Robotatouille- An Application of Deep Learning for Autonomous Baking Using the UR5e Robotic Arm

Bec Egan, Beau Hobba and Talin Roche

Abstract— **Cooking is a fundamental daily task, requiring skill and time to produce the meals that sustain active and engaged lifestyles. Given the current Covid-19 pandemic, hygiene around food preparation is also of extreme importance. An autonomous, robotic solution would not only ensure safe food preparation and consistency in recipe execution, it would minimise the time required to prepare meals, allowing more time to be spent with family and friends. Robotatouille aims to relieve the stress of producing nutritious meals by autonomously executing a given recipe within the users own kitchen. Equipped with deep learning capabilities, Robotatouille is first introduced as a baking assistant, executing cake recipes to exhibit it's dynamic functionality in deciphering a recipe and executing all required cooking instructions. The system consists of an intuitive user interface, the UR5e robotic arm equipped with the Robotiq 2F gripper, the Intel RealSense D435 RGBD camera, an Arduino weight sensor and all the required utensils and ingredients. During demonstration, Robotatouille executed a simple series of actions, from deciphering a recipe instruction to grasping, measuring, pouring and mixing the required ingredient. This demonstration exhibits the successful integration of deep learning and depth analysis as components of a dynamic, autonomous cooking robot.**

I. Introduction

In the past decade, there has been significant advancements in the development of autonomous systems that can perform complex, high precision tasks with little to no human intervention. This development in conjunction with the surge of machine learning and artificial intelligence has led to a range of 'smart' autonomous systems that can perform tasks that previously required human engagement. Hence, there are a range of complex 'human performed' tasks that are now being shifted to autonomous execution by robotic systems.

Naturally, it is desirable to implement these technologies on a consumer level, automating commonly performed tasks for a large target audience. Hence, these huge technological leaps forward and the desire to bring them to consumer level tasks has led to the innovative implementation of the system exhibited in this report; a 'smart' cooking robot that can dynamically perform tasks and respond to variations in its workspace.

This paper will discuss the implementation of deep learning as a foundational component of the automated cooking process. The design shown will implement an RGBD camera, equipped with depth analysis to accurately search for and locate the required ingredients, as well as an integrated weight sensor such that ingredients do not need to be measured prior to operation. This is an important implementation feature as it will allow a higher level of autonomy in the cooking process. This level of autonomy is necessary for widespread adoption of such a technology, as it allows operation in realistic, dynamic conditions.

This paper is structured as follows. Section II will cover relevant existing literature in the fields of deep learning and robotic cooking applications. Furthermore, it will cover the motivations and goals driving this project. Section III will document the system architecture and design choices, including descriptions of the software and hardware modules. Section IV will present the results of system testing and demonstration followed by a discussion of the implications of these results within Section V. Section VI concludes the paper, discussing the future potential of the project.

Bec Egan, Beau Hobba and Talin Roche are with The School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, NSW 2020, Australia

II. Background

The aforementioned increasing development of autonomous systems naturally lends itself to complex and commonly performed human tasks. Hence, this complex autonomy is already used in cooking robots; technology capable of creating ready to eat meals.

A. Autonomous cooking robots

There are several emerging companies that address the concept of automated cooking. Two notable companies in this space are "Moley Robotics" [1] and "OctoChef" [2]. Moley Robotics created the worlds first fully robotic kitchen, designed to emulate human hands with a high level of precision and agility. Moley robotics has a focus on consumer level products, consisting of a fully self-contained system that can perform a variety of complex cooking tasks. These complex tasks are created by recording human movements with wired gloves and specialised 3D cameras. The recorded actions are then replayed by robotic arms equipped with 5 finger grippers. Despite this advanced technology and agility, Moley has no sense of the environment around it and thus, requires knowledge of the external conditions.

OctoChef focuses on work at an industry level and is primarily designed to produce the traditional Japanese dish, Takoyaki. Like Moley robotics, OctoChef also requires all ingredients, utensils and cooking surfaces to be in an exact position, however, it uses an overhead camera allied with artificial intelligence to detect when food is adequately cooked. This is a notably useful feature as it allows for imperfections in cooking temperature and other external factors whilst still cooking the food as desired. Hence, in this aspect, OctoChef cooking techniques are more robust than that of Moley robotics and more closely emulates the human process of cooking.

Both of these companies show a great ability to accurately perform complex tasks in a specific and consistent environment. Both OctoChef and Moley still require measured ingredients in pre-allocated positions, making operation in a dynamic kitchen space where humans and the robot coexist impractical. This shortcoming could be combated by the further employment of machine learning techniques, enabling dynamic sensing of the varying environment. Thus, in their current state, the systems are not fully autonomous and have significant limitations before they can be adopted for widespread use. These shortcomings have inspired the development and implementation of the autonomous cooking system described in this paper.

B. Deep Learning and Sensing the Real World

Deep Learning describes the process by which technological systems can utilise algorithms to determine patterns in large datasets [3]. Given the dynamic nature of the kitchen environment, methods of machine learning could be employed to enhance autonomy of the designed system. In order to have widespread adoption of such technology the system must work safely and seamlessly in the existing kitchen environment. This involves dynamically responding to changes to ingredient and object positions and ensuring safely while humans move throughout the workspace. Sensors which enable image, range and force detection allow for the interpretation of the surroundings. Thus, in conjunction with these sensors, deep learning based on visual imagery obtained from a camera could allow for further autonomy of the system.

Based on the vast potential applications of deep learning, many real-world implementations allow systems to perceive the world around them and inherently allow systems to perform complex automated tasks. One such application of a functional robot that utilises deep learning is from 'Abundant Robotics' [4], a company that designs robots to pick apples autonomously. A LiDAR is used to navigate between trees, and machine learning coupled with an on-board camera detects apples and their respective ripeness. This ideology, of using deep learning for specific 'food recognition' has been implemented in a variety of different ways. A paper "Image Recognition with Deep Learning" by M. T. Islam, B. M. N. Karim Siddique and S. Rahman and T. Jabid, presented in the International Conference on Intelligent Informatics and Biomedical Sciences [5], showed the application of image recognition with deep learning to classify foods. A dataset containing 16643 images of a variety of foods produced a classification accuracy of 92.86%. The applications of such a system could be realised in an automated dietary assessment system. Implementations are also available on the consumer level with applications such as "Calorie Mama" [6]. The mobile application enables images taken by the smartphone camera to be processed by a deep learning algorithm to track nutrition information specific foods. Another application is "June" [7] which uses a similar concept to OctoChefs deep learning to determine when food is cooked to a desirable level. June applies this technology to a stand-alone oven which can evaluate meals inside of it and recognise when they are finished. Thus, it is evident that the use of deep learning is viable for a range of implementations of varying complexity.

