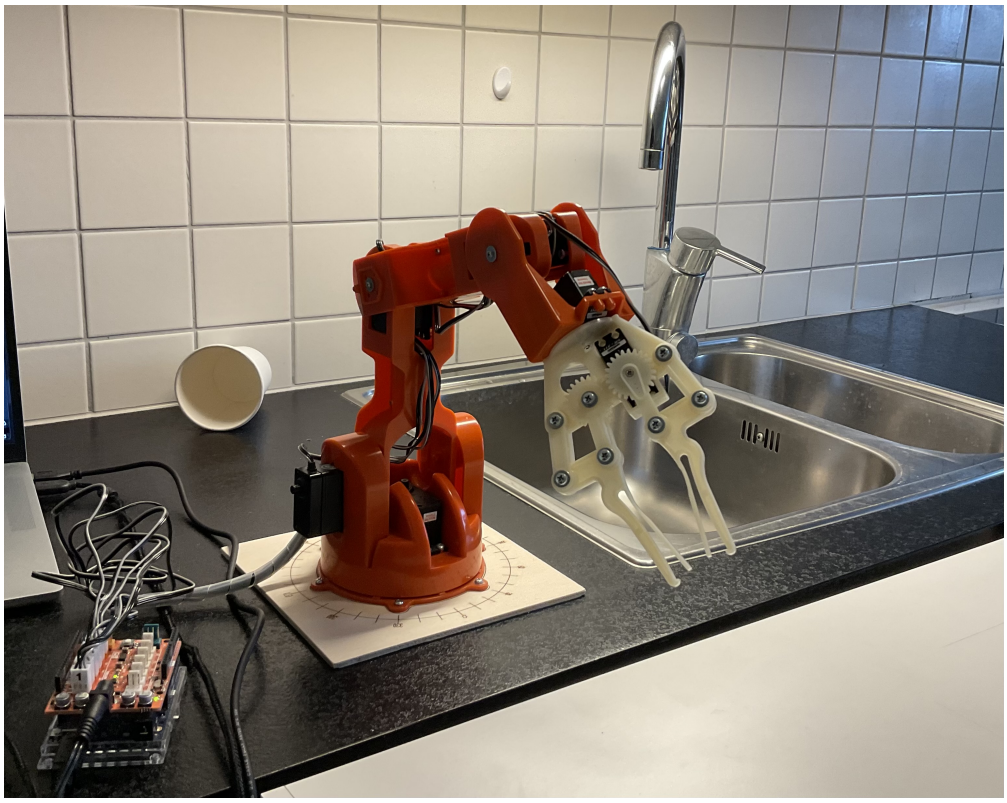
CHALMERS
UNIVERSITY OF TECHNOLOGY



The Development of an Articulated Robot Arm and and its Effectiveness in Performing Household Dishwashing Tasks

Degree project report in Computer Engineering

JOON SUH KIM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Degree project report 2025

The Development of an Articulated Robot Arm and and its Effectiveness in Performing Household Dishwashing Tasks

JOON SUH KIM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

The Development of an Articulated Robot Arm and
and its Effectiveness in Performing Household Dishwashing Tasks
JOON SUH KIM

Supervisor: Sakib Sistek, Research Engineer, Computer and Network Systems, Computer Science and Engineering
Examiner: Nicholas Smallbone, Researcher, Computing Science, Computer Science and Engineering

Degree project report 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Abstract

Dishwashing is a vital household task, necessary for maintaining hygiene and a clean kitchen environment. However, constant dishwashing is repetitive, time-consuming and in the long term can lead to symptoms of arthritis. This project aims to test, develop and evaluate different model architectures to determine which is the most effective at classifying cups and plates. This study also aims to develop an articulated robotic arm to pick up cups and plates, perform a washing motion under the sink, and place them in a designated drop-off zone. A CNN model with 3 convolutional layers, a ResNet18 model, a Resnet50 model and a VGG16 model were tested and compared based on accuracy, precision, F1-score, recall and inference time to determine which model is the most suitable to my classification model. Then, the robotic arm used this inference model to test the pick up, washing motion and drop off success rates on 25 trials for each dish object. The robotic arm successfully picked up cups 72% of the time and completed the washing phase 60% of the time, whereas plate handling success was significantly lower, with only 40% pickup and 12% washing success. These results suggest that the articulated robotic arm is more effective with cylindrical objects than flat objects with larger surface areas, highlighting the need for a new gripper design and new handling strategies for plates.

Keywords: Dishwashing, dinnerware, robotic arm, neural networks, classification model, ResNet, machine learning, image classification, practical uses of machine learning.

Acknowledgements

I would first like to express my gratitude to my supervisor, Sakib Sistek, who has provided the necessary resources for this project, consistently offered valuable feedback at the right times, and has been a great support and role model for me throughout my project.
I would also like to thank John Camilleri and Sanna Pitkänen, for being a second source of feedback and guidance during the difficult moments of my bachelor work.

Joon Suh Kim, Gothenburg, Aug 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ARA	Articulated Robotic Arm
CNN	Convolutional Neural Network
CTS	Carpal Tunnel Syndrome

Contents

1

Introduction

1.1 Background

Dishwashing is a vital household task, necessary for maintaining hygiene and a clean kitchen environment. However, it can also be perceived as tedious, time-consuming, repetitive and physically demanding. A study in the US has shown that people will approximately spend between 643 and 722 days of their lives doing the dishes [13], taking the assistance of a dishwasher into account. This time and energy could be spent towards more valuable means, highlighting the potential demand and benefits of automating this process.

The consistent developments in dishwasher technology have reduced the time spent on performing dishwashing household chores, yet has shown to be an inconsistent solution. For one, despite the consistent increase over the years, only around 49% of households in the UK own a dishwasher [16]. This proves that many people still rely on traditional hand washing for cleaning their dishes. Secondly, even after the invention of the dishwasher, doing the dishes remains a time-consuming task. This suggests that there is room for efficiency in this area, and that many people would benefit from an entirely autonomous dishwashing process, from washing the dishes to placing the clean dishes in a dish rack.

Moreover, it has been shown that repetitive tasks that make use of the hands and wrists can be detrimental to an individual's physical well-being. A study found that repetitive and/or forceful hand movements can lead to symptoms of Carpal Tunnel Syndrome (CTS) [3] [26], a medical condition characterised by pain in the hand, numbness, and tingling in the distribution of the median nerve within the carpal tunnel [1]. Dishwashing, a common household activity that involves both repetitive and sometimes forceful hand and wrist motions, can lead to many people being susceptible to CTS and causing physical harm to their hands and wrists [26]. Further studies conducted in a workplace context in Denmark [3] have shown that there is a high risk potential for CTS for jobs involving cleaning, which includes work related to dishwashing. The same study also showed that women are more prone to these symptoms [2], which has been similarly shown in studies involving housewives and high rates of hand and wrist pain, potentially caused by dishwashing. This shows that handwashing dishes is not only time-consuming, but can negatively impact the physical health of the person. Furthermore, having an autonomous robot performing dishwashing tasks can be beneficial to people who frequently have to do, who can be relieved of one of their daily tasks.

Additionally, dishwashing liquid is commonly used when washing dishes; prolonged

exposure to these chemicals can lead to irritated skin conditions. A study [9] has shown that workers in the cleaning industry (including dishwashing occupations) have an increased likelihood of developing dermatitis. The cause has been strongly linked to the chemicals present in common dish soaps, which strip away the natural oils present in the hands and can break the skin barrier. Other studies have also shown a correlation between frequent hand contact between hands and dish soap and cases of eczema and dry skin [7]. To prevent this, dishwashing gloves have been used in the past; however, they are not a complete preventative but rather a temporary solution around the issue.

1.2 Previous work

Use of robotic arms in a kitchen environment has been shown and tested in previous research [25]. This study [24] was conducted in a restaurant environment in Saitama Prefecture, Japan and has shown that using robotics to handle dishwashing tasks can lead to increased efficiency in the kitchen. Similarly, an autonomous robot in a domestic kitchen will not only improve the efficiency of washing the dishes, but also relieve the need for humans to perform these tasks, resulting in less risk of hand and wrist issues such as CTS. Furthermore, hand washing dishes usually comes with humans coming in contact with the chemicals in dish soap, and would therefore benefit from having an autonomous robot to handle these tasks.

1.3 Purpose

This research aims to design, model, and program an autonomous robotic arm (ARA) capable of performing automated dishwashing, thereby reducing the time required for this task in a household kitchen setting. This project aims to combine the concepts of machine learning and machine-oriented programming to implement a robotic arm that can perform dishwashing capabilities.

1.3.1 Goals

The goal of this project is to create a functional model, using machine learning techniques, to be able to move the arm, pick up objects, and place them in a set position.
The hardware elements of the project will have met its goals once the hardware is capable of the following:

- The robot arm can be moved manually
- The program can recognise and distinguish between different types of dishes
- The robot can pick up objects and place them in a designated area
- The actions of the robot can be reproducible and repeatable, while also performing equally well for different objects

The software elements of the project aims to attain these goals:

- The program can clearly recognise and distinguish between different types of dishes with high accuracy and recall

- Is able to perform image recognition tasks quickly and efficiently

1.4 Limitations

This project will be making use of the pre-built robotic arm the Tinkerkit Braccio Arm, meaning that the project is limited by the movement constraints in-built in the software. Plans to make use of a 3D printed three-finger gripper was executed and had reached the printing stage, but was ultimately discontinued due to the weak functionality of the 3d printed design. Benefits of using a 3D model is discussed in the "Further Experimentation" in section 8.

The tests conducted in this project will only be conducted on model paper tableware, such as paper plates and cups. This means that, when used in a practical setting, the ARA will be unable to lift heavy objects, due to the payload limitations of a model robot [21]. Instead, this project aims to show the capabilities of a robotic arm to perform the act of picking up a dish, washing it, and placing it back onto a rack. Further experimentation and research must be conducted to produce a physical, functional robotic arm that can be used for a real kitchen environment with realistic plates.

This project makes use of the Arduino Uno Wifi Rev2, resulting in the WiFi functionalities of the MIT App Inventor potentially unusable if another Arduino board type without WiFi functionalities is used.

The object detection software has been implemented using a pre-trained ResNet18 model (refer to section 3.2.3). Custom model training and dataset generation were removed from the scope to prioritise integration and prototyping.

Due to time constraints, developing an app from scratch was not only beyond the scope of this project, but also distracts from the focus of programming a classification for the robotic arm. For the development of the app, I will be using the block-based programming software MIT App Inventor (refer to section 4.1.5).

Unlike in the study conducted in Saitama Prefecture, Japan, this project will not be making use of live cameras to emulate the 3D mapping of a kitchen. This would require good knowledge of computer vision, which is beyond the scope of the program. Instead, a photo of the tableware will be taken prior to execution of the code, which would then be used as input data for the inference stage of my program.

