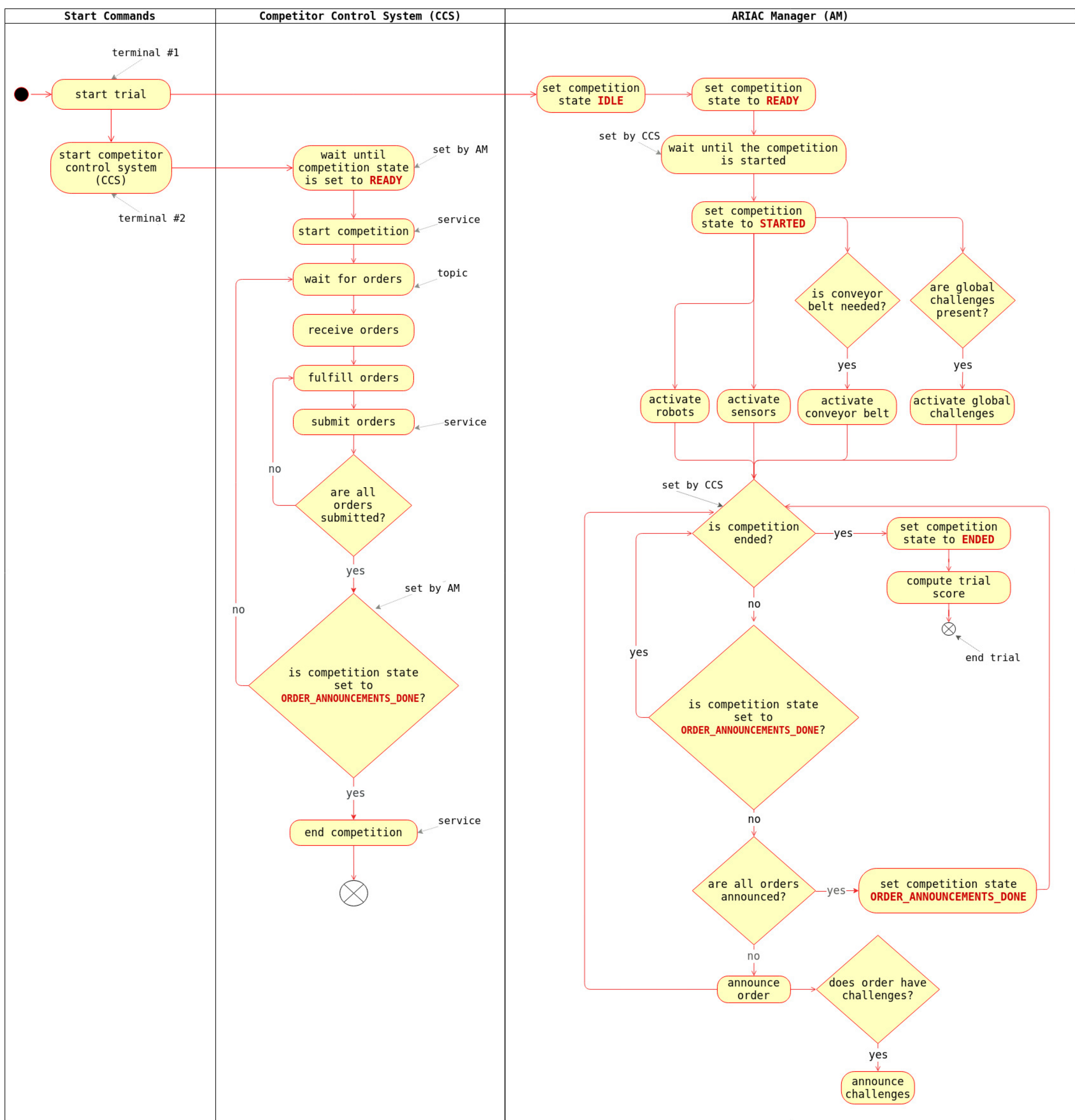


Figure A.1: A flowchart showing logic of the AM and CCS, as well as the interactions between the two. [27]. Used with Permission.