C. Goals and aims of the project

This project aims to demonstrate components of autonomous cooking, exhibited by the preliminary application of a baking robot. The project aims to identify ingredients in a dynamic setting by deep learning and depth analysis, capturing images with an RGBD camera. The project will also exhibit the breaking down of a recipe into functional components and the execution of each of these instructions. The system will measure ingredients as indicated by the given recipe and have a simple interface that could be navigated by an untrained user.

The deep learning module is trained to identify a given ingredient, regardless of the orientation or surrounding objects. Additionally, the depth image enables the calculation of the position of the resultant bounding box to output the ingredient in the 3D camera reference frame. Ingredients are to be weighed using a gram accurate weight sensor that provides real-time feedback. Finally, the resultant actions are performed by a millimetre accurate robotic arm to ensure consistent execution of all recipe instructions. Hence, the system should be able to adapt to varying circumstances to consistently perform tasks as expected.

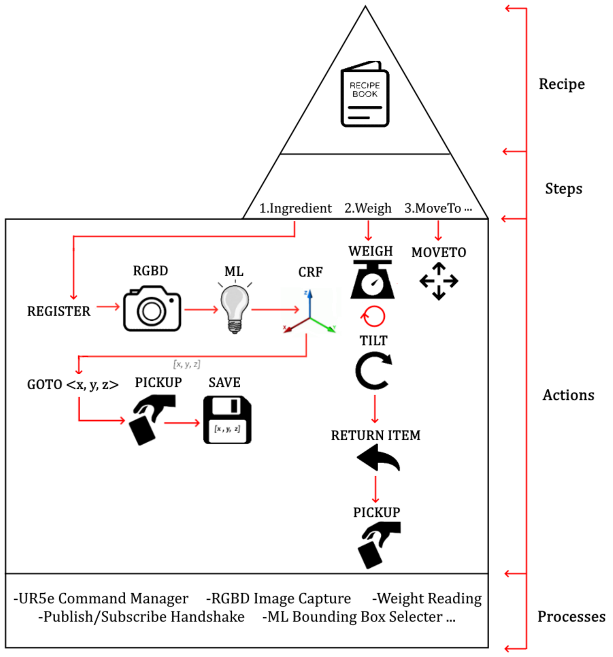

Fig. 1. Visual demonstration of the system architecture and the dynamic-natured Central Processing Hub. The top down approach follows the recipe, steps, actions and processes. Actions and processes work together, with actions defining what processes must occur in the software.

III. Experimental Setup and Design

A. System Architecture

The system architecture follows a top-down approach, where the recipe selected on the GUI described in section III-B is divided into manageable actions and processes. This structure encourages an extensible design such that new actions can be easily implemented as Robotatouille is developed further. New modules and actions are tested independently, prior to integration with other existing processes.

The central processing module connecting all modules is classified as the Hub. The Hub is responsible for decompiling a JSON recipe script into individual different steps including,*'Stir', 'Press', 'Preheat Oven', 'Mix', 'Pour', 'Bake', 'Roll', 'Combine', 'Grease', 'Refrigerate', 'Microwave', 'Whisk', 'Mash', 'Weight', 'Move' and 'Finish'*. These steps are then broken down into various actions. For example, the *Ingredient* step will *'Register' and 'goto'*. These actions are broken down into various processes, indicating low-level commands directly to the sensors and actuators. These processes can be both dynamic, such as obtaining an ingredient location based on depth and deep learning data, or static, such as moving the UR5e arm to a known location such as the mixing bowl. This hierarchy is demonstrated in Figure 1.

To communicate between the various sensors and the UR5e arm, the ROS publisher/subscriber method was utilised. The Hub is broken down into three separate access points to monitor these publishers and subscribers. The first, identified as the Central Hub is responsible for decompiling the given recipe. Secondly, an additional module interprets messages sent to the UR5e arm. Lastly, the final module interfaces the weight sensor with both the UR5e arm and the recipe decompiler.

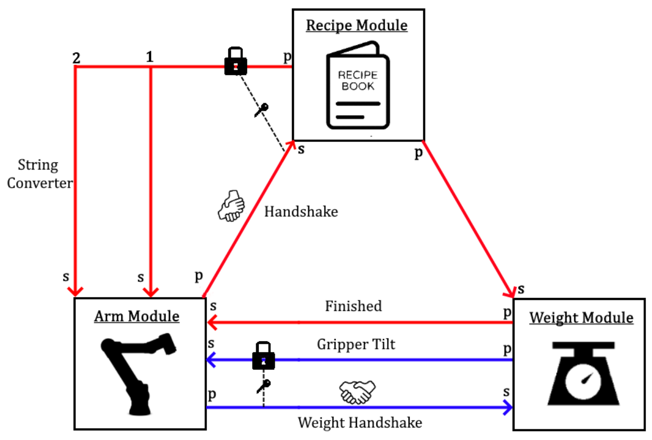

Fig. 2. Publisher and subscriber architecture

While the robotic arm is executing a movement, the functionality of the recipe and weight modules must be restricted to ensure the system is synchronous and delays in data execution are avoided. A lock and unlock system was implemented to avoid such a problem with handshake signals acting as the key. This system is represented in Figure 2. After the robotic arm has completed the given action, it provides a handshake signal which can unlock specific locks. This handshake only triggers if the subscriber has detected a published character. There are two locks, placed on the recipe module and the tilt change of the weight module. This weight lock is required as a component of obtaining feedback to accurately measure the required amount of an ingredient.

B. Graphical User Interface (GUI)

The GUI is designed using Tkinter, a python-based GUI library. Given the use of python in back-end code development, integration of the individual components is seamless. The GUI offers users the ability to view the available ingredients, methods and recipes. Users can then select an available recipe for execution. The GUI will inform users if recipes cannot be executed due to constraints in available methods or ingredients.

Once a user selects a recipe to be executed, the GUI will create a recipe JSON file, structured for simple decompiling by the Central Hub. The GUI will then publish a character to be read by the Central Hub such that execution of the recipe can begin.

C. Machine Learning

The machine learning module classifies each ingredient, constructing bounding boxes around each identified object. As a subset of machine learning, deep learning is utilised. Specifically, the use of a multiple-layer Convolutional Neural Network (CNN) is employed to extract features for identification. The archetype chosen for identification is object detection, analysing images to produce bounding boxes around the required objects. The system operates through supervised learning, with each object within an image in the dataset labelled into one of 11 classes; *butter, sugar, vanilla extract, self-raising flour, plain flour, cocoa, baking soda, salt, water, chocolate chips and vegetable oil*. Images were obtained in a controlled environment by mobile phones and the RealSense camera with varying closeness and orientation. *LabelImg* was utilised as the labelling tool [9] yielding a dataset of approximately 1750 labelled ingredient images.