The ARA was originally designed to have adaptive movements, allowing the arm to pick up objects from various positions in a pick-up zone and adjust to the location of the cup or plate accordingly. However, this also required heavy knowledge of 3D mapping and computer vision, which is beyond the scope of the knowledge taught in my program. Instead, the automatic movements were tested and programmed manually prior to execution, and the positions of the robotic arm when performing the dishwashing tasks were manually created during the testing phase (refer to section 4.2.2). This led to many restrictions and limitations in the program that are later discussed in section 6.2.

1.5 Ethical and Safety Concerns

The aim of this project is to design an articulated robotic arm (ARA) with the purpose of being installable in a household kitchen. However, the ARA used in this project is merely a scaled-down prototype, meaning it would not be used in a real kitchen setting for various reasons. This meant that the safety concerns were not a priority in this project, but rather the functionality of the robotic arm and its capability to perform dishwashing tasks was much more focused on.
In light of this, it is also important to discuss the safety and ethical concerns when working with machinery and programming that would affect people in the real world, especially when close to human proximity such as a household kitchen. The following safety concerns were considered:
The ARA can be dangerous if it works incorrectly and its rotational arms start bugging out, which can also be dangerous for the people nearby. ARAs should be able to fulfill its purpose, while also including in-built constraints to prevent the arm from performing violent movements. In a human setting, various factors could lead to the ARA malfunctioning, causing harm to the nearby surrounding and potentially other human beings. Therefore, an emergency stop option is also mandatory. This will be implemented through the use of a "safety" position that the robot arm will be placed in when a button is pressed, resulting in the arm being unable to move unless the program is restarted.
A dishwashing environment comes with frequent contact with water and cleaning chemicals. The robot should be built to be as waterproof as possible [18], to avoid any issues of damage to the robot and its surroundings.

2
Methodology

This section aims to outline the methodology by which the project was executed, including the software and hardware elements.

2.1 The software elements

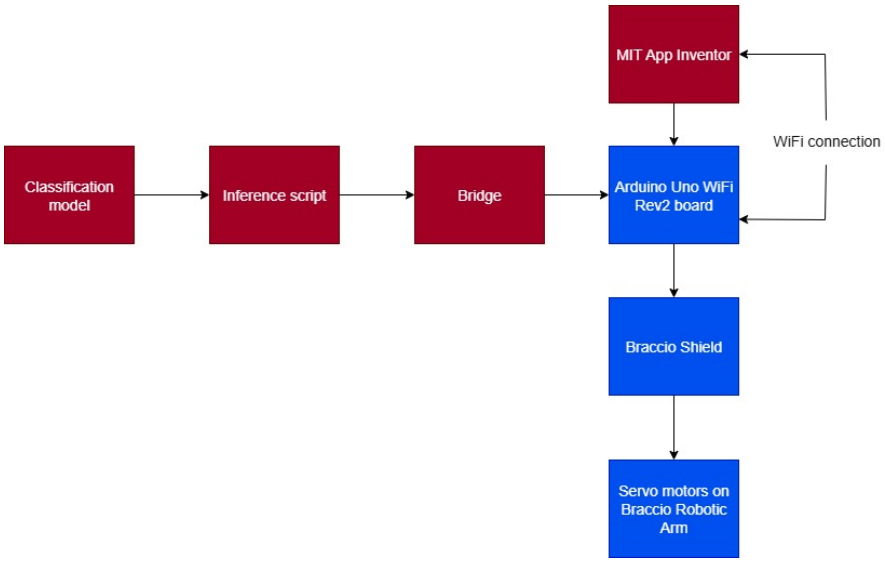


Figure 2.1: Showing the different components required for a complete execution of the robotic arm, including the software elements shown in red and hardware elements shown in blue.

This section aims to describe the process used to implement the software elements of the project, indicated by red boxes in Figure 2.1.
The ultimate goal of the programs used in this project were to send commands to the arduino board, which would then manipulate the movements of the robotic arm respective to the command given by the programs. These commands would be different depending on various factors: the classification model was designed to send a different output depending on which type of dinnerware was detected by the program (such as a cup or a plate). The subsequent low-level programming sketches

would then use these outputs as conditions to perform different dishwashing tasks respective to the signal input.

2.1.1 The Classification Model

A classification model was created to distinguish between different kitchenware, which in this project is focused on the distinction between cups and dinner plates. A dataset of cups and images were found online, separated into folders (testing, training and validation respectively) to appropriately label each image as either "0" ("CUP") or "1" ("PLATE").
Many different types of model could have been chosen for such a classification model, such as a recurrent neural network (RNN), convolutional neural network (CNN) or a multilayer perceptron (MLP). This project made use of the CNN model architecture [18], known for its image classification and recognition capabilities (see section 3.2.2). A baseline CNN model was initially tested to set a baseline accuracy that can be measured on the given dataset. This model made use of 3 convolutional layers, and had no pretrained weights included, but instead had randomly initialised weights. This baseline CNN was then compared with the other pretrained models of ResNet18, ResNet50 and VGG16, all of which had the ImageNet1K_V1 weight (refer to section 3.2.5). The most effective deep neural network necessary for the project was decided through experimentation on these aforementioned datasets.
All of these models were tested on its precision, recall, F1-score and accuracy. Then, another script for testing was used to test the inference time (the time it takes for the program to execute and recognise either a cup or a plate) for each respective model. The model that not only performs the best in the average of these categories, but also its relative inference speed compared to other programs, was chosen as the most suitable model for my classification model, and was continued to be used during the testing of the hardware.

2.1.2 Inference

After the testings were to be complete, an inference script was deemed necessary, in order to be able to receive new images and classify them either as a cup or a plate. This script is also crucial to measure the computational speed it takes from receiving an image input and outputting the correct label for the given image. This was deemed crucial mainly due to the potential scalability of the project, where the robotic arm may be expanded upon in further research to classify more classes of tableware (e.g. utensils, bowls etc,.) typically seen in a household kitchen environment. If the inference time is already recorded to be high when classifying between two objects, it is most likely going to increase when more classes are added to the model, therefore a low inference speed was highly valued in determining the final model used.

2.1.3 Communications with the Arduino sketch

The main_movements.ino Arduino sketch is the one that is most closely connected to the ARA. A low-level programming language such as C++ was needed for precise

control over the hardware and enable real-time, high-performance movements of the arm [17]. A higher level language such as python is not as efficient when it comes to computation closer to hardware, therefore the main_movements.ino sketch was written in C++ in the Arduino IDE.
In order to make use of my inference script and the main_movements.ino sketch, a "bridge" script was required to transmit the classification results from the inference phase to the robotic arm. This enabled the automatic pick-and-place movements of the ARA based on the predictions given by the inference script.

2.1.4 Manual movements of the robotic arm

In order to have the ability to manipulate, test and evaluate the manual movements of the ARA, a software was needed to isolate each axis of rotation. Storing both the image inference software and the control movements software of the arm on the same component (eg, a laptop) could result in bad abstraction and complexity in programming later during testing. For this reason, the software for the manual controls were developed on a separate hardware platform, namely a mobile phone.
The development of the software for this purpose should not distract from the main goals of this project, which are to test the models and develop the robotic arm. For this reason, a simple mobile app building platform was required, which allows for each axis of rotation to either be rotated clockwise or anti-clockwise depending on the type of button pressed.

2.2 2.2 The hardware elements

This section aims to discuss hardware elements used in this project, indicated by the blue boxes in Figure 2.1.

2.2.1 Arduino Uno WiFi Rev2

Seeing as the Braccio robotic arm is a product by Arduino [20], an Arduino board is the most compatible for this hardware. In order to communicate with the output received from the mobile application, a common WiFi signal was needed on the Arduino board itself. Therefore, the Arduino Uno WiFi Rev2 board [22] in specific was chosen for this project, which comes with an in-built WiFi module to communicate directly with the board, and consequently communicate with the Braccio robot arm.

2.2.2 Tinkerkit Braccio robot

The robotic arm model chosen for this project is the Tinkerkit Braccio robotic arm developed by Arduino [20], which will hereafter be referred to as the Braccio robot arm. This robotic arm comes with many benefits that are useful for this project:

- 6 axes of rotation: The arm makes use of two SR311 servo motors and four SR431 servo motors, which allows for the robotic arm to be controlled and moved in various ways.

- In-built constraints on movements: The Braccio arm has a maximum range of rotation for each axis, meaning that the arm parts will completely stop moving after a certain clockwise or anticlockwise rotation. This ensures safety in the usage of the robotic arm and its surroundings.
- Braccio shield: the Braccio shield is an external control board that can be used in combination with an Arduino board. Instead of connecting each individual servo onto each GPIO pin on the Arduino board, the Braccio shield simplifies this process by connecting all of its pins onto the Arduino board (see Figure 2.2).

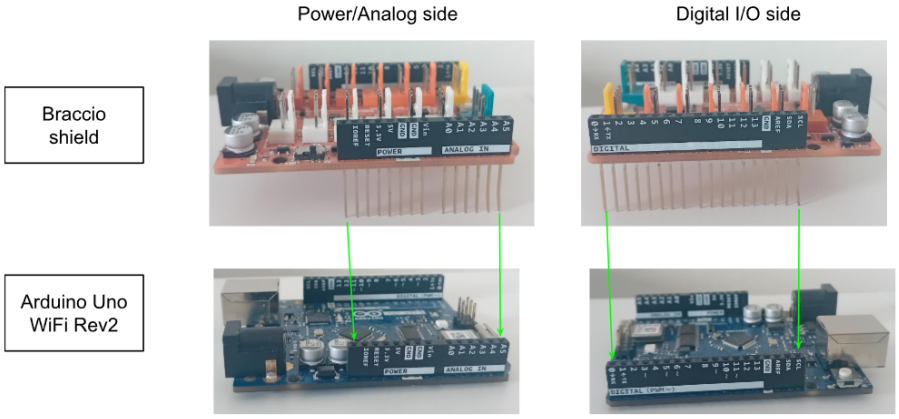


Figure 2.2: Illustrating the pins of the Braccio shield and how they connect to the Arduino board.

3
Technical Background

The components of a robotic arm required expertise from the following fields to accomplish:

3.1 Machine-oriented programming

Machine-oriented programming (MOP) refers to the low-level programming that is required to program closer to hardware. The following are the hardware components that benefit from having software that is lower-level:

3.1.1 The Arduino Uno Wifi Rev2

The Arduino Uno Wifi Rev2 development board [22], which is used to control the functions of the Braccio robotic arm, contains the following features:

3.1.1.1 ATmega4809

This is the main 8-bit AVR microcontroller that contains the flash memory, SRAM, Timers, Interrupt systems and GPIO pins. In this project, the servo motors make use of Pulse Width Modulation (PWM) signals, which produce a signal to indicate the angle of rotation the designated servo motor should turn. This allows the axes of the robotic arms to be moved, controlled and execute real-world motion.