A.2 The Challenges the ARIAC Competition Aims to Tackle

As mentioned in the introduction, the core focus of ARIAC is robot agility, and to simulate some of the challenges associated with industrial automation, ARIAC presents competitors with various tasks designed to mimic real-world problems [32]. These challenges include:

Faulty Parts Challenge: Sensors should be used detected faulty parts and ensure their removal from shipment [32].

In Process Order Update Challenge: During the execution of a task, the order may be updated, requiring the change of a needed type of product [33].

Flipped Parts Challenge: Some parts may spawn flipped on the Z-axis in the environment and thus needs to be reoriented to their correct orientation [32].

Dropped Part Challenge: Sometimes a robotic arm can drop a part while holding it [32]. The system needs to recognize that a part has been dropped and therefore pick up an identical part with the same colour.

Robot Malfunction Challenge: When a robot malfunctions, it stops moving and cannot be controlled [32]. To counteract this, another robot must be used to complete the task until the malfunctioned robot starts working again. If all robots malfunction, the system must wait until the issue is resolved for at least one of the robots before continuing with the order.

Sensor Blackout Challenge: During the competition, some sensors may stop working for a given amount of time[32]. The system must recognize this and act based on the internal world model as it was before the blackout occurred.

High Priority Order Challenge: Orders can have different priorities [32]. If an order with a higher priority is submitted, the system must immediately prioritize and complete the order with higher priority before continuing with the regular ones.

Insufficient Part Challenge: The system must detect when there are not sufficient parts to complete an order[32]. When that happens the system must submit incomplete orders.

The challenges are then graded with a score, and the scoring system is built on three main aspects; cost factor, efficiency factor, and task score[34]. More detailed information about the scoring system can be found in appendix A.6

A.3 Parts

Parts are the components used for the different pick-and-place orders. There are a total of four different parts; battery, pump, sensor, and regulator - each having a possibility of spawning as one of five colours [35]. The different parts and the colours can be seen in Figure A.2.

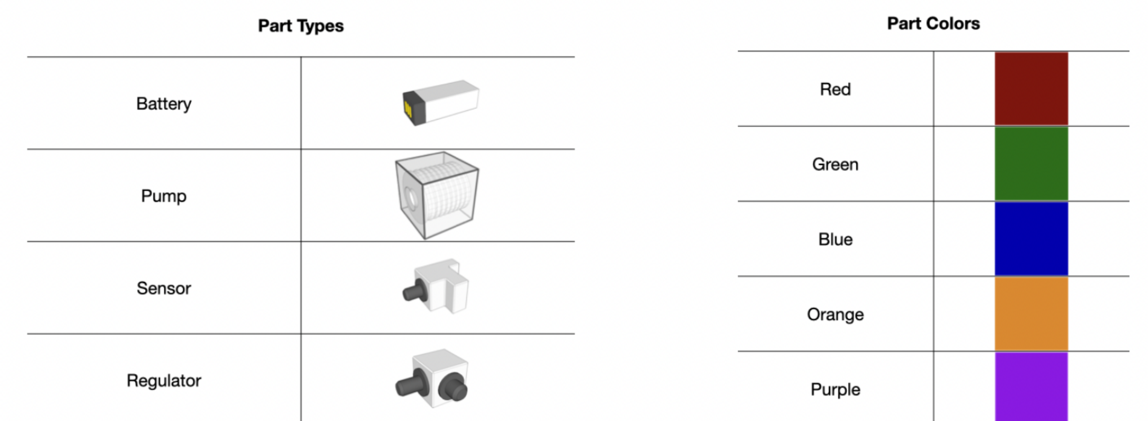


Figure A.2: An image showing the different parts and their type names, alongside with the possible colours the parts can have in the simulation.[35]. Used with permission.

There are three main locations within the simulation in which the parts can spawn:

Part bins: There are a total of eight bins in the work environment. Each bin consists of nine slots for parts, and the parts can spawn in multiple orientations in these slots. Here in Figure A.3 we can see a cluster of four bins standing next to each other in the simulated environment.