The dataset is then split into 70% testing and 30% training sets. The model is pre-trained using the YOLOv3 network, chosen for its speed, generalisation and region specific classification [10]. Thus, the network will be effective in a live environment, capable of detecting changes between multiple classes and will potentially yield less false positive item identifications. YOLOv3 contains 53 convolution layers, as demonstrated in Figure 3 [10]. YOLOv3 is built around Darknet, a CNN which creates predictions using 3x3 filters which extract features and perform average pooling [10]. YOLOv3 replaces the last layer of the Darknet CNN, with three new 3x3 convolution layers [11]. These layers introduce 1000 class classifications, having been trained with the ImageNet dataset. The YOLOv3 trains these images at an accuracy of 93.3% after 160 epochs [11]. This allows the

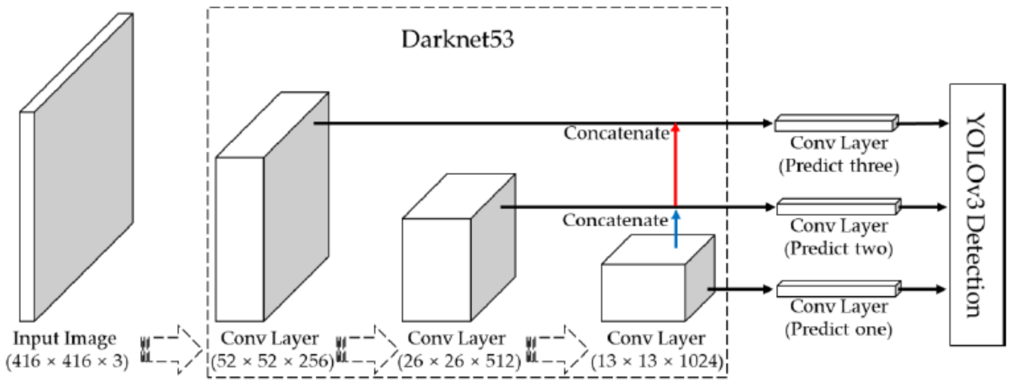

Fig. 3. You Only Look Once Version 3 (YOLOv3) Network Structure [8]

network to be useful at obtaining features and classifying objects. By transfer learning, the YOLOv3 is utilised as the foundation of the identification network. This increases the accuracy of the network and increases success in detecting new ingredient classes. Deep learning algorithms are developed through Keras, using the ImageAI python library. The model is trained over 40 epochs, using a batch size of 8. The batch size indicates how many images are passed into the CNN at a time, whilst epochs relate to when all batches having been passed into the CNN. The network repeats this procedure, incrementing the epoch count each time all images have been passed through the network.

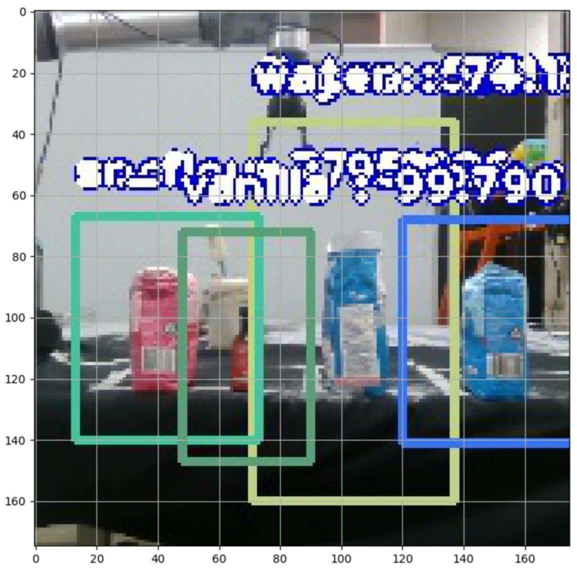

Fig. 4. All identified ingredients with labelled bounding boxes

The image input size for the network is 175×175. These dimensions best match the training data and depth sensor image. The training data collected by phone camera was under similar conditions; rotated and re-scaled to fit the input size. The CNN returns bounding boxes with associated labels and probabilities, indicating the certainty of each prediction. The accuracy is based around the error metrics of the model, and thus can be influenced by overfitting and underfitting. The returned result of the CNN is demonstrated in Figure 4.

The bounding box defines the location of the object within the image frame. In scenarios of inconsistent lighting, confused backgrounds and expected environments of grouped ingredients, multiple bounding boxes are generated. As stipulated by the Hub, only 1 ingredient should be detected at a time, and thus only 1 bounding box must be selected. Each label is observed to determine if it matches the required ingredient. Once identified, the label containing the highest probability is displayed and recorded for use in the depth analysis module. Figure 5 exhibits the identification and bounding of flour from the possible labels identified in Figure 4.

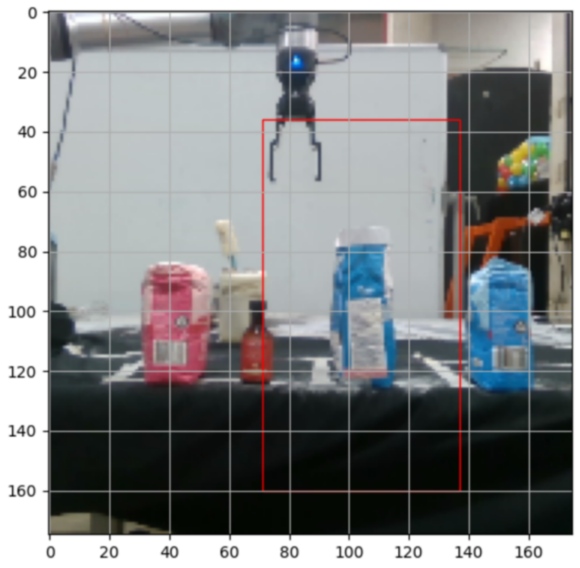

Fig. 5. Bounding box around identified ingredient

The error metrics of the system will be described through loss and mean average precision (mAP), calculated over the 11 classes. These metrics are generated by the imageAI evaluations tools as the model trains over 40 epochs. Google Colaboratory was utilised as a cloud based GPU for reduced computation time.