3.1.1.2 u-blox NINA-W102 WiFi/Bluetooth module

The feature that distinguishes it from other Arduino boards is the WiFi and Bluetooth model, which allows the Arduino board to connect to an available WiFi signal or connect to another device via Bluetooth. The module makes use of the WiFiNINA library in Arduino [23], which includes the functions required to connect to the WiFi, store IP and passwords and check the signal strength. The use of C++ functions allows for the abstraction of the code that communicates with the Arduino board.
The WiFi functionality allows for the robotic arm to be controlled via an Android App on the mobile phone, which would be impossible with a standard Arduino board without an external module. For convenience and functionality, this was the reason for the WiFi version of the Arduino board being chosen for this project. The Arduino board is then able to communicate with a mobile app, resulting in this project being classified as an IoT system. Additionally, the use of an Android App

also allows for the use of the camera on a mobile phone, which will be used for the image recognition part of the program.

3.1.2 The Braccio Shield

The Braccio Shield [20] is the interface that connects the six servo motors and the Arduino Board, as illustrated in Figure 3.1. This interface is connected to the Arduino board via GPIO pins, which allows for the board to communicate with the servo motors on the robotic arm and make movements according to its input.

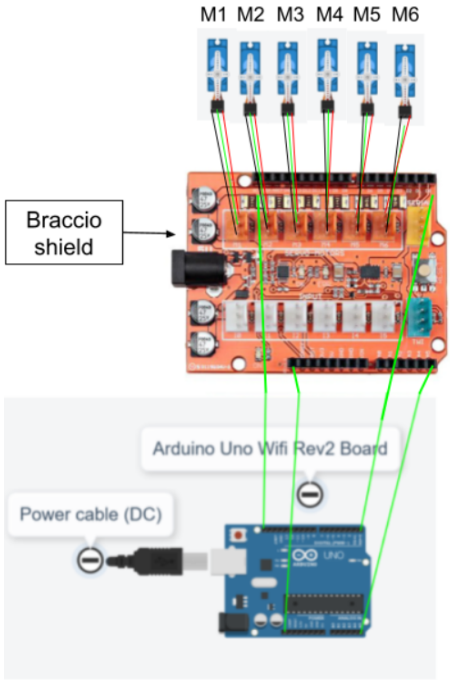


Figure 3.1: Showing how the braccio shield connects to the Arduino board, then how the 6 labelled servo motors are connected to the braccio shield.

The servo motors used for the robotic arm require more current than the Arduino board can supply, and therefore must be used alongside the Arduino board. The Braccio Shield also allows the use of the corresponding Braccio library on the Arduino IDE, which calls functions such as ServoMovement() to control the movements of the robotic arm.

3.1.3 Servo Motors

The Braccio Robot Arm makes use of six servo motors, two of which are of model SR311 and four are of model SR431 [20]. The SR 311 is an analogue servo used in the base, wrist, and gripper, whereas the SR 431 is a higher-torque variant used at the shoulder and elbow joints to handle greater mechanical loads. These servos make use of Pulse Width Modulation (PWM), which is a modulation technique that generates variable-width pulses to represent the amplitude of an analogue input signal [17].

Each servo motor is labelled M1, M2, M3..., M6, where each label represents a different axis of rotation for the robotic arm. These are then connected to the pins of the Braccio shield labelled M1, M2, M3..., M6 for each respective servo motor, which allows each servo to be controlled through the Braccio shield, the Arduino board and the code that is passed from the Arduino IDE.

3.2 Applied Machine learning

Machine learning is defined as the ability of an algorithm to learn from data and generalise from previously unseen data, and perform tasks without being explicitly programmed to do so [14]. Machine learning techniques have been applied to the image recognition tools needed for this project.

3.2.1 Classification model

A classification model is a supervised machine learning construct designed to assign input data to one of a predefined set of categories or classes based on learned patterns from historical data. In such models, training is conducted on a labeled dataset, where each input instance is associated with a known class label. The model learns a mapping function

$$f : X \to Y \tag{3.1}$$

where X represents the input feature space and Y denotes the set of discrete class labels [14].
During training, the model iteratively adjusts its parameters to minimize a loss function that quantifies the discrepancy between predicted and true labels. Once trained, the classification model is capable of performing inference on previously unseen data, producing either a discrete label or a probability distribution over the possible classes. Classification models are evaluated using metrics such as accuracy, precision, recall, and F1-score to assess their generalization performance.

3.2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) [19] is a class of deep neural network architectures designed to process data with a grid-like topology, such as two-dimensional image data or one-dimensional temporal sequences. CNNs are characterized by their use of convolutional layers, which apply learnable filters (kernels) across local receptive fields of the input to detect spatial patterns such as edges, textures, or shapes. This parameter-sharing mechanism reduces computational complexity compared to fully connected layers and enhances translation invariance.
A typical CNN architecture comprises of three main component types:

- Convolutional layers: perform the convolution operation to extract hierarchical features.
- Pooling layers: reduce spatial dimensionality, providing translation invariance and reducing computational cost.

- Fully connected layers: interpret the extracted features and output final predictions, often followed by a softmax activation for classification tasks.

CNNs are widely used in computer vision applications such as image classification, object detection, and semantic segmentation, as well as in non-visual domains like speech recognition and natural language processing. It has even been deemed as the "go-to solution for image based [machine learning]"[19], which is also the objective of this project.

3.2.3 Residual Networks (ResNet)

Residual Networks (ResNets) are a family of deep CNN architectures that address the vanishing gradient problem, where increasing depth beyond a certain point leads to decreased training accuracy due to vanishing or exploding gradients [10]. The key innovation in ResNet is the residual block, which incorporates a skip connection (or shortcut) that bypasses one or more layers by performing identity mapping. Formally, instead of directly learning a mapping

$$H(x) \tag{3.2}$$

where x is denoted as the input to the residual block, the residual block learns a residual function

$$F(x) = H(x) - x \tag{3.3}$$

allowing the original mapping to be expressed as

$$H(x) = F(x) + x \tag{3.4}$$

This formulation enables the network to learn modifications to the identity function rather than complete transformations, which empirically makes optimization easier and improves gradient flow in deep architectures.

3.2.3.1 Types of ResNet models

ResNet18 is a specific variation of the ResNet architecture composed of 18 learnable layers (17 convolutional layers and 1 fully connected layer) [10]. It represents the smallest variant in the ResNet family commonly used in practice, balancing model complexity with computational efficiency.
The architecture consists of an initial convolutional layer and pooling layer, followed by eight residual blocks arranged in four stages, each stage operating at a progressively lower spatial resolution and higher feature depth. Every residual block in ResNet18 contains two convolutional layers with batch normalization and ReLU activation, along with a skip connection that performs identity mapping (or projection mapping when dimensions differ).
ResNet18 is frequently used as a feature extractor in transfer learning scenarios, due to its low computational cost compared to other models and the ability to capture robust image features. When coupled with a classification head tailored to a specific dataset, ResNet18 serves as an effective CNN-based classification model

for a wide range of tasks, from binary classification (e.g., distinguishing between cups and plates) to multi-class recognition.
The ResNet50 model is fundamentally similarly built as ResNet18, such as the use of skip connections, and being pretrained on ImageNet [14] prior to fine-tuning for custom classification objectives. The difference is that the ResNet50 makes use of 49 convolutional layers and 1 fully connected layer, with a combined total of 50 trainable layers. The ResNet50 is known for its higher accuracy in image classification and processing, but also is computationally more demanding than the ResNet18.

3.2.4 VGG16

The VGG16 is another deep CNN model that is known for its image processing capabilities, popularised after its excellent performance in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [11]. It makes use of 16 layers with trainable weights (13 convolutional layers and 3 fully connected layers).

- All convolutional layers apply 3×3 kernels with stride 1 and padding 1, which allows the network to capture fine-grained spatial features while keeping the architecture consistent.
- Max-pooling layers (2×2 with stride 2) are inserted periodically to reduce spatial resolution while preserving important features.
- At the end of the convolutional blocks, the network includes three fully connected layers: the first two with 4096 neurons each, and the last with 1000 neurons for ImageNet classification.

One of the key design principles of VGG16 that distinguishes itself from the ResNet architecture is the use of deep but simple stacked 3×3 convolutions, which demonstrated that increasing depth while keeping convolution kernels small could significantly improve accuracy. Despite being computationally heavy and memory intensive compared to later architectures (like ResNet), VGG16 remains a foundational CNN model, known for its high accuracy of image classification. However, it struggles with training deeper networks due to not addressing the vanishing gradient problem.

3.2.5 Transfer learning

Transfer learning is a technique in machine learning where a model trained on one task is reused as the starting point for another related task [8]. Instead of training a network from scratch, which requires large datasets and significant computational resources, transfer learning allows for the feature representations (that the pretrained model has already learned) to be reused. For example, the ImageNet dataset contains 1000 object classes, over 1.2 million training images, 50,000 validation images and 100,000 test images [15]. By training convolutional neural networks on ImageNet prior to execution, it can extract general features such as edges, shapes, and textures that are useful across many vision tasks. By leveraging this prior knowledge, transfer learning speeds up training and can improve performance, especially when only a limited amount of data is available for the task.

3.2.6 Fine-tuning

Fine-tuning is the technique within transfer learning where the pretrained model is not only reused but also adapted by continuing training on the new dataset [8]. Typically, earlier layers that capture generic features are frozen, meanwhile deeper layers that have more features specific to the program are retrained or adjusted with a smaller learning rate. This selective retraining helps the model specialize to the new problem while preserving the useful representations it has already learned.

3.3 Other Elements

3.3.1 App Design

The MIT App inventor used in this project is a high-level programming platform, with the use of block-based programming to build apps from an interface. It also includes a drag-and-drop functionality with a multitude of features such as buttons, textboxes, switches, images etc,. This allows for the lack of need to program every feature from scratch, simplifying the app development process and allows for more allocation of time towards the main focus of this project, to develop the robotic arm and its classification software.
The app is also made to abstract the complex code that runs the robotic arm, in a way that people who are unfamiliar with programming can still make use of its functionalities.

4

Implementation

This section aims to describe the process of implementing the different components from the methodology into practice. It starts with an outline of the implementation of the software, such as how the images of cups and plates were collected for fine tuning, how the model architectures were tested and recorded, and how the app for manual steering was implemented using the MIT App Inventor. The second phase of this project discusses the implementation of the hardware components, how the arm's movements were controlled and how the efficiency of the robotic arm was tested and evaluated. This section also aims to clarify the relationship between the software elements of my project and how they were used in combination with my hardware elements.

4.1 Software Elements

The software elements in this project were implemented as follows:

4.1.1 Classification models

The classification models were the first software elements to be implemented, in order to evaluate the effectiveness of each model architecture before being used for the inference of images. These models were tested in a classModel.py script, making use of a configuration dictionary to test each individual model at a time. Figure 4.1 below shows four model types: A CNN model (the baseline model with 3 convolutional layers), a pretrained ResNet18 model, pretrained ResNet50 model, and a pretrained VGG16 model. These models were initially trained using the IMAGENET_1K weight [15], allowing for faster and more accurate classification of images. Then, the dataset of cups and images (refer to section 4.1.4) was used to fine tune the models, allowing for the classification models to specifically be trained for detecting cups or plates.