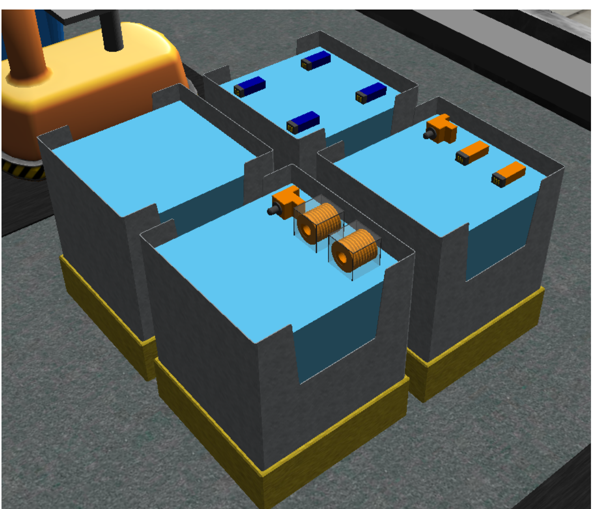


Figure A.3: Image showing four of the eight bins in the factory. Three of the bins in the image contain a few parts, while the fourth is empty.

Conveyor belts: Parts spawn on the conveyor belt at the start of the competition. Competitors need to make sure to pick them up while the belt is moving. Here in Figure A.4 and image of the conveyor belt is shown, and on it different parts have been spawned with different colours and rotations.

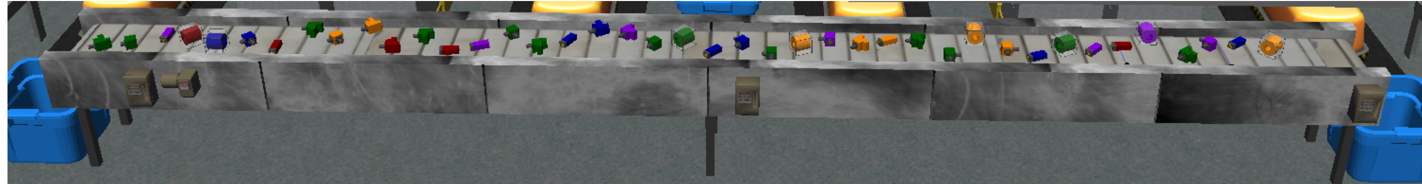


Figure A.4: An image of the conveyor belt transporting randomly spawned parts.

Automated Guided Vehicle (AGV)s: An AGV is a type of robot that moves along a pre-designed path. Figure A.5 shows what an AGV looks like in the simulated environment. Parts can spawn on an AGVs tray during assembly orders.

Figure A.5: An image showing one of the four AGVs in the simulated factory.

A.4 Sensors

In total, there are ten sensors in the simulation environment; eight "static sensors" and two "robot sensors." Static sensors can be placed anywhere the competitors choose in the simulated factory, while the robot sensors must be mounted on either the floor or ceiling robots [28]. Table A.1 shows lists of the available sensors and their prices.

The cost of each sensor is factored into the scoring system - the more money spent on sensors, the lower the overall score[34]. More details on scoring can be found in appendix B.1.

Table A.1: A listing of the sensors available in the environment, together with their cost[28].

Sensor Type	Cost
Breakbeam	$100
Proximity	$100
Laser profiler	$200
LIDAR	$300
RGB camera	$300
RGBD camera	$500
Basic logical camera	$500
Robot RGB camera	$800
Robot RGBD camera	$1000
Advanced logical camera	N/A

The best sensor in the environment is the advanced logical camera. This is because it will report the pose, type, and colour for a detected part - it is what we call a *cheat camera*. This camera is not allowed to be used in competition, but it is allowed to be used as a tool during the development.

A.5 Kit Trays and assembly inserts

A kit tray is a tray where parts are to be placed. There are five sections in total, four sections, where each section is in its quadrant, and a fifth section where the ID of the kit tray is marked with a tag. See Figure A.6 for visualisation purposes. The mark is of the ARUCO standard, which is a fiducial marker system and is used for unique identification, object detection, and pose estimation. The different marker IDs can be seen in Figure A.7.