The loss function is comprised of classification loss, localisation loss and confidence loss. These functions use sum-squared error between the predictions and the pre-labeled images. Classification loss relates to how accurate the model is at classifying an object [11].

$$\sum_{i=0}^{S^2} l_i^{obj} \sum_{classes} (p_i(c) - \hat{p}i(c))^2 \tag{1}$$

Where l_i^{obj} is whether or not a object appears in the current cell (i), $p_i(c)$ is the probability of the class and $\hat{p}i(c)$ is the conditional probability of the same class [11]. Localisation loss relates to how closely the predicted bounding box corresponds to the labelled data set [11].

$$\begin{aligned} &\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B} l_{ij}^{obj}[(\sqrt{w_i - \hat{w_i}})^2 + (\sqrt{h_i - \hat{h_i}})^2] \\ &+\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B} l_{ij}^{obj}[(\sqrt{x_i - \hat{x_i}})^2 + (\sqrt{y_i - \hat{y_i}})^2] \end{aligned} \tag{2}$$

Where, l_{ij}^{obj} determines if an object is detected in the cell of the respective j boundary box. λ_{coord} is the weight for the loss and x, y, h and w correspond to boundary box coordinates [11]. Confidence Loss relates to if an object is detected or not. Due to variances in the background, this weight is reduced to ensure only desired ingredients are detected [11].

$$\sum_{i=0}^{S^2}(l_i^{obj}\sum_{j=0}^{B} l_{ij}^{obj}(c_i - \hat{c}_i)^2 \tag{3}$$

Where c_i is the confidence in the cell i of the j bounding box; $\hat{c}_i$ is the conditional alternative. If the result does not contain an object, a weight is a applied to reduce the significance of the loss [11].
The overall loss is the summation of the classification, localisation and confidence loss, describing how much error is in the model. Additionally, if there is little change of loss over many epochs, potential overfitting is indicated. Overfitting relates to when the model trains the data too well, and is then only able to correlate patterns to the data set, thus having no flexibility to variance.

maP was utilised to measure the average precision of all classes. Average precision was used to obtain the maximum precision of each separate class, with the term *precision* relating to all correct predictions for a class divided by the total amount of predictions (both correct and incorrect). This metric requires set parameters. The IoU (intersection over union), was set to > 0.5. This ratio indicates the predicted bounding box to the labelled data. If the intersection value is < 0.5, it is evident that the predicted box shows little similarity to the true label. The object threshold is set to 0.3, corresponding to the likelihood of an object being in the bounding box.

D. Depth Analysis

Depth analysis was utilised to determine the location of the specified ingredient in 3D space. Equipped with stereo cameras, the Intel RealSense camera captures both a colour image and a depth image. While the RGB image is utilised within the machine learning module, the depth image is then utilised to convert the depth data into the 3D location of the object with respect to the base of the robotic arm.

The camera returns a 640×480 pixel image, with the depth and RGB photo extracted independently. As explored in Section III-C, the input to the machine learning module is a 175×175 pixel image. To ensure little image data is lost, the image is cropped to 480×480 pixels, with 80 pixels cropped from both sides of the image. Given the image width and the required resolution for use in the machine learning module, this cropping technique limits the amount of ingredients that can be observed at one time. The image is then scaled down to 175×175 pixels such that the machine learning module can attempt to predict the ingredient class.

The deep learning algorithm returns a bounding box of 4 corner coordinates. The centroid of these coordinates is found by averaging the x and y components. As the depth sensor image is at the original resolution, this centroid value must be converted. Thus, it is scaled and then re-positioned according to the previous cropping. This is expressed through:

$$scale = \frac{R_C}{R_S} = \frac{480}{175} \tag{4}$$

Where R_C corresponds to the width resolution of the cropped image and R_S relates to the resolution of the scaled image. The centroid (C) of the image is found:

$$\begin{aligned} X_C^D &= (X_C^B * scale) + 80 \\ Y_C^D &= Y_C^B * scale \end{aligned} \tag{5}$$

X_C^D is the pixel value in the horizontal axis and Y_C^D is the pixel value in the vertical axis of the depth image. X_C^B and Y_C^B correspond to pixels in the bounding box. Now, the bounding box has been re-expressed in terms of the depth photo as demonstrated in Figure 6, and thus, the 3D coordinates of the classified ingredient with respect to the camera can be determined.

Calibration of the depth sensor was performed by both manual testing and the Intel RealSense tools. To calibrate the RGB camera the *Intel Dynamic Calibrator* application was utilised, alongside a custom print target, consisting of 10mm white and black bars. The depth data was calibrated using the On-Chip Self Calibration module on the *Intel Realsense Viewer* application. This consisted of pointing the RGBD camera to a wall of known distance. Testing the depth results of the camera yielded small error margins in both the x and y coordinates. This is likely due to misplacement of the RGBD camera with respect to the robotic arm world frame and imperfections with the sensor. To confirm, the UR5e arm is programmed to move to the correct position of the item. It was evident that the high precision required was not being met and thus, further calibration was necessary. An additional

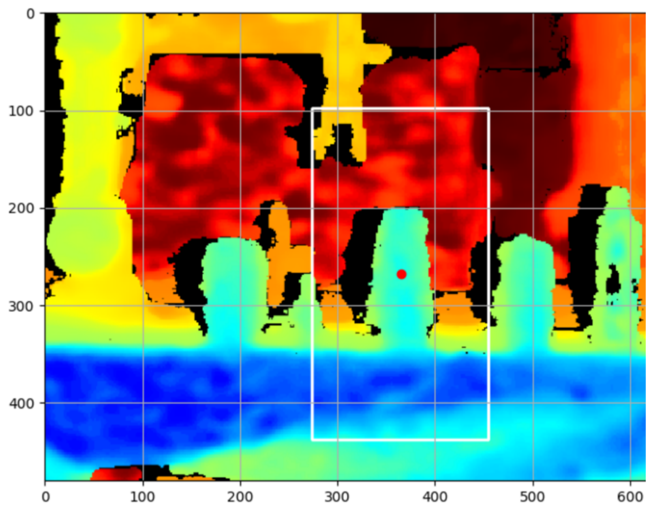

Fig. 6. Depth image with the centroid of the bounding box and centre of the respective ingredient shown

manual calibration involved the placement of objects at set 5cm intervals from the base of the robotic arm to obtain a dynamic equation aiming to reduce the potential error.

$$\hat{x} = 0.00027x^3 - 0.0049x^2 + 0.92x + 14 \tag{6}$$

$$\hat{y} = -0.0011y^2 + 0.92y + 3.8 \tag{7}$$

E. Coordinate Reference Frames

The Coordinate Reference Frames (CRFs) of the robotic system are defined by the x, y and z planes. Each rotation is performed about the XYZ axes in the given order. Translations are defined as homogenous equations in the form

$$t = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The world frame is defined as the centre of the base of the UR5e. This initial point is referred to as:

$$^{UR5e}P = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{9}$$

The object which is detected via the machine learning module must be able to be obtained by the UR5e arm. Thus, it must be converted from the reference frame of the camera to the robot reference frame. This requires the transformation:

$$^{UR5e}_{ING}T = {}^{CAM}_{ING}T * {}^{UR5e}_{CAM}T \tag{10}$$

ING refers to the ingredient and CAM to the RGBD camera. A diagram of this transformation is depicted in Figure 7. The camera is positioned at a static location with coordinates represented by:

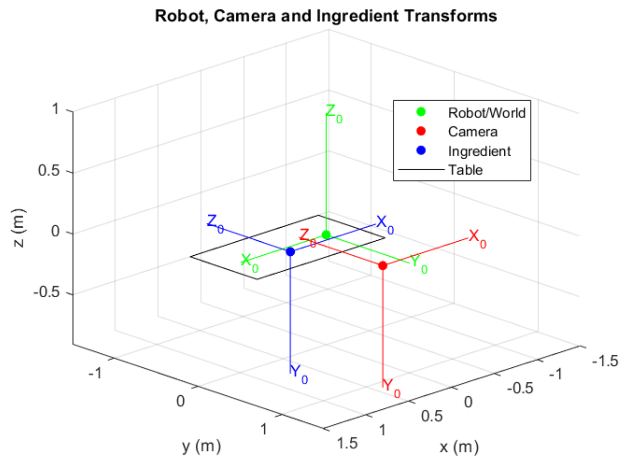

Fig. 7. CRF of the RGBD camera in the setup environment

$$^{CAM}P = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{11}$$

The camera follows a new reference frame, where the Z axis is normal to the lens. In the fixed location facing the table, the rotations are expressed as a $-90°$ roll rotation and a subsequent $-180°$ yaw rotation, given by:

$$\begin{gathered} ^{UR5e}_{CAM}R = Rx(\theta_{roll})Ry(\theta_{pitch})Rz(\theta_{yaw}) \\ Rx(\theta_{roll}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-90) & -\sin(-90) & 0 \\ 0 & \sin(-90) & \cos(-90) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ Ry(\theta_{pitch}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ Rz(\theta_{yaw}) = -1 * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{gathered} \tag{12}$$

The transformation from the base frame of the robot can then be calculated by the translation to the camera, and the camera rotation.

$$^{UR5e}_{CAM}T = {}^{UR5e}P + {}^{UR5e}_{CAM}R^{CAM}P \tag{13}$$

Finally, the depth sensor returns the x, y and z position of the ingredient with respect to the position of the camera. Thus, it requires only a translation from the point of capture. This translation is expressed as:

$$^{ING}P = \begin{bmatrix} x_{DEPTH} \\ y_{DEPTH} \\ z_{DEPTH} \end{bmatrix} \tag{14}$$

Thus, the final transformation from Equation 10 is expressed as:

$$^{CAM}_{ING}T = {}^{CAM}P + {}^{ING}P \tag{15}$$

Hence, the position of the ingredient is now expressed in the robotic arm reference frame, allowing a direct correlation between the ingredient identified by deep learning to the movement of the robotic arm to grasp the ingredient.

F. Robotic Arm Control

The robotic arm chosen is the Universal Robots UR5e, a flexible collaborative robot utilised widely in industry, equipped with the Robotiq 2 finger gripper [12]. To access the full capabilities of the robotic arm, the Robotic Operating System (ROS) is used to interface with the UR5e. For the purpose of performing actions required in the baking process, the arm is required to move to given locations in a controlled manner. Thus, MoveIt!, a motion planning and manipulation toolkit for ROS is used to control the arm. This provides frameworks for planning Cartesian paths based on set waypoints and constraining orientation during motion. Additionally, Gazebo and Rviz are utilised for simulation and motion testing to ensure efficient and safe movements prior to application on the physical robot.

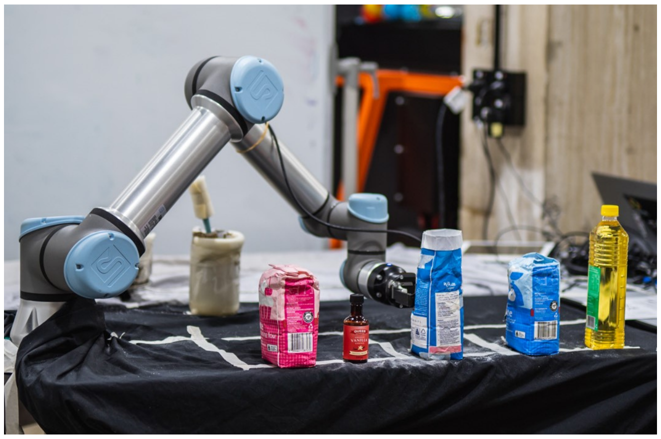

Fig. 8. UR5e in operation, grasping an ingredient

The code for control of the arm is developed in Python, broken into individual action functions to be called by the Central Processing Hub. These actions include grasping ingredients, moving to the scales for measuring, pouring the measuring bowl into the mixing bowl, grasping a utensil, mixing, whisking and returning ingredients to their original position. Additional action functionalities include turning the oven on to a selected temperature, pouring the mixing bowl into a cake tin, opening the oven door, placing the cake tin in the oven and closing the oven. Utilising similar motions, the UR5e is also equipped with the capability of retrieving the cake from the oven once a set amount of time has passed. Each of these actions is comprised of a series of movements of the end effector to a desired pose, movement of specific joints and the opening and closing of the gripper. The pouring motion is enabled by iteration of the current joint angles. Functions for the tilt of the gripper and the wrist joint enable the pouring motion in both axes required.

The Cartesian motion path is defined by setting waypoints to ensure the approach to the desired location is unobstructed. The target positions are defined both dynamically, such as the location of the required ingredient, and statically such as the position of the scales for measuring. As such, all functions are defined with the required parameters for execution. The random nature of the search for the optimal path in MoveIt! adds an element of uncertainty to the requested motion, combated by the use of a central 'home' location, defined such that motions from this point operate in a desirable kinematic manner. Orientation constraints are an important component in ensuring controlled motion, particularly once an ingredient has been grasped and is being moved to a desired location. Once grasped, the current orientation is recorded and set as an orientation constraint. This is enforced until it is either cleared or a new orientation goal is given.

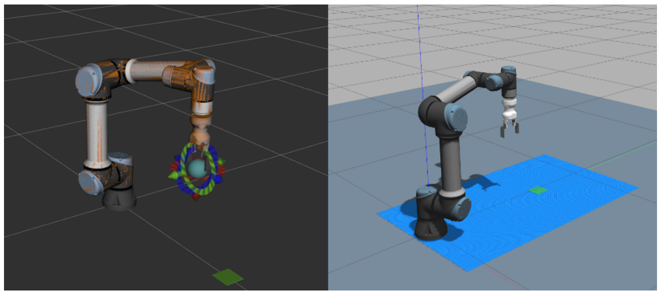

Fig. 9. Rviz (left) and Gazebo (right) for simulated robotic arm motion

Prior to execution on the physical UR5e arm, all functions were run on Gazebo and Rviz. The output of the MoveIt! path planner exhibits the certainty in computing the path to the desired locations and orientations. Thus, some positions were deemed difficult to reach given the kinematic constraints of the robot. As such, this testing procedure enabled the full execution of motion in simulation. Once satisfied with the simulated motion, the UR5e arm and the Robotiq 2F gripper were connected. By implementing a required user input before each motion, Rviz was utilised to confirm the success and safety of each movement prior to execution. After rigorous testing with a range of dynamic ingredient placements, these safety stops were removed for smooth, autonomous operation.