4.1.2 Collecting the image datasets

```
# ----------------------------
# Configuration (set parameters here)
# ----------------------------
config = {
    "model_type": "pretrained",      # cnn, pretrained (ResNet18), pretrained50 (ResNet50), pretrainedVGG
    "data_dir":r'C:/Users/joons/OneDrive/Documents/Chalmers/Machine Learning/cups_and_plates_data/data',
    "epochs": 10,
    "batch_size": 16,
    "lr": 1e-3,
    "data_aug": False,             # Enable data augmentation if True
    "save_path": "model.pth",
    "test": True,                 # If True, run inference on the test set
    "output_file": "output.txt"
}
```

Figure 4.1: The configuration dictionary used for the testing of the models, the testing variables such as epochs and batch size, the output files of the data and models, and the ability to run tests on the test set.

4.1.3 Inference on new data

To test the inference speed of the models, a testing.py was created, which imports the models trained in the classModel.py script (can be seen in Figure 4.2). A brand new image was taken of either a cup or a plate, then the path to that image was pasted into the code to test the time it takes to classify that image, similarly to which is seen in Figure 4.1. The "time" library in python was used to measure the run-time of the program, which starts at executions and ends when the label has been printed.

```
PS C:\Users\joons\OneDrive\Documents\Chalmers\Machine Learning> & C:\Users\joons\A
xe "c:/Users/joons/OneDrive/Documents/Chalmers/Machine Learning/testing.py"
Using device: cpu
Prediction: CUP (Inference time: 50.29 ms)
```

Figure 4.2: The print statement of the testing.py that shows how the file directory to the image was used to make an inference and note the inference time. The testing.py script would return a label of either "CUP" or "PLATE" and the time it took to make an inference on the image.

The inference.py script was created based on the functions used in the testing.py file. Two functions were defined in the inference script, as seen in Figure 4.3:

- load_model(): This function allows for the chosen model (in this case, the ResNet18. To understand the reasoning behind this model choice, refer to section 6.1.2).
- predict_image(image_path, model): This function enables the inference of the unseen image. It uses the file directory (image_path in this case) of the newly taken photo of either the cup or plate, then runs the chosen model to perform an inference on the new photo, to then return the predicted label as output.

```python
idx_to_label = {0: "CUP", 1: "PLATE"}

def load_model():
    model = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
    in_features = model.fc.in_features
    model.fc = torch.nn.Linear(in_features, 2)
    model.load_state_dict(torch.load("model.pth", map_location="cpu"))
    model.eval()
    return model

def predict_image(image_path, model):
    image = Image.open(image_path).convert("RGB")
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ])
    image_tensor = transform(image).unsqueeze(0)  # shape: [1, 3, 224, 224]
    outputs = model(image_tensor)
    _, pred = torch.max(outputs, 1)
    return idx_to_label[pred.item()]
```

Figure 4.3: Showing the functions developed in the inference.py script, with the intention of importing them to be used in bridge.py.

4.1.4 Communication Between Python and Arduino

The control of the Braccio robotic arm required real-time communication between the Python environment, where object classification and automation logic were executed, and the Arduino Uno WiFi Rev2 microcontroller. To achieve this, a custom Python script (bridge.py) was developed to transmit serial commands directly to the Arduino, as seen Figure 4.4. On the Arduino side, the C++ control program was adapted to parse incoming commands and execute the corresponding servo movements. From the python script, the functions made in the inference.py script were used to load the classification model and enable inference using the load_model and the predict_image functions respectively.

This setup allowed for integration between high-level decision-making in Python and low-level servo motor control on the Arduino.

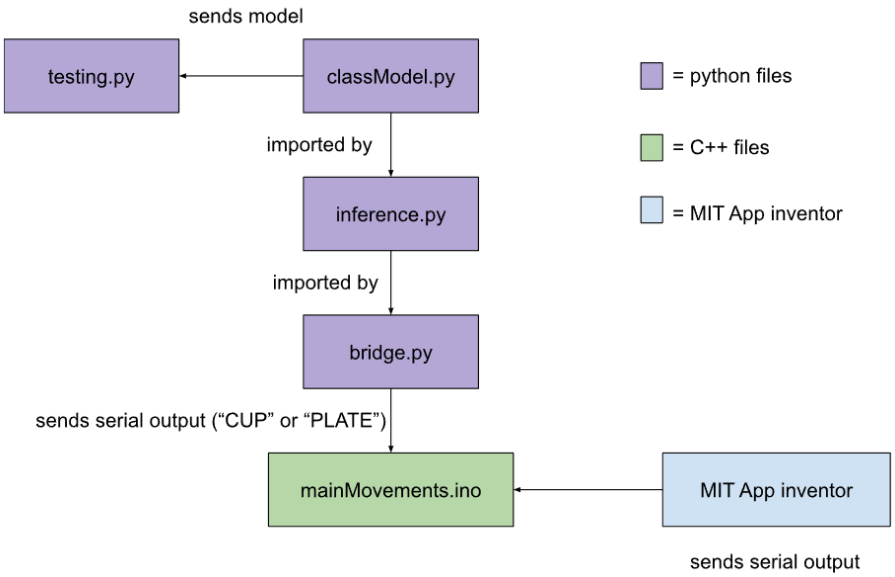


Figure 4.4: Showing the connection between the classModel, testing, inference and bridge python scripts, as well as the outputs delivered to the mainMovements Arduino/C++ sketch and the serial outputs sent via the MIT App Inventor.

4.1.5 Collecting the image datasets

The image datasets used for classification and training in this project are all images sourced from publicly available online databases, alongside a few that I have personally taken to provide additional diversity. To clarify, these images are also separate from the images used in the ImageNet-1K dataset, removing the risk of the classification model potentially “learning” from the same images. The images of the tableware are taken from different angles and directions, to increase variety and improve training of the ResNet18 model. These were then used for the fine-tuning of the classification models, to more accurately distinguish between the images of cups or plates.
The examples of datasets and their sources that are used in this project:
Cup.cv from the images.cv database [5]:

- Contents: 642 total images of coffee cups and mugs
  - 392 for training
  - 189 for testing
  - 61 for validation
- Image size: 256x256 pixels
- Data augmentation: none

Plate.cv from images.cv database [12]:

- Contents: 1300 images of tableware plates for training, testing and validation
  - 756 for training
  - 391 for testing
  - 146 for validation
- Image size: 256x256 pixels
- Data augmentation: none

Cup_mug_dataset from the Kaggle database [6]:

- Contents: Includes a total of 1044 images of cups and mugs
  - 900 images of cups and mugs for training
  - 144 images for validation
- Image size: 256x256 pixels
- Data augmentation: none

Cleaned vs Dirty from the Kaggle database [4]:

- Contents: Includes a total of 784 images of plates, both dirty and clean
  - 40 images for training
  - 744 images for testing
- Image size: 256x256 pixels
- Data augmentation: none

The images were then used to fine-tune the model architectures, with the purpose of improving the detection rate of cups and plates. In the case for the Kaggle datasets, the datasets would either lack a testing folder from the Cup_mug_dataset and a validation folder from the Cleaned vs Dirty dataset respectively. Therefore, the order of images dedicated to cups and mugs were randomised and half of them were delegated to training, whereas the other half would be used for testing. Then, the rest of the folders were combined accordingly, and including the images that I had taken, this resulted in the total dataset becoming:

Cups:

- 852 images for training (10 of them being my own)
- 639 images for testing
- 205 images for validation

Plates:

- 800 images for training (4 of them being my own)
- 1135 images for testing
- 146 images for validation

This new combined dataset was named the *Cups and plates* dataset accordingly.

4.1.6 Implementation of the app for manual movements

To enable manual operation of the Braccio robotic arm and easier methods of debugging any issues with the physical components of the mechanical arm (such as motors, grippers and connectivity to the Braccio Shield/Arduino microcontroller), a mobile application was developed using MIT App Inventor. When a button was pressed, the application sent a serial command to the Arduino Uno WiFi Rev2 via Wi-Fi signal. This WiFi signal was produced via a mobile Wi-Fi hotspot activated on the same mobile phone, which would connect to the WiFi module of the Arduino board.

The design of the application can be seen in Figure 4.5a and Figure 4.5b. Figure 4.5a shows an interface with directional control buttons corresponding to each joint of the arm, as well as the opening and closing mechanics for the gripper. The Arduino firmware parsed these commands and adjusted the corresponding servo angles in real time. Continuous movement was achieved by keeping the button pressed, allowing incremental changes to the servo position until the desired arm configuration was

reached. This would continually send the same serial output to the Arduino board until the button was released, resulting in a new position of the articulated robotic arm (ARA).
Figure 4.5b shows a simple interface with buttons corresponding to pre-set positions that the robotic arm would automatically move to when pressed. These positions were decided after the testing of the manual movements of the arm (refer to). In particular, the "Safety" position was also programmed so that the ARA stays in this position after the button was pressed, until the program is terminated. This was in line with the safety functionality named in chapter 1.5, giving the user an option to completely halt the program once pressed, including the automatic movements of the arm, and placing it in an upright position that is visible and unusable. The setup of these two interfaces provided a reliable and responsive method for testing and calibrating the arm, as well as performing tasks that were not pre-programmed in the automated sequence.

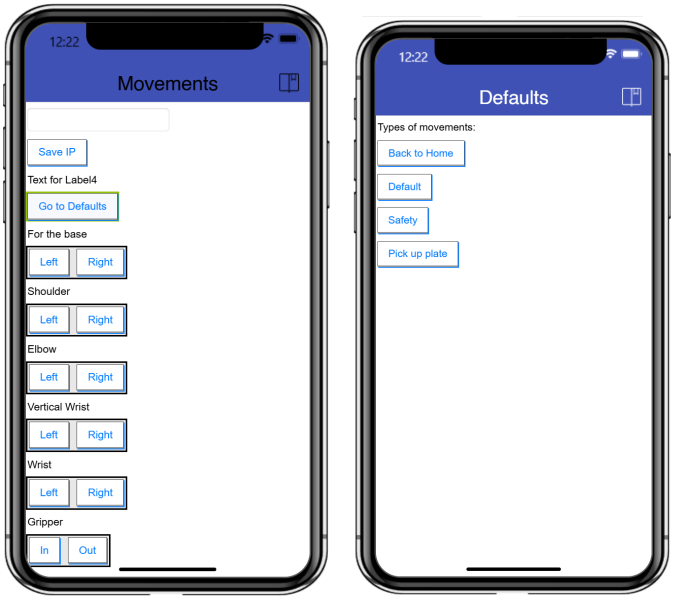


Figure 4.5: a) A screenshot (left) showing the interface of the "Movements" page developed on MIT App Inventor, showing the buttons used to steer each axis of rotation of the ARA.
b) b) The "Defaults" page (right) on the MIT App inventor application, showing buttons that can be pressed to move the robotic arm to pre-programmed positions

4.2 Hardware elements

Next, the following methods were executed to implement the hardware elements of the project:

4.2.1 Assembling the Robotic Arm

The assembly of the Braccio robot arm was done precisely as the instructions in the kit were given. The resulting product is the robotic arm seen in the diagram below.