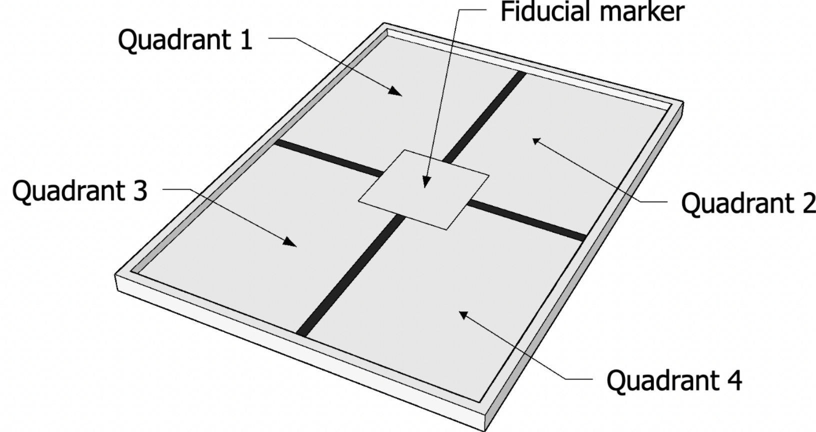


Figure A.6: An image showing a kit Tray, that is used during kitting tasks. Arrows are showing the four different quadrants parts can be placed in, as well as where the fiducial marker is located. [35]. Used with permission.

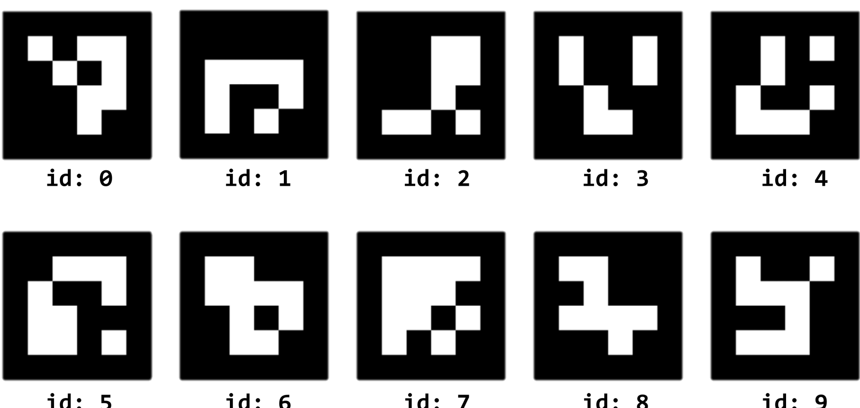


Figure A.7: The different marker IDs for the fiducial marker system used for the kit trays. [35]. Used with permission.

As mention when describing the assembly order in section 2.2.4.2, parts are assembled in an fixed insert. The different parts, explained in section A.3, are assembled into different positions in the insert, which can be seen in Figure A.8.

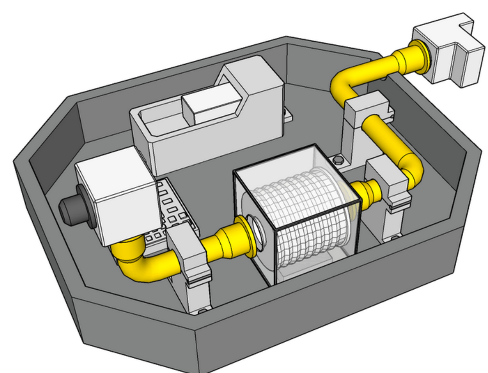


Figure A.8: An image of the insert used for assembly orders. [35]. Used with permission.

A.6 Robots

Robots play a crucial role in automation. Their tasks include material handling and assembly. The ends of each robot arm are equipped with a vacuum gripper which is their way of picking up parts. They cooperate with sensors to determine the distance and colour of parts. The robotic arms that will be used in this project are of the UR10e arm model. See Figure A.9 for reference.

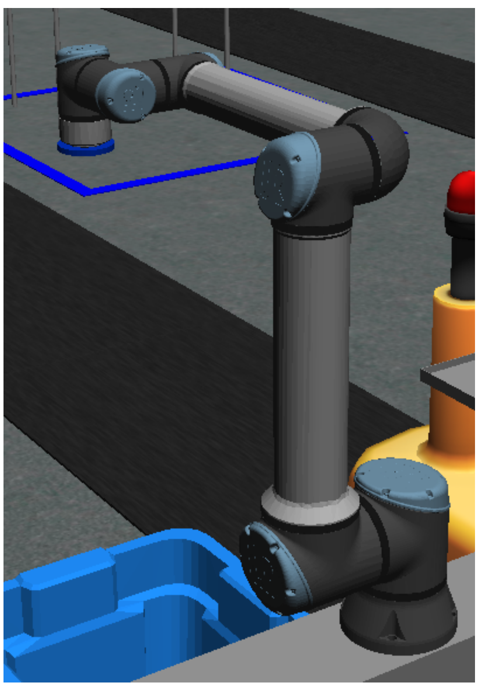


Figure A.9: An image of a robotic arm of the UR10e model taken from the Gazebo simulation.