G. Weight Sensor

The weight sensor is built using a gram accurate weight cell, accurate for up to 1kg load. The direct output from the weight cell is sent to an Arduino to read the raw data and output it as a serial output to the Hub. An Arduino is used as it provides the necessary two analogue input ports as well as a 3.3V output.

A calibration/zero function is initially run in order to determine the relationship between the raw data and the resulting weight. The calibration function requires a zero weight and then an additional weight offset. This is only required on initial setup.

The nominal weighing operational is executed when a weight is requested by other processes. The code will average every five readings and publish the resulting weight.

The second weight module component interprets a desired weight setpoint and commands the UR5e tilt such that the ingredient are poured to the correct weight. This code will command the UR5e arm to increment the tilt of ingredients at a set rate of 5 degrees per second. As the ingredient approaches the threshold of the required weight, the UR5e tilt is commanded to revert by 10 degrees and will then increment the forward tilt at a lower rate of 1 degree per second. This allows for consistent and accurate weighing of ingredients.

IV. Results

A. Machine Learning Training

The machine learning model was successfully trained on the provided dataset. Training of the network took approximately 1116s per epoch with a step of 742ms. For 40 epochs, the training process took 12.4 hours to complete. The loss of the final model was 3.64, as indicated in Figure 10.

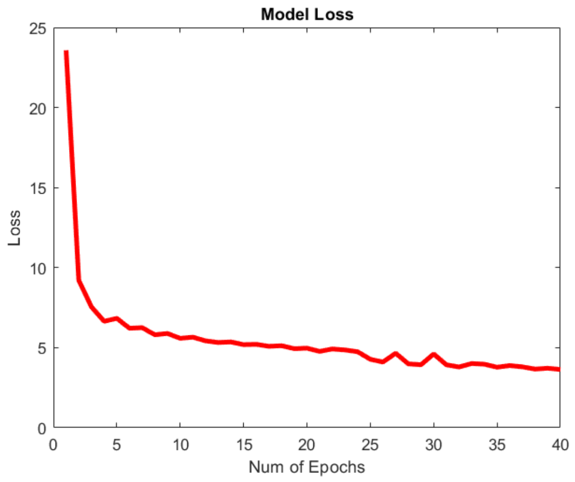

Fig. 10. The loss of the machine learning model over 40 epochs

This figure portrays the reduction in loss with increasing number of epochs. After the first 10 epochs, the rate of loss decreases dramatically, indicating that the model could likely perform successful identification and classification at this benchmark. Loss is the summation of errors in the model as it trains and is validated. From 10 epochs to 40 epochs loss continues to decrease, with small oscillations at various points. Computational time and processing power limits any exploration of higher epoch counts. The relative stability of the graph indicates that the loss will have little decrease over more epochs, and thus, there is a possibility that the data is becoming overfitted. Instead of classifying the data, the model is using its 'memory' to solve the ingredients in the training stage.

The mean average precision of the model indicates that the model reaches near 100% precision over the 40 epochs. Likewise, at 10 epochs, the system has achieved near full

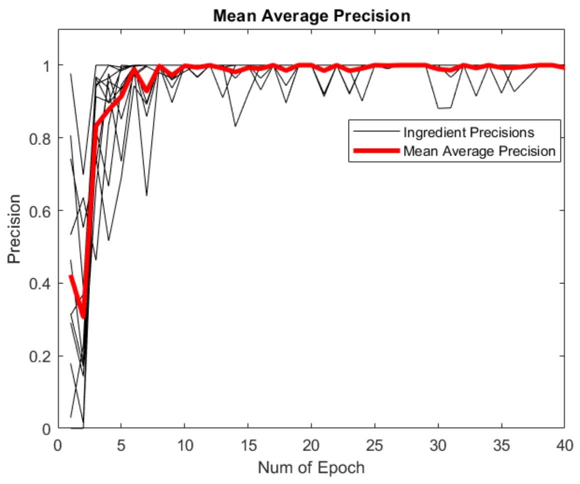

Fig. 11. Mean Average Precision of the model over 40 epochs. The black lines represent the precision of each of the 11 classes.

precision to the training data provided. Each ingredient (labelled in black) has improved in their respective precision, each class being 100% precise at 40 epochs. Small spikes indicate that certain ingredients become less precise at certain points of the training process. The final model depicted at 40 epochs did not contain any precision spikes. Although this aligns with the goals to successfully identify the classes in the dataset, overfitting is likely present. Without enough variety and content in the dataset images, the machine learning model has been easily trained, and will potentially struggle on unique environments or new scenarios.

During integration testing, ambient lighting and environment disturbances caused significant decreases in precision to the model. Lighting decreases precision of items to around 70%, whilst environment disturbances such as having foreign objects, such as the UR5e arm in the image caused significant errors in classification and incorrect bounding boxes around items. By reducing variations in lighting by blocking direct sunlight around the UR5e arm, the precision was increased to 95%, allowing the algorithm to successfully detect each classified item. Precautions such as ensuring the UR5e arm was raised above the camera range and moving utensils and bowls to new positions, minimised classification errors. The camera placement also had significant effects on accuracy. By observation, the machine learning model was accurate at distances less than 1.5m before accuracy began to drastically decrease. At a distance of 2m, the model consistently failed. This was expected as the camera was intended to be positioned close to the table, and thus, the training data was structured to match this design choice.

B. Depth Sensor

The results of the depth sensor are depicted in Figure 12. As can be observed in Figure 12, the measured values in both x and y do not accurately follow the true distances. As x covers a larger range of distances, the measured values are less accurate, with a maximum of 15cm difference. The

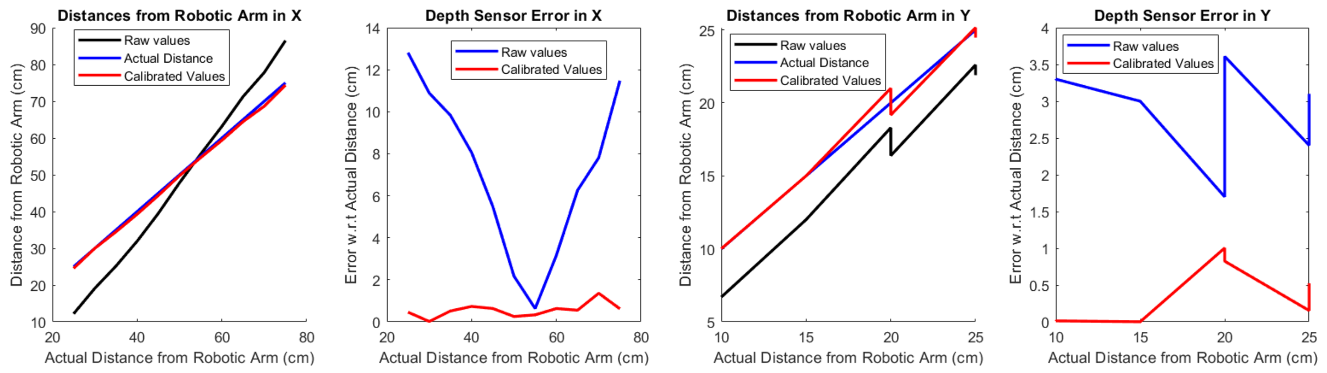

Fig. 12. Distance from the robotic arm w.r.t to X and Y and error between the measured/calibrated distance to the actual distance in X and Y

UR5e arm requires accurate positioning, with less than a 2cm threshold due to the gripper configuration.

The calibrated x distance from the robotic arm follows the true distance with high accuracy. The error between the calibrated distance to the actual distance is less than 2cm at all times. This value increases as the true distance increases; however, due to the size of the table, this trend has little importance. The measured values are highly inaccurate, with an error value of up to 13cm, indicating a strong requirement for calibration of the x coordinates.

The calibrated y distance from the robotic arm follows the true distance with improved accuracy. The y values cover a small range, due to the ingredient placement to allow ample room for the UR5e arm to move. Thus, any change is noticeable. A strong threshold of less than 1.5cm is required to ensure the arm does not push the item off the table, or incorrectly knock over the item. The error graph indicates that the calibrated values are less than this threshold. As the actual distance from the arm increases, this threshold spikes. This is potentially due to the depth sensor reading, however the calibration fit ensures an appropriate y coordinate can still be determined. The measured values are approximately always 3cm away from their desired actual values. The error graph is sporadic, indicating y without calibration is unpredictable and likely to cause the integrated system to fail.

During integration testing, the depth sensor provided accurate positions of the ingredients in each test run. Thus, the x and y calibration was successful in transforming the raw data to coordinates in the robot base frame.

C. Weight Sensor

During modularised testing, it was observed that the constant gripper tilt rate yielded a large overshoot in measurements, as high as 102% error. Thus, the angle was rapidly increased up to 50° before the angle increment was reduced from 5° to 3°. This yielded smaller measurement error exhibited by the overshoot of 34g during the demonstration indicating further control is required to tune the weight sensor.

D. System Functionality

Given the individual success of the machine learning and depth analysis modules, the overarching measure of system functionality was in actions successfully performed by the robotic arm.

During demonstration, a simple recipe was loaded such that a single sequence of actions was executed, indicating grasping of sugar as the first action. The machine learning module successfully identified the sugar packet from the array of available ingredients. The calibrated depth module then successfully determined the coordinate of the ingredient, utilising built in transforms to yield coordinates with respect to the base of the robotic arm. Given the high accuracy of the movements of the arm, the gripper end effector followed a Cartesian path to the given ingredient location, closing the gripper and raising the ingredient clear of the other surrounding ingredients. The next action extracted from the recipe instructs the system to measure 50g of sugar. Having grasped the ingredient, the arm moved to the fixed location above the measuring bowl sitting atop the scales. The weighing function then utilised feedback to tilt the gripper joint to pour the required amount, overshooting slightly to measure 84g. As more sugar was poured, the force on the gripper is maintained such that the grasp tightens as the packet reduces in size. Once the desired amount of sugar has been poured into the measuring bowl, the sugar packet is returned to the original location, following a Cartesian path that ensured the avoidance of surrounding obstacles. The next instruction saw the grasping of the measuring bowl containing the sugar and tilting the wrist joint such that the sugar is poured into the central mixing bowl. Placing the measuring bowl back on the scales for the next ingredient, the recipe then instructed mixing. As a component of mixing, the arm moved to the spoon, held vertical for ease of grasping, and moved to the mixing bowl. Taking the *'mix amount'* as a parameter, the spoon was rotated to exhibit a mixing motion. Finally, the spoon is returned to it's original location and the arm is sent to a central location ready to perform the next sequence of actions. This simple demonstration of the interpretation of a sequence of actions, collision-free path planning and successful action execution indicate the overall functionality of the system. Despite demonstration with just one ingredient, a more complicated recipe would only require further concatenation of the sequences performed during the demonstration.

V. Discussion

A. Machine Learning Module

The machine learning module was accurate in testing, regularly predicting the correct item. An obstacle faced was the environmental variations in lighting and foreign obstacles in the background. Another problem experienced was the extensive computation time of the model.

To optimise the model, the machine learning algorithm will need to be trained on data at different times of the day. Additionally, obtaining more data of the cooking environment, such as having images with the UR5e arm in the frame, would eliminate the incorrect identification and bounding box locations. Variance will allow the model to be more generalised and work in different environments and scenarios. The risk of overfitting will be reduced by significantly by increasing the amount of images in the dataset.

Computation time limited the potential minimisation of loss in the model. Using the computer graphics of a standard *Intel Core i7* processor produced the efficient 10 epoch model after 10 hrs. Each epoch was processed in approximately 3600 seconds. As expressed in results, this loss value is decreased as the number of epochs increase. Thus, to provide more computational power, the model was trained on Google Colaboratory. This provides a fast and efficient online GPU, suitable for creating machine learning models. The optimal 40 epoch model was trained in 12hrs, reducing the processing time of each epoch to 1116s. By increasing the computation power, an accurate model was created, with both increased precision and decreased overall loss.

The required input image size to the machine learning model was matched with the depth sensor image. As training data was collected with a phone before data was collected with a depth sensor, these sets had to be scaled and cropped appropriately such that they were consistent. Photos were initially taken in portrait mode, and thus to collaborate the two data sets, both required cropping. In future projects, the optimal input image size will be determined prior to the collection and labelling of data.

B. Depth Sensor

The depth sensor was accurate in testing, successfully providing coordinates of each ingredient in the scenario. The static position of the camera was recorded using a tape measure, often leading to inaccuracies in the robot frame position. Additionally, the tripod stand of the camera was occasionally at a tilt, rectified by utilising a tripod with a level to ensure the camera was not rotated about any axis. Due to environmental constraints, the camera was positioned at different ends of the table in regards to the robotic arm. This has the potential to change the yaw rotation expressed in Equation 12.

Due to the table size and UR5e arm configuration, the z coordinate could not be appropriately tested. Smaller z values caused the arm to malfunction due to detecting a potential collision with the table. Small ingredients such as vanilla extract and salt were avoided in testing for simplicity. Larger ingredients were unaffected by this issue due to little variance in their z values. Thus, to allow appropriate testing of the z coordinate using the depth sensor it will need to be calibrated to all heights and require successful motion planning by the UR5e arm.

C. Weight Sensor

As exhibited by the inaccuracies in measurements, further development of the weighing function is necessary. Firstly, the implementation of a Proportional Integral Derivative (PID) controller would yield more precise feedback that could be tuned during testing. Secondly, consideration to the viscosity of materials would yield more appropriate motion in relation to the ingredient. For example, liquids such as oil and water would require an extremely gradual tilt increment whereas ingredients such as sugar and cocoa require a certain tilt to begin the flow from the packet. As such, the weigh function could incorporate the ingredient as a parameter, adjusting the weighing motion accordingly. Additionally, a fast side to side motion could be implemented to ensure the gradual flow of ingredients such as flour that have a tendency to form clumps within the packet. The outcome of these measures is likely to improve the accuracy of the weighing function such that the correct ingredient amount is measured.

D. Demonstration Environment

Demonstration took place in the University of Sydney Halliday Laboratory, the location of the UR5e robotic arm. Given the environment, certain implementation components could not be physically verified. The use of an oven was not possible and as such, the actions of setting temperature, opening the oven door, placing the cake tin in the oven, closing the oven door and retrieving the cake at the conclusion of baking were not tested in practice. Additionally, given the experimental nature of the motion and the rules about food and drink within the laboratory, liquid ingredients were not opened during testing. These ingredients were still successfully identified and located by the machine learning and depth modules.

E. Hardware Limitations

During the process of configuration of the robotic arm, the UR5 was originally programmed. Complications arose due to the gripper configuration, lack of alignment of the robot axes with the required frames and abnormal, unpredictable height offsets. Given the variety of complications, the code was adjusted for use of the UR5e robotic arm. The gripper was adjusted from the RG2 gripper being utilised on the UR5e arm to the Robotiq 2F gripper. The general operation was confirmed, yielding a more reliable system where further functionality could be safely implemented.

Melted butter and cracked eggs are components of majority of baking recipes. In the given time frame, the implementation of functions to add each of these ingredients was not realised. The cracking of an egg was not feasible given the 2 finger gripper implemented on the robotic arm. A device such as an egg cracker was considered, operated by placing the egg into the device and closing the gripper on the handles. The device was not obtained due to limitations in availability online and delayed shipping. Recipes that used butter required the butter to be melted. The deep learning module was able to successfully identify butter unwrapped in the standard block form. Unlike the packet ingredients, butter needed to be cut to the required amount and melted. This limitation may require a user to cut the required amount of butter and place it in the workspace at the beginning of the cooking process. A hotplate could be utilised to melt the given amount of butter which could then be added to the mixing bowl and handled like any other ingredient. Given the limitation in access to a hot plate, this functionality was not yet realised. The hub structure is designed such that these functions are easily integrated into the existing code framework once implemented.

Another obstacle was faced as the number of USB computer ports was insufficient for the robotic arm, the gripper and the weight sensor Arduino. Despite the attempt to use an extension hub, for full functionality, control had to be manually swapped from the gripper to the Arduino at the time of weighing. This also involved the configuration of the USB port each time. Acknowledging this shortcoming as a hardware limitation, such a consideration will be managed by utilising a computer with sufficient USB ports for future operation.

VI. Conclusion and Future Works

In this paper, a method of integrating deep learning into an autonomous cooking system was developed and tested. The overall goal was to create a baking robot that verified technologies and processes of cooking with high precision and accuracy in a dynamic environment. The use of the Intel RealSense RGBD camera allowed for capturing of images and depth data for accurate object and position identification. In conjunction with the UR5e robotic arm and an Arduino driven weight sensor, specific cooking actions were consistently performed.

The deep learning model was able to successfully detect and classify the 11 ingredients in a stable environment. Testing of the depth sensor showed a consistent calibrated accuracy in the x and y directions respectively, allowing the UR5e arm to successfully manoeuvre ingredients to appropriate locations. The fine tuned weight sensor required control methods to assist with providing accurate readings.

Overall, the designed system exemplifies the viability of implementing deep learning as a foundational component of a cooking robot, allowing for a highly autonomous system that can adapt to a dynamic arrangement of ingredients. It is evident that the employment of such techniques addresses a lacking component of several existing products on the market. To extend upon the basic implementation developed, several areas for the future direction of the project are proposed.

In extension to the proposed project, implementation with multiple robotic arms or more agile gripper configurations would allow for greater overall agility, thus increasing the range of available actions. Incorporating further sensors would allow for a more accurate interpretation of the surrounding environment. One such extension could be the placement of an additional camera mounted on the robotic arm for further visibility and controlled adjustment during precise movements. Inspired by the technology of Moley Robotics, the system could extend the basic object detection deep learning module to incorporate further machine learning applications such as action recognition and repetition. Furthermore, identification of the utensils and bowls required would enable a more dynamic system that did not require fixed equipment positions. With the overarching goal of widespread system adoption, future works should be centred on bringing further functionality such as inventory tracking and cleaning for a comprehensive, autonomous cooking assistant.

References

[1] "Moley robotics," 2020. [Online]. Available: https://www.moley.com/

[2] "Connected robotics: Octochef." [Online]. Available: https://connected-robotics.com/en/product/octochef.html

[3] K. Hao, "What is machine learning?" [Online]. Available: https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/

[4] "Abundant robotics." [Online]. Available: https://www.abundantrobotics.com/

[5] M. T. Islam, B. M. N. Karim Siddique, S. Rahman, and T. Jabid, "Image recognition with deep learning," in *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, vol. 3, 2018, pp. 106–110.

[6] "Calorie mama," 2020. [Online]. Available: https://www.caloriemama.ai/

[7] "June smart oven," 2020. [Online]. Available: https://juneoven.com/

[8] Ma, Liu, Ren, and Y.-F. Luo, "Detection of collapsed buildings in post-earthquake remote sensing images based on the improved yolov3," *Remote Sensing*, vol. 12, p. 44, 12 2019.

[9] "tzutalin/labelimg," 2020. [Online]. Available: https://github.com/tzutalin/labelImg

[10] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *ArXiv*, vol. abs/1804.02767, 2018.

[11] J. Hui, "Real-time object detection with yolo, yolov2 and now yolov3."

[12] "Universal robot ur5e," 2020. [Online]. Available: https://www.universal-robots.com/products/ur5-robot/