Once the physical components of the robot were completed, the electronic components were connected. The pins on the Braccio Shield to the digital and input pins were connected to their matching Arduino Uno WiFi Rev2 board, taking up all of the pins on the arduino board to be used by the Braccio shield instead. Connect the six servo motors to the Braccio Shield by matching the label on the servo motor (M1, M2, M3 etc,.) with the appropriate position given on the board. Figure 4.6 below shows the complete assembly of the Braccio robotic arm, including the connections of the Arduino Uno WiFi Rev2 board, the Braccio shield and the 6 servo motors.

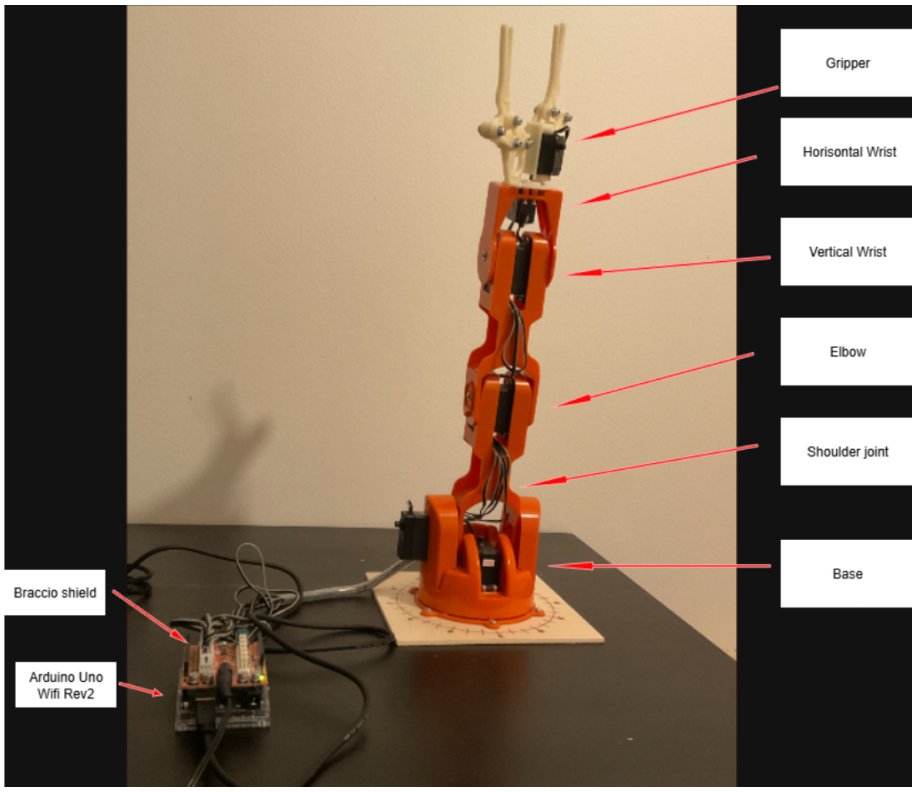


Figure 4.6: Image showing the complete assembly of the robotic arm, in the "Safety" position, and its 6 servo motors connected to a Braccio shield connected to an Arduino Uno WiFi Rev2 board.

4.2.2 Testing methods

In the absence of precise real-time positional tracking, the robotic arm was programmed to follow a sequence of pre-defined servo positions for key tasks:

- Pickup: Move from the home position to the object's location.
- Transport: Lift the object and rotate towards the sink.
- Washing: Position the object under a tap for a set duration, where the arm will move in a manner to replicate a handwashing motion. Here, the robotic arm will replicate a washing motion,
- Placement: Move to a designated storage area and release the object.

The app created on the MIT App Inventor was used to modify the positions and movements of the robotic arm, by holding down the buttons on the "Movements"

page (seen in Figure 4.5a.

The purpose of testing the manual movements was to develop the pre-programmed positions of the arm, to eventually be implemented into the automatic movements of the arm (the programmed positions would then be written into the code as seen in Figure 4.7. The testing methods for the manual movements of the robotic arm were as follows:

- Ensure that the computer and the mobile phone used in this project are both connected to the same local WiFi.
- Connect the DC power cable to the Braccio Shield and the USB-B cable to the Arduino Board.
- Run the Arduino code on the Arduino IDE, the IP that the Arduino Board is connected to should show as a serial print statement.
- Open the MIT App Inventor App, connect the app to the WiFi and enter the IP Address shown on the Arduino IDE.
- Test manual movement functions, ensure that all servo motor movements are working as intended.
- Start by identifying the coordinates for the "Default" position. Use this position to start both the movements to pick up a plate and a cup.
- Next, identify the position needed to rotate the base 90 degrees towards the direction of the tableware.
- Then, identify the positions required to pick up the plate, close the gripper, move the arm to be positioned underneath the tap in the sink, simulate a washing motion, then place the dish back in the drop-off zone in that order.
- Finally, use the MIT App Inventor app to test the default positions of the robotic arm ("Default", "Safety" position for emergency stops")

```
void moveToCupPosition() {
  Serial.println("Auto-moving to cup...");
  // Initial position

  basePos = 90;
  shoulderPos = 90;
  elbowPos = 90;
  wristVerPos = 90;
  wristRotPos = 90;
  gripperPos = 73;
  Braccio.ServoMovement(10, basePos, shoulderPos, elbowPos, wristVerPos, wristRotPos, gripperPos);
  delay (1000);

  //Rotate and face the cup
  basePos = 0;
  shoulderPos = 90;
  elbowPos = 90;
  wristVerPos = 90;
  wristRotPos = 90;
  gripperPos = 73;
  Braccio.ServoMovement(10, basePos, shoulderPos, elbowPos, wristVerPos, wristRotPos, gripperPos);
  delay(1000);

  // move the arm so that it is positioned to pick up the cup
  basePos = 0;
  shoulderPos = 80;
  elbowPos = 0;
  wristVerPos = 170;
  wristRotPos = 95;
  gripperPos = 10;
  Braccio.ServoMovement(20, basePos, shoulderPos, elbowPos, wristVerPos, wristRotPos, gripperPos);
```

Figure 4.7: Showing the pre-set positions of the Braccio robotic arm when picking up a cup, which allowed the robotic arm to move from position to position to simulate dishwashing. The first set of coordinates is the "Default" position, followed by a 90 degree rotation towards the cup in the pick-up zone, then the position to be stationed to pick up the cup.

4.2.3 Pre-programmed Movements for Dishwashing

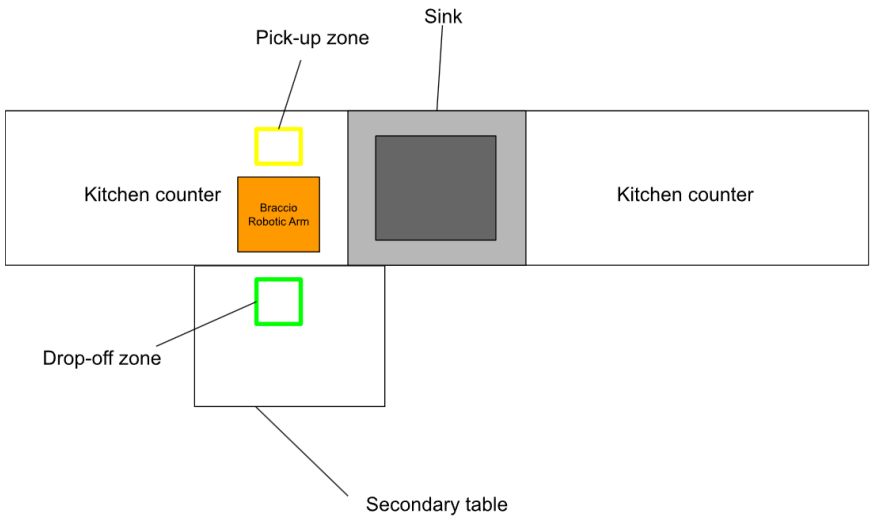


Figure 4.8: Showing the layout of the kitchen used for testing, the placement of the Braccio robotic arm, the pick-up zone, and the drop-off zone.

The testing methods for the autonomous movements of the robotic arm were as follows:

- Connect the DC power cable to the Braccio Shield and the USB-B cable to the Arduino Board.
- Run the Arduino code on the Arduino IDE, the IP that the Arduino Board is connected to should show as a serial print statement.
- Starting by choosing either the cup or the plate. Then, take a photo of the tableware in the designated pick-up zone (see methodology). Save this photo into the computer.
- Find the file directory to the photo and paste it into the bridge.py script. If done correctly, the file will send a serial to the Arduino file, which will then begin the pre-set autonomous movements of the Braccio robot arm of either the plate or the cup respectively.
- Verify whether it has successfully classified the correct object in the photo, picked up the dish, moved it under the sink to wash, then placed it on the drop-off zone.
- Repeat the previous step 25 times with the same dinnerware (cup or plate), then do the same step with the other dinnerware.

The method in which the robotic arm, the pick-up zone and the drop-off zone can be seen in Figure 4.8.

5
Results

The following section presents the results obtained from both the software and hardware components of the project. First, the performance of the classification models were evaluated on its accuracy, precision and inference time. Secondly, the results from the hardware implementation are presented, where the robotic arm was tested for its ability to execute the tasks based on the model's predictions. Together, these results provide an evaluation of the system's feasibility as a prototype for robotic dishwashing.

5.1 Results of training the classification models

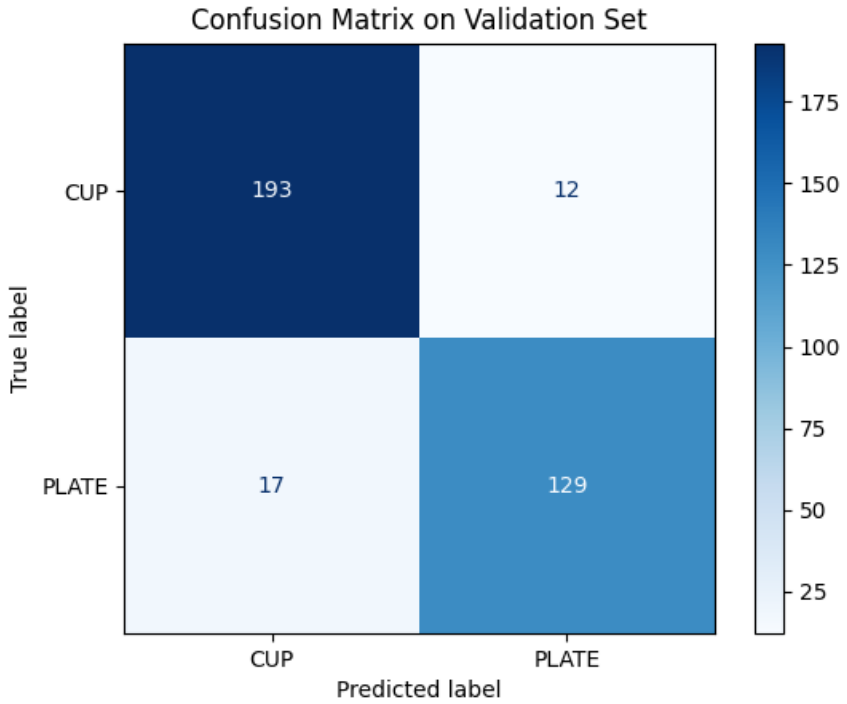


Figure 5.1: Showing the confusion matrix of the CNN baseline model trained on the cups and plates dataset. 193 images of cups were shown true positives, 129 plates returned as true negatives, whereas 12 cups returned as false positives and 17 plates were false negatives.