There are two robotic arms in total: The floor robot and the ceiling robot. The floor robot is mounted and moves parallel to the conveyor belt, positioning it to pick up items from various locations, such as parts from the conveyor belt, bins, or kit trays from the two kitting stations.

The ceiling robot is mounted on the gantry. The gantry moves in the X and Y directions, allowing the ceiling robot great flexibility in reaching different areas. The gantry can also move around its torso. Due to its flexibility, the ceiling robot is able to perform both kitting and assembly.

Appendix B - Scoring in ARIAC

The ARIAC trial scoring system in based on three main factors 1) Cost factor, 2) Efficiency factor, and 3) Task score - how good was the system's solution. [34]. These following four sections describes the different parts of the scoring system in detail and will help with prioritising and understanding what needs to be done. All the following information, including equations, is taken from National Institute of Standards and Technology [34].

B.1 Cost Factor

The cost factor, CF, is conditional to the average cost of the sensors used in the system across all competitors. This average cost is what we call TC_{avg} in equation (b.1). The individual team's total sensor cost, TC, is then compared to the average cost and then weighted with a constant, ω_c. The full equation for CF is shown in equation (b.1). Equation (b.1) shows that one need to minimize TC to maximize CF, which is the wanted outcome.

$$CF = \omega_c \frac{TC_{avg}}{TC} \tag{b.1}$$

B.2 Efficiency Factor

The Efficiency factor EF_i is calculated for each task, i, and compares the time it takes for one team to complete a task compared with the average time it takes for all teams. Equation (b.2) describes how to calculate the score for a task, in which $T_{avg,i}$ is the average time it takes for the teams to solve the task, T_i is the time it takes for the individual team to solve the task, and ω_t is a constant weight factor. Equation (b.2) shows that to maximize EF_i, one has to minimize T_i.

$$EF_i = \omega_t \frac{T_{avg,i}}{T_i} \tag{b.2}$$

B.3 Task score

Each task (kitting, assembling, or combined) gets a score based on generated boolean conditions [34]. These following sections describes the boolean conditions for the different tasks.

B.3.1 Kitting Score

NIST describes the kitting scoring as[34]:

- A kitting task has n parts that need to be placed on the kitting tray.
- A shipment has m parts on the kitting tray. For each task there are two Boolean conditions:
  - $isCorrectTrayID \rightarrow A$: true if the shipment tray ID is correct.
  - $isCorrectDestination \rightarrow B$: true if the shipment was sent to the correct destination.
- For each quadrant q of the kitting tray there are four Boolean conditions:
  - $isCorrectType_q \rightarrow C$: true if the part type in quadrant q is correct.
  - $isCorrectColor_q \rightarrow D$: true if the part colour in quadrant q is correct.
  - $IsFlipped_q \rightarrow E$: true if the part in quadrant q is flipped.
  - $isFaulty_q \rightarrow F$: true if the part in quadrant q is faulty.

The task score consists of tray points (b.3a), quadrant points (b.3b), and bonus points (b.3c). There are also two penalties that remove can points form the final score: extra part penalty (b.3d) and wrong tray penalty (b.3e). There is also lastly a destination multiplier (b.3f) that makes sure that you only receive points if the shipment was sent to the correct destination. The equation to calculate the full task score can be seen in equation (b.4).