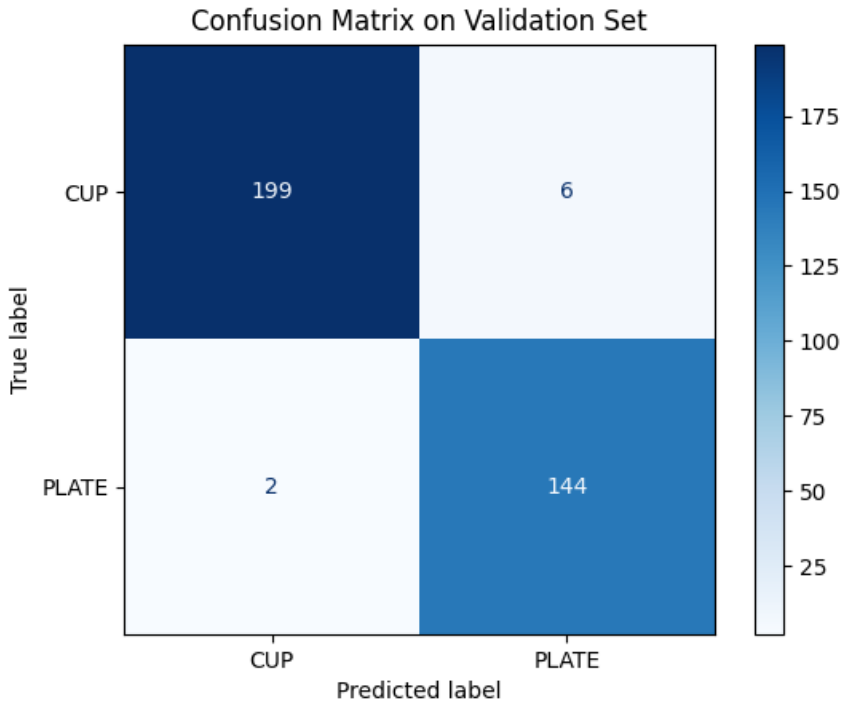


Figure 5.2: Showing the confusion matrix when the pretrained ResNet18 model was trained and fine tuned by the cups and plates dataset. 199 images of cups resulted in true positives, 144 plates returned as true negatives, whereas 2 cups returned as false positives and 6 plates were false negatives.

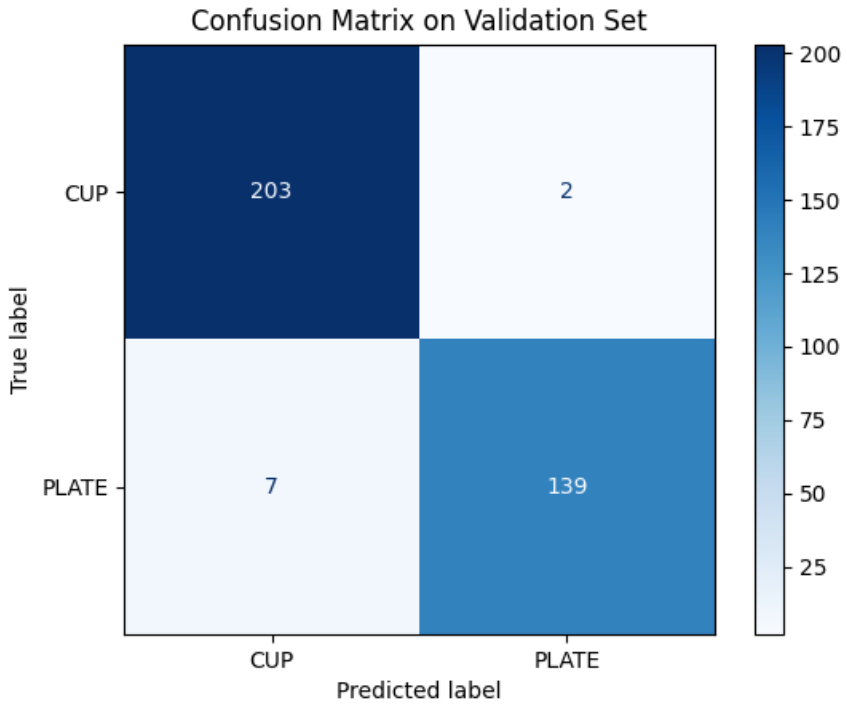


Figure 5.3: Confusion matrix when the pretrained ResNet50 model was trained and fine tuned by the cups and plates dataset. The model achieved 203 true positives for cups, 139 true negatives for plates, 7 false positives, and 2 false negatives.

Model	Class	Precision	Recall	F1-Score	Accuracy
CNN Baseline	Cup	0.9415	0.9190	0.9301	0.9174
	Plate	0.8836	0.9149	0.8990	0.9174
VGG16	Cup	0.9854	0.9806	0.9830	0.9801
	Plate	0.9726	0.9793	0.9759	0.9801
ResNet18	Cup	0.9902	0.9667	0.9783	0.9744
	Plate	0.9521	0.9858	0.9686	0.9744
ResNet50	Cup	0.9707	0.9900	0.9803	0.9772
	Plate	0.9863	0.9600	0.9730	0.9772

Table 5.1: Showing the precision, recall, F1-score and accuracy measured from the four different model architectures (CNN, VGG16, ResNet18 and ResNet50).

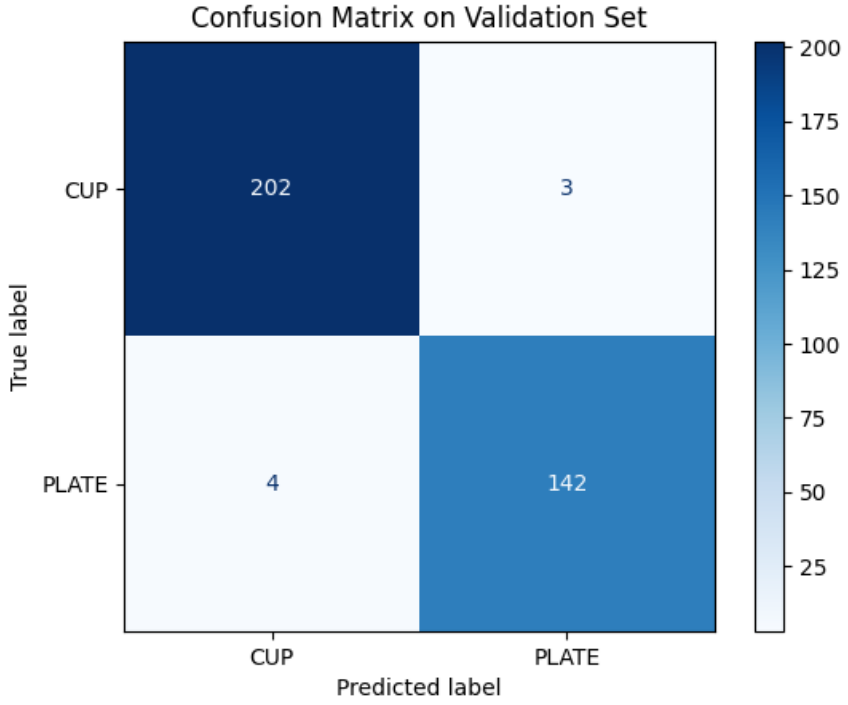


Figure 5.4: Confusion matrix when the pretrained VGG16 model was trained and fine tuned by the cups and plates dataset. The model achieved 202 true positives for cups, 142 true negatives for plates, 4 false positives, and 3 false negatives.

	Trials	CNN Baseline	VGG16	ResNet18	ResNet50
Inference time (ms)	1	10.17	183.82	25.47	76.43
	2	9.19	164.29	25.59	34.61
	3	10.92	158.51	30.26	38
	4	14.58	125.83	23.67	41.2
	5	9.74	227.12	28.38	45.6
	Mean	**10.92**	**171.91**	**26.67**	**47.17**

Table 5.2: Showing the inference time (ms) taken to identify a cup or a plate based on an image, and the average time taken from 5 trials for each model type.

5.2 Results from the hardware tests

5.2.1 Data for the trial runs of the articulated robotic arm

Number of times the cup was picked up: 18/25
Number of times the cup completed a washing procedure: 15/25
Number of times the cup was carried over to the drop-off zone: 11/25

Number of times the plate was picked up: 10/25
Number of times the cup completed a washing procedure: 3/25
Number of times the plate was successfully placed in the drop zone: 1/25

5.2.2 Common issues that were observed and experienced during experimentation

- Due to the shape and the strength of the gripper, the objects that were caught during the initial pick-up were occasionally not carried over to the final destination. This was especially prevalent with paper plates, but was also occasionally observed when paper cups were grabbed in suboptimal positions.
- Due to the shape of the gripper, it was difficult to grab the side of the plate without having the plate positioned on an elevated surface, in order for the gripper to reach both the top and bottom edges simultaneously.
- Due to the weight of the arm, it was a regular occurrence that the base of the robotic arm would tip over to one side, due to the weight distribution of the robotic arm when performing certain tasks. This issue was temporarily mitigated by having the base physically held down during experimentation.
- Experimentation involving the tap of the sink turned on was avoided, due to potential issues with the electrical components of the robotic arm getting into contact with liquids and potentially causing harm to the surrounding kitchen.
- The robotic arm was not able to grab onto the plate from the table, due to the shape of the gripper not being suited to grab plat objects from a flat surface. To mitigate this, the plate was placed on a cardboard roll, in order for the robotic arm to grab the plate.
- The robotic arm was unable to grab the paper cup, due to the cup rolling away from the original position due to its light mass. A paperweight (in this

case, a lock) was therefore used to prevent the cup from rolling, and therefore resulting in the ARA not being able to grab the cup.

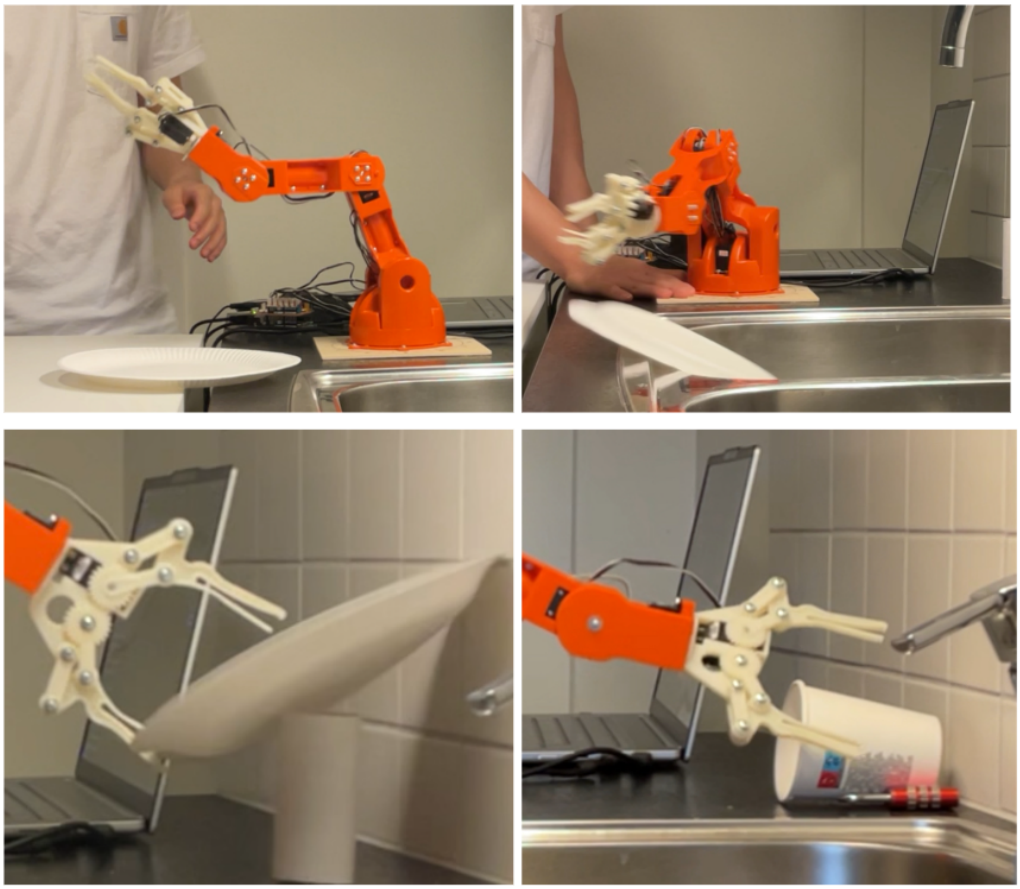


Figure 5.5: Showing common mistakes made during the experimentation of the robotic arm, such as (but not limited to): the plate getting dropped before reaching the drop-off zone (top left), the plate getting caught by the sink on its way to the drop-off zone and falling off (top right), the robotic arm's unsuccessful grab on the plate, causing it to drop (bottom left and the robotic arm not being able to grab the cup, due to the cup not being in the exact position of the programmed robotic arm. (bottom right)

6
Discussion

This section aims to evaluate the results of the testing of the models and the effectiveness of the articulated robotic arm (ARA) on its automated movements to perform dishwashing.

6.1 Evaluating the classification models

The models were evaluated based on their accuracy, precision, recall, F1-score, inference time and their confusion matrices. The pretrained models, in particular, were also measured by their improvements in these categories compared to the baseline CNN model.
In practice, however, some variables have been deemed too negligible to be taken into consideration in determining the most ideal model for this project. For one, the range of F1-score between the three models is

$$0.9830 - 0.9783 = 0.0047 \tag{6.1}$$

for cups, and

$$0.9759 - 0.9686 = 0.073 \tag{6.2}$$

for plates respectively,indicating a miniscule difference between the models. As for accuracy, the range is

$$0.9801 - 0.9744 = 0.0057 \tag{6.3}$$

for both cups and plates, which does not show a significant enough of a difference to be valuable in a comparison. Therefore, in order to perform a valuable comparative analysis on the pretrained models, the accuracy and F1-scores are deemed as negligible variables in determining the most suitable model for this project. Instead, the focus of the analysis was placed towards recall, precision, the confusion matrices and inference time.

6.1.1 Baseline CNN

Seen in Table 5.1, the baseline Convolutional Neutral Network (CNN) shows the lowest accuracy at around 91.7%, which is the lowest recorded out of the four model types.
When it comes to cups, the CNN model was able to recognise them at a high precision of 94.15%. However, there is a large imbalance between the way the program classifies cups and plates, seen in the significantly lower 88.36% precision

in classifying plates. This imbalance can also be seen in the confusion matrix of Figure 5.1, showing that a significantly larger quantity of 193 cups were labelled as true positives, compared to 129 plates being labelled as true negatives. One reason for this difference could be due to the ambiguous shape of a plate, which relatively lacks notable features to its shape that stand out from other objects, such as a mug with a handle would. The recall for plates being greater than its precision for plates further indicates that some cups are being misclassified as plates in this model. Furthermore, the CNN model was also not pretrained, meaning that it may not be able to identify specific details that distinguish the images of cups and plates as effectively as the other three pretrained models.

The benefits of using pretrained models can also be seen in the other categories, where the CNN model was outclassed in terms of recall, F1-score and accuracy relative to the other three models. The only category that the CNN model significantly excelled at in comparison to the other model is the inference time. As seen in Table 5.2, the model measured an average 10.92ms of inference, which can be mostly credited for the lack of the ImageNet weight in its training, and its few convolutional layers, allowing for the program to execute much faster than its competition.

However, even though it is able to complete inference quickly, it is not able to do so accurately, which is a detriment in this context. For one, it provokes a safety issue, where the misclassification of a tableware item could not only lead to a poor execution of the program, but also a safety issue, where a flawed pick-up of an object could lead to destruction of cups or plates in the worst case scenario. Also comparatively, the CNN model does not outclass any of the other pretrained models in terms of recall, precision, F1-score or accuracy, which resulted in the model not being used for the hardware phase of this project.

Compared to the CNN baseline model, the pretrained models are large improvements in terms of accuracy, where each model exceeds 97.4% accuracy. Similarly, the values for the F1-score, precision and recall for both cups and plates are all higher than the baseline CNN model, further supporting the claim that the pretrained models are an improvement upon a non-pretrained model.

6.1.2 The pretrained models ResNet18, ResNet50 and VGG16

In Figure 5.2, The ResNet18 model shows a 199 true positive rate as well as a 144 true negative rate, which is the recorded highest recall of plates. This was highly valued in this project, where a misclassification of a tableware can be potentially safety-critical with a model used for real plates and cups. This model is also recorded with the second lowest inference speed, with an average 26.67ms per execution, which was also a key factor.

However, the benefits of the ResNet50 cannot be ignored. Figure 5.3 shows that the ResNet50 the true positive and true negative quantity is very similar to the ResNet18, except for the true positives being slightly higher in the ResNet50 model and the true negatives being a few quantities lower than the ResNet18. Due to this, the differences in the models are negligible in terms of confusion matrices.

A similar pattern can also be observed in the VGG16 in Figure 5.4. The VGG16 predicted one less true positive than the ResNet50, but three more true negatives

compared to the ResNet50. These differences in numbers are too small to make a real comparison, therefore the confusion matrices were not used as a determinant when choosing the model.
Both the ResNet50 and the VGG16 outperformed the ResNet18 by small margins in terms of precision for plates and recall for cup, showing that they benefit from having a more balanced correct prediction rate of cups and plates, which was one of the criteria of the selection. The ResNet18, however, had the benefit of a stronger precision rate of 99.02% and a higher recall of plates than both ResNet50 and VGG16, balancing the advantages and disadvantages of using either model. The final criteria, the inference time, was eventually the deciding factor on which model to use moving forward. As seen in Table 5.2, the ResNet18 model had a significantly faster inference time at an average of 26.67ms, compared to the higher 47.17 average time for the ResNet50. This difference can be attributed to the larger number of convolutional layers used by the ResNet50, seeing as the decrease in number of convolutional layers has been shown to lead to decreased inference time, observed by the baseline CNN model. The significantly larger 171.91ms of average runtime of the VGG16 can be attributed to the model's design, which consists of sequential convolutional layers with large numbers of parameters and lacks skip connections. Therefore, in the end, the ResNet18 model was chosen for this project, due to its computational speed, which allows for fast inference and scalability for future developments of the model, while maintaining a good level of accuracy, recall and precision in all categories.

6.2 Evaluating the dishwashing capabilities of the robotic arm

Seen in the data from section 5.2.1, the cups demonstrated many successful trial runs, where 72% of the trials passed the initial pick-up phase. This shows that the robotic arm was relatively effective at picking up cups, most likely due to the shape of the cups. The occasional unsuccessful runs are likely due to the positioning of the cup in the pick-up zone, where the robotic arm would not be able to pick up the cup unless placed in a specific position, indicated by Figure 5.5. In contrast, the success rate of picking up the plate was only 40%, showing a significant decrease in effective usage of the robotic arm when picking up plates. This is most likely caused by the 2-finger gripper that comes with the Braccio robotic arm kit, which was mostly unsuitable for picking up flat objects such as plates.
Due to this flaw, the plates were also consistently dropped during the washing phase (when the plate was held under the sink tap), with only a 12% success rate. This indicates a crucial flaw in the physical components of the robotic arm, which drastically decreases the consistency and effective usage of the robotic arm. Similar unsuccessful runs have been recorded with the cups, where only 60% of the cups completed a washing motion. These unsuccessful runs were mostly caused by the movements of the robotic arm when performing the washing motion, and the tableware would fall to the sink due to the weak grip strength of the gripper caused by the in-built constraints on the Braccio robot arm. Even though these constraints have been previously beneficial to implement safety into the program, it has also

been shown to restrict the capabilities of the robotic arm, by not being able to fully close the gripper.
In the final drop-off phase, only 44% of the cups were successfully placed in the drop-off zone, completing an entire trial successfully. The ones that were unsuccessful were caused due to the cup landing on the table, but not within the designated drop off zone. Similar trends have been observed with the testing of the plates, where only 1 trial was completed successfully. In this case, the plates were seen getting caught inside of the sink (refer to Figure 5.5), before reaching the drop-off zone. The plate was also observed to be barely hanging by the gripper by the end of the washing phase, resulting in a high likelihood of the plate falling off of the gripper due to the weak grip strength of the gripper.
Overall, some successful trials were recorded, however the quantity of unsuccessful trials cannot be ignored. Constantly dropping objects when not intended can lead to damage to the tableware and the kitchen environment if objects are broken, therefore resulting in the robotic arm currently not being suitable for use in a real household kitchen.

6.3 Limitations and potential improvements

This section aims to identify the limitations of the project, and potential improvements that can be made to improve.

6.3.1 Creating a custom gripper

During the project, the two-finger gripper proved to be a significant limitation in this project, particularly when handling flat objects such as plates. Even though the gripper was relatively effective at grasping cylindrical shapes such as cups, its limited surface contact and its two-finger design caused frequent failures when attempting to pick up or hold plates with flat surface areas. This limitation could be improved upon by exploring alternative gripper designs better suited for diverse object geometries. A three-finger gripper was designed and in development during this project, however the development was ultimately discontinued. The design was deemed impractical for the intended application, and the base of the gripper was found to be incompatible with the designated servo motor, limiting its functional integration into the robotic system. A well designed gripper would improve the grip strength of the robotic arm, as well as overcoming the constraints placed on the original gripper that came with the kit.

6.3.2 WiFi Router Connectivity Issues

During experimentation, the Arduino Uno WiFi Rev2 was unable to connect to my local WiFi signal, due to security blockages with the router. This was a common issue that was prevalent among various wifi routers. Instead, a mobile network was used to connect to the board instead. In terms of usage, this does not affect the effectiveness of the program. It does however affect its scope with its compatibility with potential WiFi routers in the market. More research could have gone towards

finding a router that could have been used for this project, but was ultimately decided to not be a focal point of my study.