$$pt_t = \begin{cases} 1, if A \\ 0, otherwise \end{cases} \tag{b.3a}$$

$$pt_q = \begin{cases} 0, if \neg C \vee E \vee F \\ 3, if D \\ 2, if \neg D \end{cases} \tag{b.3b}$$

$$pt_b = \begin{cases} n, if \sum_q^n pt_q = n \times 3 \\ 0, otherwise \end{cases} \tag{b.3c}$$

$$pn_{ep} = \begin{cases} m - n, if m > n \\ 0, otherwise \end{cases} \tag{b.3d}$$

$$pn_t = \begin{cases} 1, if \neg A \\ 0, otherwise \end{cases} \tag{b.3e}$$

$$pm_d = \begin{cases} 1, if B \\ 0, otherwise \end{cases} \tag{b.3f}$$

(b.3g)

$$s = (pt_t + \sum_q^n (pt_q) + pt_b - pn_{ep} - pn_t) \times pm_d \tag{b.4a}$$

$$S_k = \begin{cases} s, s > 0 \\ 0, otherwise \end{cases} \tag{b.4b}$$

As can be seen in equation (b.4a), the destination multiplier ensures that the shipping destination is the most important aspect. As one can see in equation (b.4b) there can be no negative task score.

B.3.2 Assembly Task Score

NIST [34] describes the assembly task score as:

- "An assembly task has n parts that need to be assembled into the insert.
- $isCorrectStation \rightarrow A$: true if the assembly was done at the correct station (as1, as2, as3, or as4).
- Each slot s in the insert has the following Boolean conditions:
  - $isAssembled_s \rightarrow B$: true if the part in slot s is reported as assembled.
  - $isCorrectColor_s \rightarrow C$: true if the part in slot s is of correct colour.
  - $isCorrectPose_s \rightarrow D$:true if the part in slot s has the correct pose."

The assembly task has two types of points: slot points (b.5a) and bonus points (b.5b). There is also a station multiplier (b.5c). The full equation to calculate assembly score is seen in equation (b.6).

$$pt_s = \begin{cases} 0, if \neg B \\ 3, if C \land D \\ 2, if C \lor D \\ 1, if \neg C \land \neg D \end{cases} \tag{b.5a}$$

$$pt_b = \begin{cases} n, if \sum_s^n pt_s = n \times 3 \\ 0, otherwise \end{cases} \tag{b.5b}$$

$$pm_s = \begin{cases} 1, if A \\ 0, otherwise \end{cases} \tag{b.5c}$$

$$S_a = (\sum_s^n pt_s + pt_b) \times pm_s \tag{b.6}$$

B.3.3 Combined Task Score

A combined task is a combination of the other two tasks. NIST [34] describes the scoring of it as :

- "A combined task has n parts that need to be gathered from the environment and assembled to the insert.
- For each task there is one Boolean condition:
  - $isCorrectStation \rightarrow A$: true if the assembly was done at the correct station (as1, as2, as3, or as4).
- Each slot s in the insert has the following Boolean conditions:
  - $isAssembled_s \rightarrow B$: true if the part in slot is reported as assembled.
  - $isCorrectColor_s \rightarrow C$: true if the part in slot s is of correct colour.
  - $isCorrectPose_s \rightarrow D$: true if the part in slot s has the correct pose."

The combined task score consists of slot points (b.7a), bonus points (b.7b), and a station multiplier (b.7c). The full equation for the combined task score can be viewed in equation (b.8).

$$pt_s = \begin{cases} 0, if \neg B \\ 5, if C \land D \\ 4, if C \lor D \\ 3, if \neg C \land \neg D \end{cases} \tag{b.7a}$$

$$pt_b = \begin{cases} n, if \sum_s^n pt_s = n \times 5 \\ 0, otherwise \end{cases} \tag{b.7b}$$

$$pm_s = \begin{cases} 1, if A \\ 0, otherwise \end{cases} \tag{b.7c}$$

$$S_c = (\sum_{s}^{n} pt_s + pt_b) \times pm_s \tag{b.8}$$

B.4 Trial score

Lastly, there is a trial score, TS (see equation (b.10)), that combines the cost factor, CF (see equation (b.1)), efficiency factor, EF (see equation (b.2)), and task scores, S (see equations (b.4),(b.6), or (b.8)), into a single score. This score is then weighted with a priority multiplier (b.9). This score is then used to rank different teams [34]. The trial score is works as follows:

- "For each order there is one Boolean condition:
  - $isPriorityOrder \rightarrow A:$ true if the order is classified as a priority order"

$$pm_p = \begin{cases} 3, if A \\ 1, otherwise \end{cases} \tag{b.9}$$

$$TS = CF \times \sum_{i=0}^{n} (pm_p \times EF_i \times S_i) \tag{b.10}$$