6.3.3 Pre-set Image in the Inference Script

During the early planning phase, several additional features were considered for implementation, such as the real-time camera tracking of the tableware.
An early design goal was to incorporate a camera-based tracking system capable of determining the 3D position of dishes in a kitchen environment. This function would have allowed the ARA to dynamically adjust its movements based on the detected location of a cup or plate. However, the implementation of real-time spatial localisation proved beyond the available technical expertise and project scope, as it required advanced computer vision, depth sensing, and motion planning algorithms not covered in the current programme curriculum. Instead, a sequential, step-by-step approach was taken, where the image was pre-taken, inputted into the inference model and then classified by the classification model, instead of having a live camera recognise the object in real-time and immediately label the object. This lead to various limitations with the program, including:

1. The program was unable to adapt if an object were to a different position in the pick-up zone, rendering the entire dishwashing procedure to be unsuccessful. The angle in which the photo was taken of the dish did not impact the movements of the robotic arm.
2. The process of taking the photo of the dish, adding the file directory of the image to the program and re-executing the program was an ineffective process that could be negated and made more efficient.

6.3.4 Pre-set automatic movements of the ARA

In terms of hardware limitations, the robotic arm was programmed specifically to operate within the layout of my kitchen. As a result, its functionality lacks flexibility when applied to different environments, where variations in the position of the sink and the placement of the robotic arm may prevent the system from operating as intended. To improve upon this limitation, computer vision can be used to dynamically detect the pick-up zone, sink and drop off zone, using a real-time camera and sensors. Another classification model will be needed to detect drop off zones, pick up zones and sinks. To add, cameras can be used to perform environmental mapping, allowing for the robotic arm to recognise the environment and be able to work around the surroundings.

7
Conclusion

The primary aim of this project was to develop a robotic arm system capable of performing basic dishwashing tasks through both manual and automated control. Specifically, the project aimed to achieve the following goals:

- Enable manual movements of the robotic arm via a custom mobile application.
- Implement a program capable of recognizing and distinguishing between different dinnerware types (cups and plates).
- Demonstrate the robotic arm's ability to pick up objects, place them under a tap for washing, and ultimately place them in a designated area.
- Ensure that the robot's actions are reproducible and consistent across different objects.

The developed system successfully meets a handful of these objectives. The robotic arm can be manually controlled through the MIT App Inventor application, providing the user a platform to perform real-time steering. The image recognition program is able to classify cups and plates with reasonable accuracy, although certain viewpoints—such as images taken from directly above—sometimes cause misclassification, notably confusing cups for plates. The robotic arm is able to execute the pre-programmed sequence of picking up, washing, and placing dishes on a simplified dish rack. However, while the actions are repeatable, the outcome is not perfectly reproducible, with some variation in object handling observed across trials.

Several limitations were identified during experimentation. The default gripper lacks sufficient strength and does not fully close, limiting the types of objects it can securely grasp. Additionally, the robotic arm's stability is compromised in certain positions due to its relatively small base and lack of fixed mounting, which can cause it to tip over. Finally, the image recognition accuracy could be improved by incorporating more advanced techniques such as 3D object detection or multi-angle imaging.

Overall, this project demonstrates a general idea of an automated robotic arm simulated in a kitchen environment. While the model does not yet achieve complete reliability or versatility, it lays the groundwork for future enhancements in mechanical design, image processing, and control strategies to potentially bring fully autonomous dishwashing robots closer to practical reality in the future.

7.1 Further Experimentation

Further experiments can be performed to improve the safety of using the robotic arm. When working with electronics in close proximity to water and chemicals, there

is a huge safety concern that needs to be addressed, especially when the product aims to be used in a household kitchen environment. The work done in this project can be expanded upon with the use of a design that fully encapsulates the robotic arm inside a waterproof casing, ensuring more safety from water coming into contact with wires.
The current classification model is also limited by only being able to detect cups and plates. Further experimentation can be done on a wider variety of dinnerware, such as pots, kitchen utensils, etc. Alternatively, further experimentation could focus on more specific types of cups -such as mugs, bottles, or wine glasses- as well as different kinds of plates, including soup plates, dinner plates, and bowls.
More developments can be made towards the mobile app, which is currently limited by the functionalities solely provided by the MIT App Inventor. Developing an application from the ground-up can be useful in the flexibility of the program, and eliminates the reliance on a secondary programming software, allowing the project to be more self-contained and adaptable.
More research can be conducted with a wider variety of model types unused in this project, for example the VGG19, Inception, EfficientNet etc,.
Finally, the project can be expanded upon by implementing a full-scale functional model that works in a realistic household kitchen environment. The current robotic arm is limited by its low payload [19], meaning that heavier objects such as plates, pots and frying pans are not suitable for the robotic arm used in this research.

Bibliography

[1] A. Genova, O. Dix, A. Saefan, M. Thakur, and A. Hassan, "Carpal tunnel syndrome: A review of literature," Cureus, https://pmc.ncbi.nlm.nih.gov/articles/PMC7164699/ (accessed May 20, 2025).

[2] A. Osinuga et al., ""Assessing the relationship between domestic work experience and musculoskeletal health among rural Nigerian women," PLoS One, vol. 17, no. 12, e0276380, 13-Dec-2022. doi: 10.1371/journal.pone.0276380. [Online]. Available: pubmed.ncbi.nlm.nih.gov/36512538 (accessed Aug 12, 2025).

[3] C. B. Lund, S. Mikkelsen, L. C. Thygesen, G.-Å. Hansson, and J. F. Thomsen, "Movements of the wrist and the risk of carpal tunnel syndrome: A nationwide cohort study using objective exposure measurements," Occupational & Environmental Medicine, https://oem.bmj.com/content/76/8/519 (accessed May 21, 2025).

[4] "Cleaned vs Dirty Plates Dataset", Kaggle. [Online]. Available: kaggle.com/datasets/gauravduttakiit/cleaned-vs-dirty (accessed Aug 12, 2025) .

[5] "Coffee cup Labeled Image Dataset - Free Download & High Quality Annotations | images.cv," Images.cv, 2025. https://images.cv/dataset/coffee-cup-image-classification-dataset (accessed Aug 12, 2025).

[6] "Cup & Mug Dataset", Kaggle. [Online]. Available: kaggle.com/datasets/malikusman1221/cup-mug-dataset (accessed Aug 12, 2025) .

[7] G. Jacobsen, K. Rasmussen, A. Bregnhøj, M. Isaksson, T. Diepgen, and O. Carstensen, "Causes of irritant contact dermatitis after occupational skin exposure: a systematic review," International Archives of Occupational and Environmental Health, vol. 95, 2021, doi: 10.1007/s00420-021-01781-0.

[8] IBM, "Transfer Learning," Ibm.com, Feb. 12, 2024. https://www.ibm.com/think/topics/transfer-learning (accessed Aug 12, 2025)

[9] J. Douwes et al., "Determinants of hand dermatitis, urticaria and loss of skin barrier function in professional cleaners in New Zealand," International journal of occupational and environmental health, `https://pmc.ncbi.nlm.nih.gov/articles/PMC6060852/#:~:text=Transepidermal%20water%20loss%20was%20significantly,loss%20of%20skin%20barrier%20function.` (accessed May 22, 2025)

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015. Available: https://arxiv.org/pdf/1512.03385 (Accessed Aug 27, 2025).

[11] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv.org, Apr. 10, 2015. https://arxiv.org/abs/1409.1556 (accessed Aug 29, 2025).

[12] “Labeled Image Datasets for AI & Computer Vision Models,” Images.cv, 2025. https://images.cv/download/plate/1300 (accessed Aug 12, 2025).

[13] Mulberrys Garment Care, “How Many Days Do You Spend Doing Chores in a Lifetime?,” Dec. 06, 2019. https://www.mulberryscleaners.com/blog/how-many-days-spent-doing-chores-in-lifetime/ (accessed Aug 12, 2025).

[14] Murphy, K. P. "Machine Learning: A Probabilistic Perspective," MIT Press, 2012. Available at: `https://www.cs.ubc.ca/~murphyk/MLbook/pml-intro-5nov11.pdf` (Accessed Aug 27, 2025).

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” International Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y (Accessed Aug 27, 2025).

[16] “Percentage of households with dishwashers in the United Kingdom (UK) from 1994 to 2018,” Statista, Jan. 2019. https://www.statista.com/statistics/289151/household-dishwashing-in-the-uk/ (Accessed Feb 3, 2025).

[17] P. Petersen Moura Trancoso, “Föreläsning 5,” Canvas, Jan. 24, 2023. Unpublished.

[18] R. D. Christ and R. L. Wernli, “Chapter 7 - Power and telemetry,” The ROV Manual (Second Edition) , pp. 141–161, 2014. doi:10.1016/b978-0-08-098288-5.00007-5. (Accessed Aug 29, 2025).

[19] R. Johansson, "Convolutional NNs," Lecture slides, DAT341 / DIT867 Applied Machine Learning, Chalmers University of Technology, Gothenburg, Sweden, Feb. 25, 2025. [Online]. Available: pdf. (accessed Aug 29, 2025).

[20] TinkerKit Braccio Robot — Arduino Official Store. Arduino. [Online]. Available: store.arduino.cc/products/tinkerkit-braccio-robot (accessed Aug 12, 2025).

[21] TinkerKit Braccio Robotic Arm Datasheet, Arduino, Scarmagno, Italy. [Online]. Available: docs.rs-online.com/6d63/0900766b814da230.pdf (accessed Aug 12, 2025).

[22] UNO WiFi Rev2, Arduino Documentation. [Online]. Available: docs.arduino.cc/hardware/uno-wifi-rev2/ (accessed Aug 12, 2025).

[23] UNO WiFi R2: Hosting a Web Server, Arduino Tutorials. [Online]. Available: docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-hosting-a-webserver/ (accessed Aug 12, 2025).

[24] W. S. Lo et al., "Developing a Collaborative Robotic Dishwasher Cell System for Restaurants," in Intelligent Autonomous Systems 16. IAS 2021, M. H. Ang Jr., H. Asama, W. Lin, and S. Foong, Eds. Cham: Springer, 2022, Lecture

Notes in Networks and Systems, vol. 412, pp. 238–248. doi: 10.1007/978-3-030-95892-3_20 (accessed Aug 12, 2025).

[25] Y. Fukuzawa, Z. Wang, Y. Mori and S. Kawamura, "A Robotic System Capable of Recognition, Grasping, and Suction for Dishwashing Automation," 2021 27th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), Shanghai, China, 2021, pp. 369-374, doi: 10.1109/M2VIP49856.2021.9665169. (accessed Aug 27, 2025).

[26] Y. Roquelaure et al., "Carpal tunnel syndrome and exposure to work-related biomechanical stressors and chemicals: Findings from the constances cohort," PLOS ONE, vol. 15, no. 6, Jun. 2020. doi:10.1371/journal.pone.0235051 (Accessed Aug 12, 2025).

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY