ASHESI

ASHESI UNIVERSITY

DESIGN OF A LOW-COST ROBOTIC MANIPULATOR FOR FOOD HANDLING

CAPSTONE PROJECT

B.Sc. Computer Engineering

Larkuo Wilson-Tetteh

2022

ASHESI UNIVERSITY

DESIGN OF A LOW-COST ROBOTIC MANIPULATOR FOR FOOD HANDLING

CAPSTONE PROJECT

Capstone Project submitted to the Department of Engineering, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Engineering.

Larkuo Wilson-Tetteh

2022

DECLARATION

I hereby declare that this capstone is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

Candidate's Name:

Larkuo Wilson-Tetteh

Date:

28/04/2022

I hereby declare that preparation and presentation of this capstone were supervised in accordance with the guidelines on supervision of capstone laid down by Ashesi University College.

Supervisor's Signature:

……………………………………………………………………………………………

Supervisor's Name:

Dr. Stephen K. Armah

Date:

……………………………………………………………………………………………

Acknowledgements

To my supervisor, Dr. Stephen K. Armah whose encouragement and academic advice helped me undertake this project.

Abstract

Industry 4.0 improved production processes for many manufacturing processes however, this change was not as pronounced in many service industries including the food service industry. Considering the high cost of implementing technologies such as robotics, edge computing and machine learning there are not many options for establishments such as restaurants and fast-food places to also implement various parts of Industry 4.0 into their services. With the growing consumer demand especially regarding dietary differences, it has become more important for food service establishments to make their processes smarter to keep up with the demand.

In this project, a low-cost, computer vision-controlled robotic manipulator is implemented to assemble a custom plate of food without human assistance. This was achieved by collecting and annotating 1000 images as well as training an object detection model with no less than 90% average precision for each of the possible classes as well as an average recall of 94.45% where there were no more than 10 objects in the image frame. A 3D printed robotic manipulator was then assembled with servo motors, a Raspberry Pi, and a low-cost web camera and the model, together with python scripts were then used to make real-time detections while picking and dropping the food items from one plate to another within the manipulator's workspace.

Table of Contents

DECLARATION i

Acknowledgements ii

Abstract iii

Table of Contents iv

Chapter 1: Introduction 6

1.1 Background 6

1.2 Problem Definition 7

1.3 Proposed Solution 7

1.4 Objectives 7

Chapter 2: Literature Review & Related Work 9

2.1 Literature Review 9

2.2 Related Work & Gaps 11

Chapter 3: Design 16

3.1 Requirements 16

3.2 Material Selection 16

3.3 CAD Model 20

3.4 Electrical Circuit 21

Chapter 4: Data Collection & Model Training 23

4.1 Image Collection & Labeling 23

4.3 Model Training 26

Chapter 5: Model Results & Prototyping 27

5.1 Overview 27

5.2 Model Results 28

5.3 Assembling the Manipulator 29

5.4 Real-time detections & Manipulator movements 30

Chapter 6: Conclusion & Future Works 31

6.1 Conclusion 31
6.2 Future Works 31
References 32
Appendix 33
7.1 Python Scripts 33
7.2 Training Notebook 39
7.3 Images and Annotation Files 39
7.4 CAD Model 40

Chapter 1: Introduction

The food industry has not seen much adaptation of Industry 4.0 in its operations. With the ever-increasing demand and variance in dietary preferences, it has become more important that food service establishments begin to explore options to integrate smart technologies into their kitchens. In this chapter of the project, a detailed background will be discussed followed by a problem definition and a proposed solution. Finally, the objectives of the project will be highlighted.

1.1 Background

Industry is an essential part of the global economy that produces goods through mechanized and often automated processes. Industrialization has experienced multiple technological cycles that enhanced processes throughout history with the most recent being Industry 4.0. This cycle of the industrial revolution is changing the way industries manufacture their products with the introduction of Internet of Things (IoT), Edge computing, Robotics, Artificial Intelligence (AI) and machine learning into their manufacturing processes [1]. Essentially, Industry 4.0 describes the advent of smart factories which have predictive maintenance, quicker responses, and more autonomously intelligent manufacturing procedures.

While many industries have integrated many parts of Industry 4.0 into their operations, the food production industry especially in customer-facing establishments has not seen much change with regards to making their processes smarter. Commercial kitchens are one of such customer-facing establishments and form a significant part of the industry. In commercial kitchens, large quantities of are cooked and served in real-time daily and such kitchens are typically found in places such as hotel restaurants, cafeterias, and fast-food establishments. Like many other industries, commercial kitchens are designed to produce food quickly and repetitively without jeopardizing food quality and customer satisfaction.

1.2 Problem Definition

Commercial kitchens do a great job at serving multiple customers, these services involve manned activities which are prone to error and inefficiency especially when exhaustion sets in. Considering the cost of implementing traditional industrial automation and the growing human population as well as an increasing variance in the dietary preferences of food consumers [2] it has become more complex to use traditional industrial technologies to automate processes in commercial kitchens unlike with other industries whose processes do not involve as many customizations.

1.3 Proposed Solution

By means of smart, low-cost, and versatile technology commercial kitchens can begin to implement parts of Industry 4.0 in their establishments to help them keep up with increasing demand and ever-changing custom dietary requirements of the customers they serve. For this project, the proposed solution would be to design and build an affordable robotic manipulator that is smart and can identify different types of food items as well as assemble customizable plates for customers.

1.4 Objectives

1.41 Main Objective

With the increasing demand placed on commercial kitchens and the need to keep up with competitors, the aim of this project is to design and prototype an autonomous low-cost robot that can assemble a meal with minimal human input. To reduce the need for human interference or supervision, another important part of this project is to make the robot smart.

1.4.2 Specific Objectives

After establishing the primary objective of the project, which is to design a smart, low-cost robot to assemble a meal, some more specific objectives of this project are mentioned subsequently.

1. Design a robotic arm that can pick and drop cooked food without significantly altering the organoleptic properties of the food.
2. Design a robotic arm whose workspace is suitable for a kitchen workstation.
3. Train an object detection model that can correctly identify different food items and kitchen utensils as well as their XYZ-positions in the workspace in real-time.

Chapter 2: Literature Review & Related Work

An important part of the engineering design process is reviewing existing literature and projects to draw inspiration and identify gaps. In this chapter of the project, manipulator dynamics will be studied along with important concepts in computer vision and object detection such as image labelling and model evaluation metrics. Subsequently, similar existing projects and papers that have been published will be mentioned to draw some inspiration for the solution and identify some gaps.

2.1 Literature Review

2.1.1 Manipulator Dynamics

A manipulator's kinematics can be described in two ways, forward kinematics, and inverse kinematics. Forward kinematics are used to describe the position and orientation of the robot's end effector based on the angular displacement or velocity inputs from each joint of the robot. Whereas inverse kinematics are used to calculate the value each joint variable given the specific position and orientation of the end effector [3]. The Denavit-Hartenberg (D-H) method [4] is a standard method used to derive the kinematics for a robotic manipulator for all robot configurations and will be used in this paper.

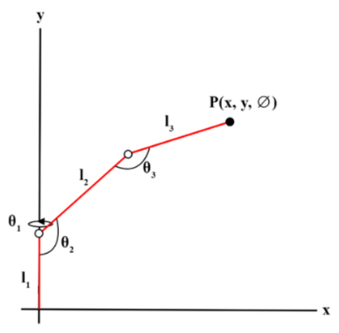

Figure 2.1: Labeled diagram of joint angles, link lengths, and end effector positions for a robotic manipulator with 3 degrees of freedom.

From Figure 2.1,

$$P\ (x, y, \emptyset) = x, y\ position\ of\ end\ effector\ with\ orentation\ \emptyset$$

$$l_i = length\ of\ link\ i$$

$$\theta_i = angle\ of\ rotation\ \theta\ of\ link\ i$$

And, given a robot with 3 degrees of freedom (3-dof), like in figure 2.1, the forward kinematics are,

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) - - - (1)$$

$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) - - - (2)$$

$$\emptyset = \theta_1 + \theta_2 + \theta_3 - - - (3)$$

While the inverse kinematics are,

$$\theta_1 = \tan^{-1}\frac{y}{x} - - - (4)$$

$$\theta_2 = \cos^{-1}\frac{x^2 + y^2 + z^2 - l_1^2\, l_2^2}{2 l_1 l_2} - - - (5)$$

$$\theta_3 = \sin^{-1}\frac{\emptyset}{\sqrt{x^2 + y^2 + \emptyset^2}} + \tan^{-1}\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2} - - - (6)$$

2.1.2 Object Detection & Position Calculation

Object detection/recognition is a type of computer vision task where instances of objects in an image or video frame are located [5] with details such as a bounding box, scale of the object in the frame and the score of the prediction which describes how sure the model is about the prediction. Algorithms that perform object detection tasks are often a result of using machine learning to train the algorithm to detect the desired objects.

Measuring the performance of an object detection model is important before the model is deployed into an application. Some performance metrics used to evaluate object detection models include, Intersection over Union (IoU), Precision and Recall.

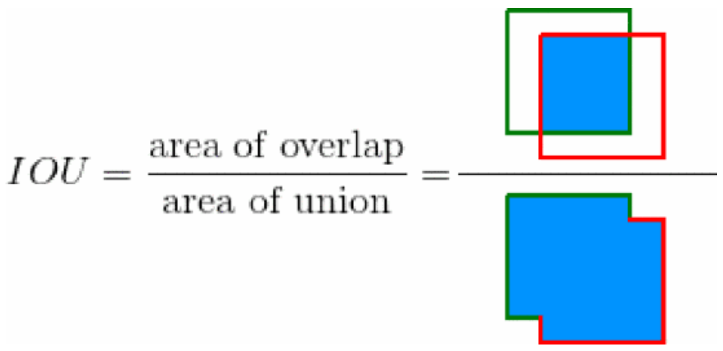

Figure 2.2: A graphical representation of the Intersection over Union (IoU) metric [6].

IoU is a metric that "measures the overlapping area between the predicted bounding box and the ground-truth/actual bounding box divided the area of the union between them" as [6] as demonstrated in Figure 2.2. Precision describes a model's ability to only make correct predictions. These include true positives and true negatives; true positives refer to correct bound box detections where there are identifiable objects in the image/video frame while true negatives refer to the absence of bound boxes where there are no identifiable objects in frame. The metric recall assesses the model's ability to find all existing bound boxes. [6].

2.2 Related Work & Gaps

2.2.1 Interpreting and Executing Recipes with a Cooking Robot (Bollini, Tellex, Thompson, Roy & Rus)

In their paper "Interpreting and Executing Recipes with a Cooking Robot" Bollini, Tellex, Thompson, Roy and Rus implemented a robot that reads plain text recipes for simple dishes and executes them; one recipe at a time [7]. With the assumption that the robot had access to a model of the kitchen, a set of labeled ingredients and the locations of tools such as a whisk or an oven the robot was initialized with a set of ingredients in labeled bowls laid out on a table. It was also initialized with a set of natural language instructions describing how to use the ingredients laid out to cook a specified dish. For instance, a group of ingredients which include flour, butter, sugar, and eggs may be initialized together with instructions such as “Add sugar and beat to a cream”. The robot also has a repertoire of basic actions which include pouring and mixing ingredients. After parsing the plain text recipe to the robot, it performs the inferred actions associated to the instruction using the PR2 Robot Manipulation Platform.

The PR2 Robot Manipulation Platform is a robot that was built by researchers at Willow Garage and first released in February 2009 and made available for purchase in September the next year; 2010 captured in Figure 2.3. It has two manipulators and runs on ROS (Robot Operating System) which is an open-source robotics framework that allows communications, message passing, remote procedure calls, among many other components [8].

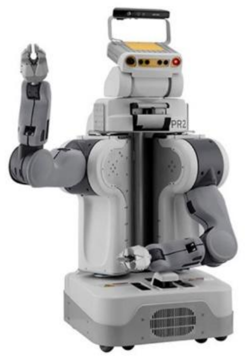

Figure 2.3: The Willow Garage PR2 Personal Robot 2 mobile manipulation platform used for the BakeBot system [8].

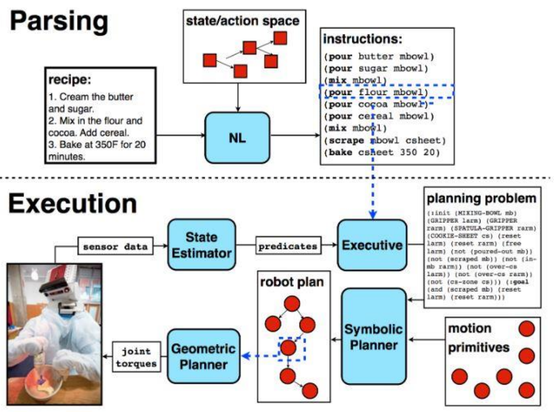
Parsing
state/action space
recipe:
1. Cream the butter and sugar.
2. Mix in the flour and cocoa. Add cereal.
3. Bake at 350F for 20 minutes.
NL
instructions:
(pour butter mbowl)
(pour sugar mbowl)
(mix mbowl)
(pour flour mbowl)
(pour cocoa mbowl)
(pour cereal mbowl)
(mix mbowl)
(scrape mbowl csheet)
(bake csheet 350 20)
Execution
sensor data
State Estimator
predicates
Executive
planning problem
(:init (MIXING-BOWL mb) (GRIPPER larm) (GRIPPER rarm) (SPATULA-GRIPPER rarm) (COOKIE-SHEET cs) (reset larm) (reset rarm) (free larm) (not (poured-out mb)) (not (scraped mb)) (not (in-mb rarm)) (not (over-cs larm)) (not (over-cs rarm)) (not (cs-zone cs))) (:goal (and (scraped mb) (reset larm) (reset rarm)))
robot plan
Symbolic Planner
motion primitives
joint torques
Geometric Planner

Figure 2.4: Architecture of the Bollini et al. BakeBot System that shows the progression from NL processing of the plain text recipe to executing the task [7].

A graphical representation of the architecture of the cooking robot system by Bollini et al. is shown in Figure 2.4.

To test their approach, the researchers experimented with their language processing system and a database of 60 recipes to test instruction inference and a real-world demonstration in a kitchen which had two work surfaces to test execution. After doing the real-world experimentation 27 times, the researchers found that their defined basic motions enabled a variety of recipes to be executed. They also discovered that their natural language system worked well because for most instructions, it correctly inferred the corresponding action. However, they also recorded an average runtime of 142 minutes from start to finish which was far slower than the average human. In their tests, they also noted that the robot made many more failures such as the end effector slipping of the oven door halfway through the opening procedures.

An identifiable gap in Bollini et al.'s research paper is that the design focused heavily on natural language processing. This resulted in the partial neglect of the design for the hardware especially the end effector of the robot manipulators which resulted in mistakes. The design and experimentation did not consider the different textures, hardness and sizes of the tools and ingredients that were used for the demonstration which affected the end effector's performance. Finally, the experiment did not mention how the robot adhered to kitchen hygiene standards. Thus, the safety of the food for human consumption is not assessed.

2.2.2 A Methodology for the Selection for Industrial Robots in Food Handling. (Farah Bader & Shahin Rahimifard)

Another paper that explored the robot selection process for the food industry is "A methodology for the selection of industrial robots in food handling" by Bader and Rahimifard [2]. The aim of this paper's research was to discover a design methodology that makes it easier for engineers and producers in the food industry to design flexible robots and suitable end effectors that can handle food appropriately based on the foodstuff in the application. The researchers also aimed at ensuring that the design process ensured that the organoleptic properties of all foodstuffs handled by the selected robot was preserved. Bader and Rahinmifard named their four-step process of selecting the appropriate robot for food handling Food Industrial Robot Methodology (FIRM).

Step one of FIRM involves defining foodstuff by type and condition as well as the state of the foodstuff under consideration.

- Foodstuff Type: Meat & Poultry, Seafoods, Fruits & Vegetables, Baked Goods & Confectionaries, Dairy Derivatives or Others (e.g., grain like rice and powdered spices)
- Foodstuff Condition: Raw, Cooked or Frozen
- Foodstuff State: Whole or Segmented

Step two of the selection method further classifies food items by rigidity how hard or soft the foodstuff is and deformability how easy it is to change visual and textural properties the foodstuff when force is applied by the robot. In both classifications, foodstuff can be described as follows.

- Deformability: Deformable or Non-Deformable
- Rigidity: Rigid, Semi-Rigid or Non-Rigid

These two classifications of foodstuff in this step creates six distinct groups which made selection of the type of robot and suitable end effector easier.

Overall, this project excelled at creating a standard methodology for engineers who in the future will work on industrial robots for food. However, a gap of this paper was that a limited group of options were selected as options for handling the food. Again, though an example of the FIRM procedure was demonstrated in the paper, the was no documented test to give an idea of how effective this method is in real world food handling applications.

Chapter 3: Design

In this chapter of the project, the primary of objective will be to design an effective food handling robotic manipulator and end effector that correctly identifies, locates, and picks up food items to assemble a meal. For proof of concept, this project will focus on assembling a plate of potato fries with a piece or more of chicken as well as a portion of ketchup where requested.

3.1 Requirements

To achieve the objectives of this project, there are some functional and technical requirements the manipulator needs to satisfy.

Functional Requirements:

1. A camera to enable computer vision-based object detection.
2. At least 3 degrees of freedom (3-dof) in the robotic arm to enable easy movement in the workspace.

Technical Requirements:

3. An object detection model with at least 90% Average precision (AP) for each item that needs to be detected.
4. An object detection model with at least 90% Average recall when there is a maximum of 10 objects in the image/video frame.

3.2 Material Selection

In this section of the design chapter, materials including motors, a microcontroller and a camera will be selected using weighted Pugh Matrixes which feature a datum as well as comparison options for each part of the proposed solution.

3.2.1 Motors

In considering motor options for the manipulator, some significant requirements included cost, feedback for closed loop control, power consumption, precision, torque, and accuracy. These requirements are also weighted in the table below because requirements such as cost, and feedback are more relevant to the project because the robotic manipulator needs to be low cost and allow feedback for controls. However, requirements such as torque are weighted less because all food items listed in the design overview weight less than 1 kilogram (kg) since they will each be handled one at a time. The datum for this matrix was a DC motor with the options being a servo motor and a stepper motor as shown in Table 3.1.

Table 3.1: Pugh matrix for the selection of motors with DC motor as datum

Criteria	**Weight**	**DC Motor**	**Servo Motor**	**Stepper Motor**
Low Cost	3	0	0	-
Feedback/Control	4	0	+	+
Power Consumption	2	0	0	-
Precision	3	0	+	+
Torque	4	0	+	+
Accuracy	4	0	+	+
Continuous Rotation	5	0	+	+
Total		**0**	**+20**	**+15**

In this selection, a servo motor was selected because like the datum, it is low cost however, unlike the DC motor, it allows feedback and control without the need for an external encoder. Finally, the servo motor had a higher score because it is accurate even at high speeds.

3.2.2 Manipulator Material

With the selection of the material most suitable for the manipulator, the criteria is focused on ease of cleaning, food safety, cost, heat resistance, durability, and humidity resistance. This is because in typical kitchens where a solution of this type will be implemented, there is high heat and humidity which are factors that cause corrosion and wear in some materials. The datum for this selection is stainless steel; a material often used for kitchen utensils and the options are aluminium, copper and polylactic acid (PLA).

Table 3.2: Pugh matrix for the selection of the manipulator material with stainless steel as datum

Criteria	**Weight**	**Stainless Steel**	**Copper**	**Aluminium**	**Polylactic Acid (PLA)**
Low Cost	3	0	-	+	+
Ease of cleaning	3	0	0	0	-
Food Safety	5	0	-	0	0
Heat Resistance	4	0	-	-	+
Durability	4	0	-	-	-
Humidity Resistance	4	0	-	-	+
Total		**0**	**-20**	**-14**	**+4**

From Table 3.2, the material with the highest score is polylactic acid (PLA) which was scored higher than the other materials because of its low cost and food safety. Again, it is less susceptible to corrosion in humid environments and does not conduct heat well which is important for this project.

3.2.3 Microcontroller

For the selection of a microcontroller for this project, processing power was a very important criteria because of the use of a camera in the design. This is because implementing vision algorithms especially in real-time is computation heavy. Again, cost, power consumption, and having enough GPIO pins were important to consider especially because of the motors required for this project.

Table 3.3: Pugh matrix for the selection of a microcontroller with Arduino Nano as datum

Criteria	**Weight**	**Arduino Nano**	**Raspberry Pi 4**	**ESP 32**	**Tetrix Prizm**
Low Cost	3	0	-	+	-
Processing Power	5	0	+	-	0
GPIO Access	5	0	+	-	+
Power Consumption	3	0	-	+	-
Total		**0**	**+4**	**-4**	**-1**

The Raspberry Pi 4 was selected in Table 3.1 because though it is slightly more expensive than the datum, this microcontroller has USB ports compatible with many cameras. The Raspberry Pi also has the best processing power compared to the datum and the other options. This makes it the most appropriate for a computation heavy application such as object detection. Again, the Raspberry Pi has enough pins to house all other components required for the robotic manipulator.

3.2.4 Camera

For the selection of an appropriate camera, resolution, cost, field of view imaging rate and suitable interconnect protocols were important criteria. Interconnect or interface standards for cameras can be described as a codified identity for how a camera is connected to a computer or micro-controller to allow easier and more effective use of vision as a technology. With this selection, the datum was the Raspberry Pi Camera v2 with the rest of the selection options being USB 2.0 camera as well as the Raspberry Pi NOIR camera. The criteria and the options are shown in the Table 3.4.

Table 3.4: Pugh matrix for the selection of a camera with Raspberry Pi Camera v2 as datum

Criteria	**Weight**	**Raspberry Pi Camera v2**	**USB 2.0 Camera**	**Raspberry Pi NOIR Camera**
Low Cost	3	0	+	0
Interface Standard	5	0	+	0
Power Consumption	3	0	-	0
Field of View	4	0	+	0
Resolution	4	0	0	0
Imaging Rate	4	0	+	0
Speed	5	0	+	0
Total		**0**	**+18**	**0**

Ultimately, the USB 2.0 camera was selected because it cost less, had a more common interface standard as well as the camera standard having a larger bandwidth and wider range for its field of view. Again, the selected camera is compatible with the selected microcontroller; Raspberry Pi 4.

3.3 CAD Model

To model the movements of the manipulator prior to building the manipulator, a 3D model was created using Onshape; a web-based computer aided design (CAD) software. The model is a robotic manipulator with 3 motors: one for each degree of freedom. See Figure 3.1 and Figure 3.2.

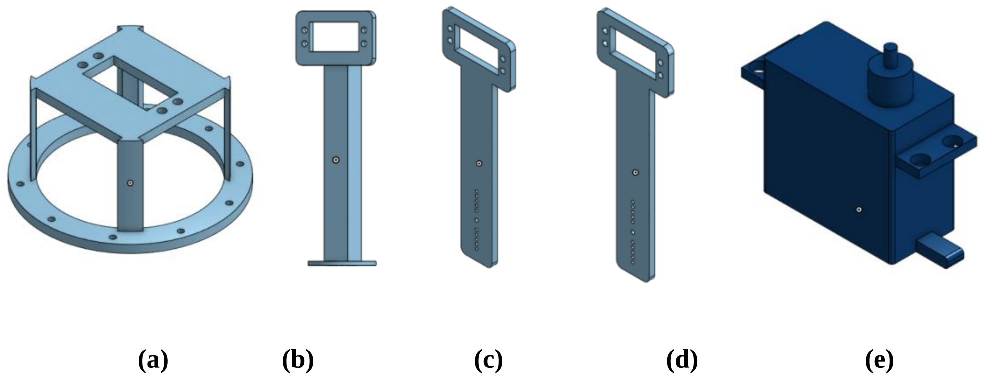

Figure 3.1: Diagram of parts for the 3D model of the manipulator (a) Link 1 (b) Link 2 (c) Link 3 (d) Base (e) Servo Motor

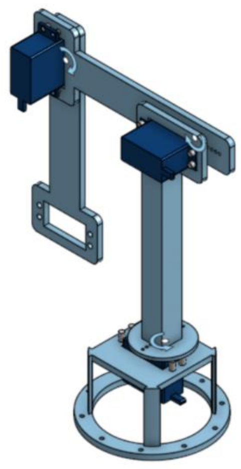

Figure 3.2: Image of manipulator assembly CAD model

3.4 Electrical Circuit

Figure 3.3 shows how the electrical and electronics parts of the robot manipulator will be connected. The connections include 4 servo motors, a Raspberry Pi, a USB camera plugged into the Raspberry Pi and 2 power supplies; one for the Raspberry Pi and another for the servo motors.

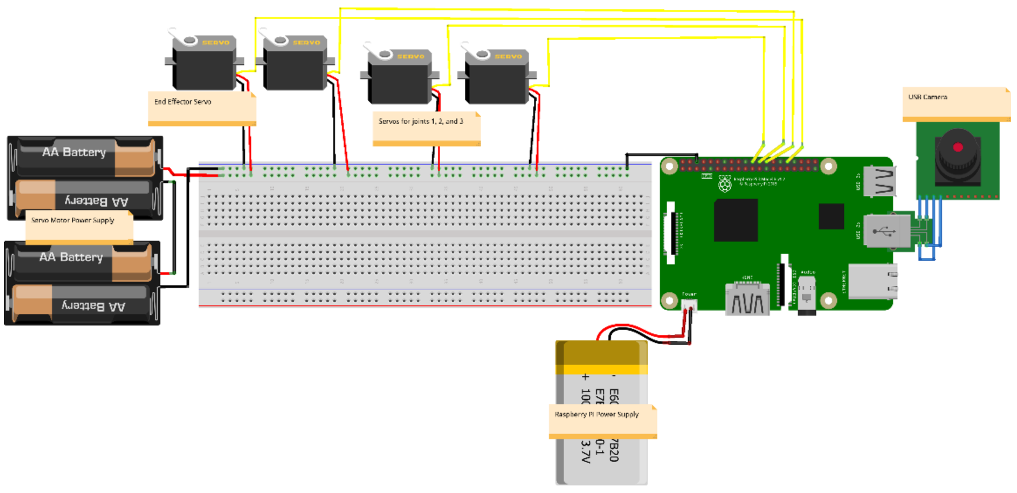

Figure 3.3: Circuit diagram of electronic connections for a 3 DOF manipulator in Fritzing

Chapter 4: Data Collection & Model Training

In this chapter of the project, details about collecting the images, processing the images, and labelling the images will be discussed. Setting up the training environment and other required details as well as the process of training the object detection model will be highlighted.

4.1 Image Collection & Labeling

4.1.1. Image Collection

One of the most important parts of training any machine learning model is the process of collecting the data. For this project, a USB camera was connected to a computer at a height, 30 cm from the kitchen worktable as shown in Figure 4.1.

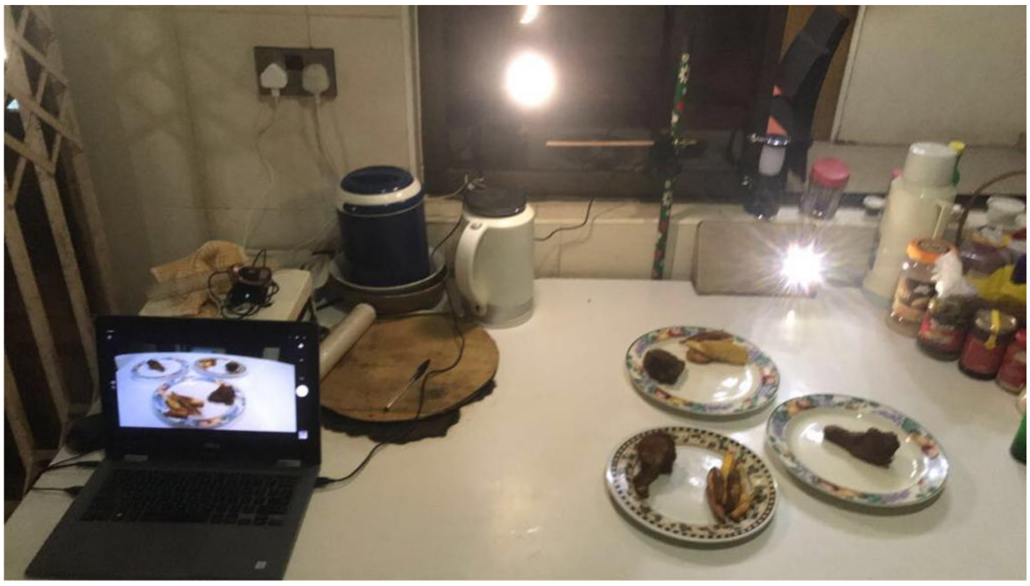

Figure 4.1: An image of the data collection setup with the laptop and camera

With the computer and camera setup, 1000 images were taken: each with a resolution of 640 pixels x 320 pixels. Images were taken in batches of 5 with variations which included changing the orientation of items, reducing the amount of light, and moving objects in and out of the image frame. Some samples of the images are shown in Figure 4.2.

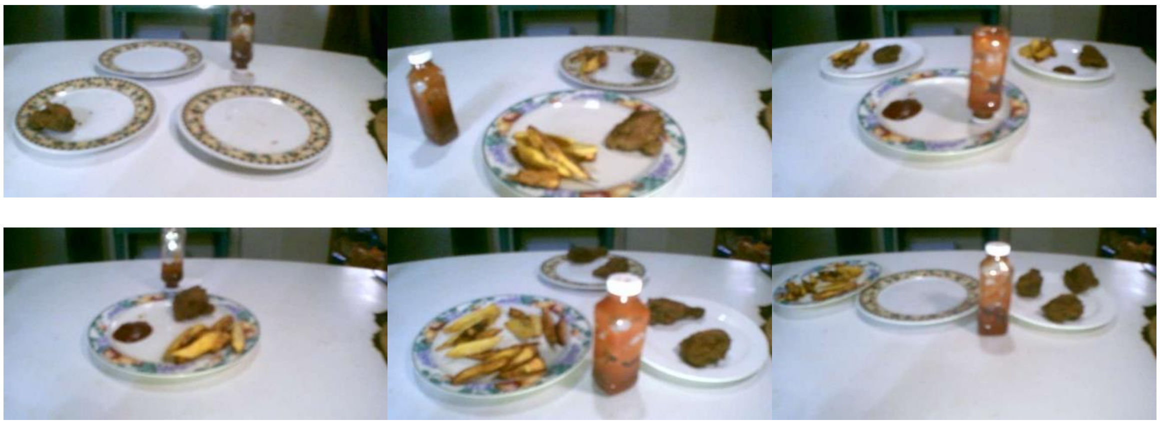

Figure 4.2: Sample images from data collection

4.1.2. Image Processing & Labeling

Following the collection of the images, a python script was written to rename the images with numbers in order to improve the coherence of the file naming and make it easier to associated related annotation files. Each file was named *img-<number>* where the number count began from 0 and ended at 999.

With a more coherent naming format for the images, the images were labelled with LabelImg: a python-based graphical tool that allows users to create bound boxes around objects in an image and save the boxes and labels as a preferred file format [10] depicted in Figure 4.3 below. Through the labelling process, bound boxes were drawn around each item in each image and labelled as one of the following five classes/labels: ketchup bottle, ketchup portion, potato fries, chicken, and empty plate.

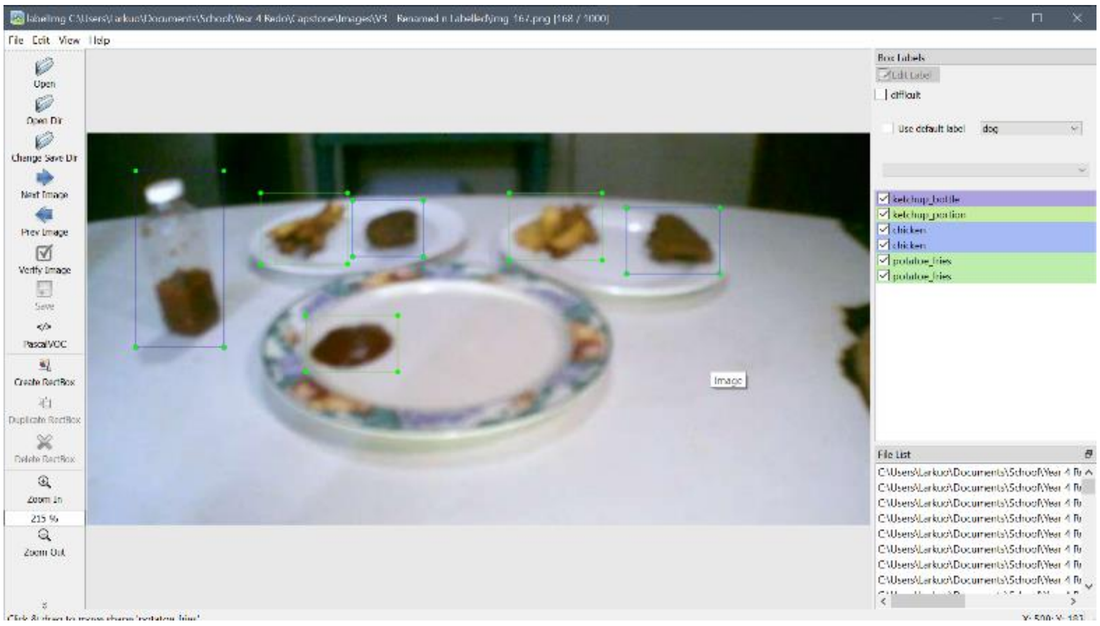

Figure 4.3 A screengrab of the LabelImg application while images were being labelled

Labeling all 1000 images resulted in a total of 4301 individual bound boxes with the average number of bound boxes per image being 4. In the bar chart below, the distribution of the classes for the bound boxes is shown.

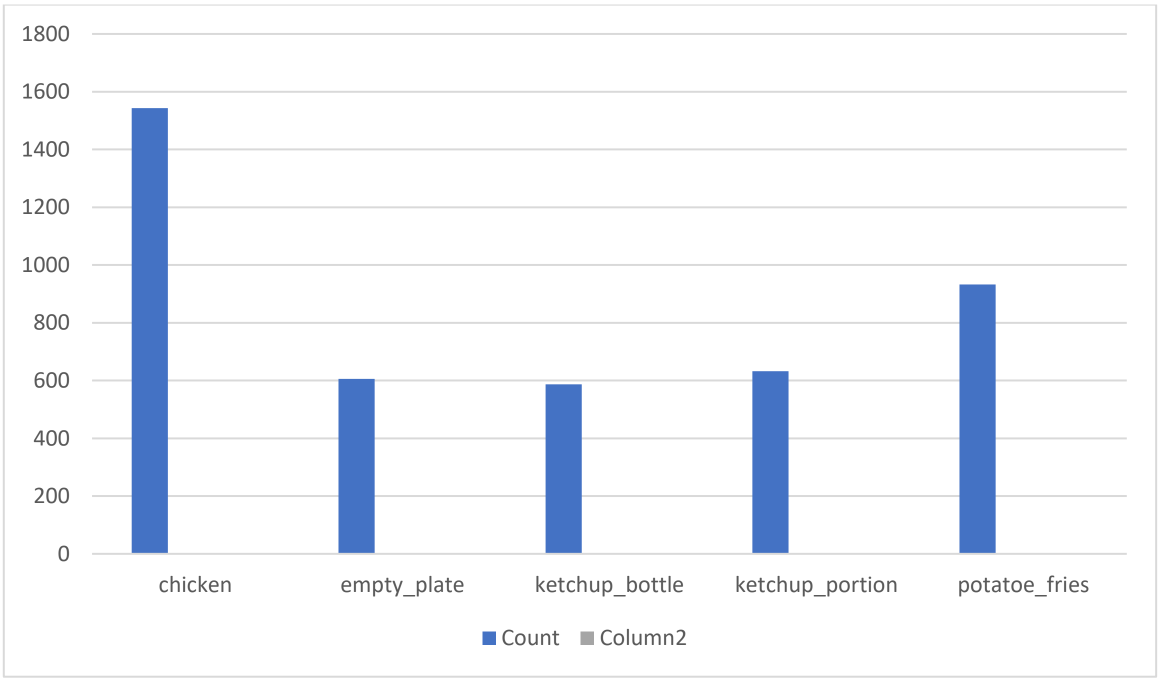

Figure 4.4: A bar chart of the count of the bound box classes

4.3 Model Training

Training the model for this project was done in Gradient; a web-based machine-learning and deep-learning resource by the company Paperspace that offers python notebooks as well as CPUs and GPUs for training machine learning models. Before training the model, a python script was used randomize the images and split the data into an 80:20 ratio for training and validating the object detection model.

Tensorflow is an open-source software library developed by Google researchers that allows users to build, train and evaluate machine learning models and applications [11]. Tensorflow Lite is a version of Tensorflow that allows machine learning applications to be deployed on devices with more limited memory and storage such as mobile devices, microcontrollers as well as other edge devices.

To train the model, the EfficientDet computer vision architecture was used. EfficientDet is an object detection architecture which uses multiple optimizations as well as a compound scaling method which allows a single model uniformly scale the resolution, depth and width of images regardless of the backbone [12]. In the Tensorflow Lite library, there are 5 architectures based on EfficentDet; EfficientDet-Lite0, EfficientDet-Lite1, EfficientDet-Lite2, EfficientDet-Lite3 and, EfficientDet-Lite4. Where, EfficientDet-Lite0 has the lowest latency (146 ms) and EfficientDet-Lite4 has the highest latency (1886 ms). For this project, Efficient-Lite0 was chosen to train the model because the model was going to be used to make predictions in real-time hence, minimal latency was important.

With all the training data and required libraries saved into the python notebook's workspace, the model was trained for 100 cycles (epochs) with a batch size of 10 per step of each epoch. At the end of 100 epochs the model was saved in the "*tflite*" format and evaluated to verify that it met the requirements.

Chapter 5: Model Results & Prototyping

In the model results and prototyping chapter of the project, an overview of the prototyping process will be outlined. The results of evaluating the object detection model will also be discussed together with the process of assembling the manipulator and using python scripts to make real-time object detections while using manipulator kinematics to move the robotic manipulator.

5.1 Overview

After training the model and evaluating it, the model was downloaded unto the Raspberry Pi and used to make predictions. The predictions were then used to pick and drop food items from one point of the kitchen workbench to another point. In this chapter, the results of the model's evaluation will be discussed alongside assembling the robot and using the model's predictions and python scripts to move the manipulator servos to pick and drop food items.

To using the manipulator prototype and the detection model, the sequence of events are shown in the flow chart; see Figure 5.1.

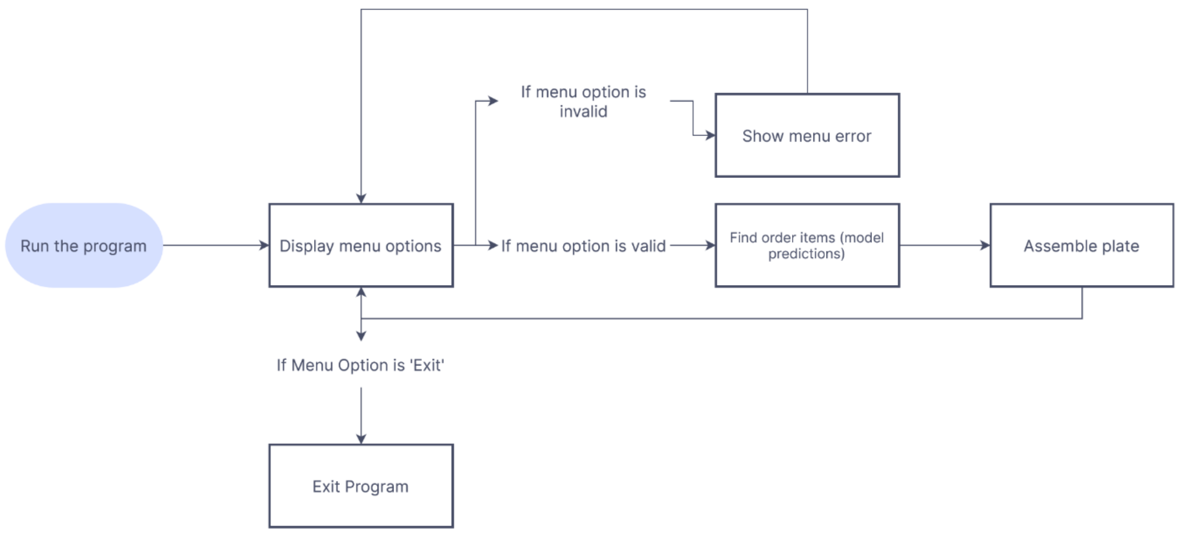

Figure 5.1: A flow chart demonstrating the sequence of events for prototyping

5.2 Model Results

An important part of the engineering design process is reviewing existing literature and projects to draw inspiration and identify gaps. In this chapter of the project, manipulator dynamics will be studied along with important concepts in computer vision and object detection such as image labelling and model evaluation metrics. Subsequently, similar existing projects and papers that have been published will be mentioned to draw some inspiration for the solution and identify some gaps.

After saving and evaluating the detection model, the average recall (AR) for predictions with a maximum number of 10 bound boxes per frame was 94.45%. The average precision (AP) of each class after the model was evaluated is shown in Table 5.2.

Table 5.2: Table of Average Precision for each Object class/label

Class	*chicken*	*ketchup_portion*	*ketchup_bottle*	*potatoe_fries*	*empty_plate*
AP (%)	92.18	93.01	91.52	93.43	92.74

On saving the model to the Raspberry Pi, and setting a score threshold of 85%, the model made predictions as shown in Figure 5.2 and Figure 5.3.

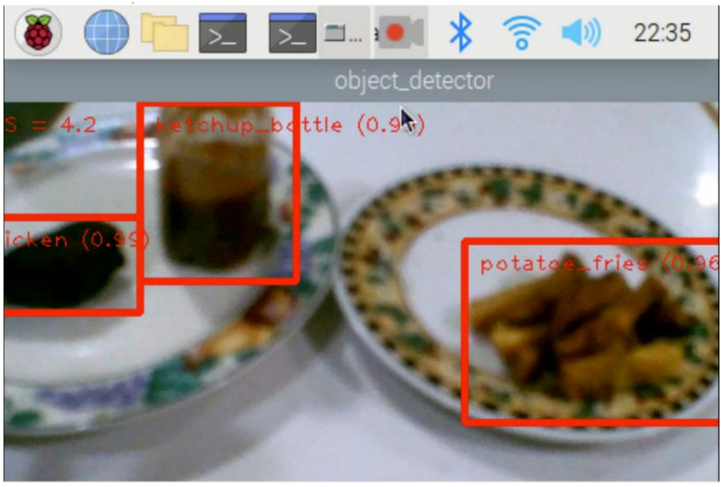

Figure 5.2: A screen grab of the Raspberry Pi while the model was in use

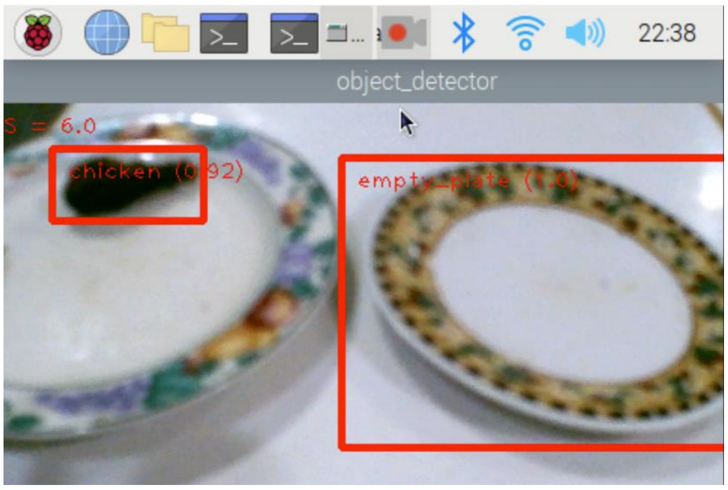

Figure 5.3: A screen grab of the Raspberry Pi while the model was in use

5.3 Assembling the Manipulator

To prototype the solution proposed in Chapter 1, the parts of CAD model were 3D printed with a PLA (polylactic acid) filament. Following the print, the robotic manipulator was assembled using screws to attach the motors to the printed parts. All other electronics were subsequently assembled and connected as shown at the end of the design chapter. An image of the assembled model is shown Figure 5.4.

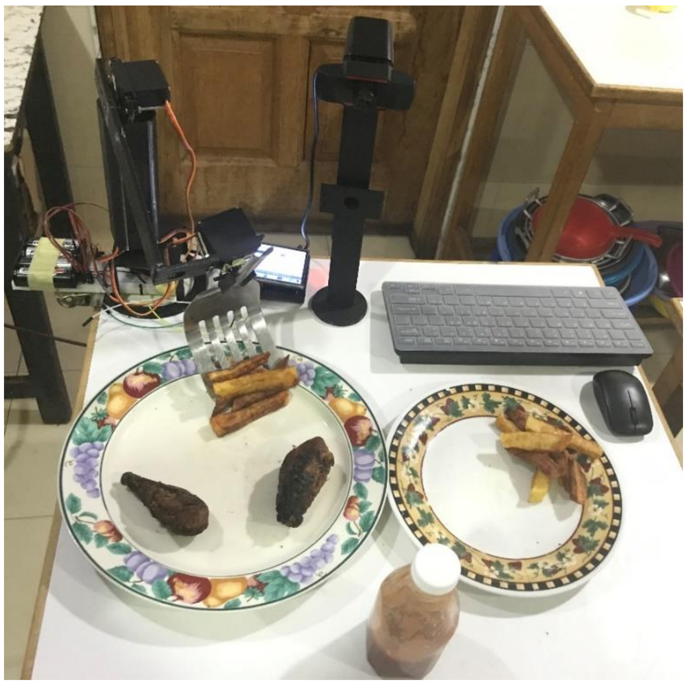

Figure 5.4: An image of the assembled manipulator

5.4 Real-time detections & Manipulator movements

Using the inverse kinematic equations of a 3-dof robotic manipulator discussed in the literature review, functions were written in python to get the position of an object, compute the joint angles and move the end effector to the correct position using the servo motors.

The model detected objects with the output being bound boxes, labels and scores. However, for the manipulator to have moved to the food item, it was important to convert the bound boxes to real world coordinates.

First, the centre of the bound boxes for the detections were computed as follows given a bound box [x min, y min, x max, y max].

$$bound\ box\ centre\ x, x_c = \frac{x \min + x\ max}{2}$$

$$bound\ box\ centre\ y, y_c = \frac{y \min + y\ max}{2}$$

Next, the centre point was converted to a 3D point [9]. To do this, OpenCV was used to calibrate the camera in order to get the intrinsic properties of the camera; focal length (f_x, f_y) and optical centre in pixels (c_x, c_y). Then, the x, y and z coordinates of the objects were computed as shown in the subsequent equations.

$$z = \frac{f_x *\ known\ width}{x \max - x\ min}$$

$$x = \frac{(x_c - c_x) * z}{f_x}$$

$$y = \frac{(y_c - c_y) * z}{f_y}$$

Chapter 6: Conclusion & Future Works

6.1 Conclusion

By means of the growing variation in dietary demands of customers, it has become more important for food service establishments to make their processes smarter. In this project, the aim was to explore a low-cost solution to this problem by using a robotic manipulator with a camera. With objectives including high precision for an object detection model to identify food items, this project demonstrated that affordability is feasible for building smart solutions and applying more Industry 4.0 technologies to food handling.

6.2 Future Works

Moving forward, it will be beneficial to invest more time into finding more cost-effective methods to deploy computer vision models for food related applications. It will also be useful to build embedded systems that can be integrated into existing commercial kitchen electronics instead of building new technology. Finally, for future work, exploring the use of sensors besides cameras could be valuable to maintaining safety in commercial kitchens while making Industry 4.0 more accessible to the food service industry especially through cost.

References

[1] H. Lasi, P. Fetteke, T. Feld and, M. Hoffmann, (2014) “Industry 4.0”, *Business & Information Systems Engineering*. Vol 6: Iss. 4, pp. 239-242. Available: https://aisel.aisnet.org/bise/vol6/iss4/5

[2] B. Farah and S. Rahimifard, (2020) “A Methodology for the Selection of Industrial Robots for Food Handling”, *Innovative Food Science & Emerging Technologies*. Vol 64: Iss. 3, 102379. Available: https://doi.org/10.1016/j.ifset.2020.102379

[3] G. Chen, and S. Liu, “Dynamics and Control of Robotic Manipulators with Contact and Friction,” in Robotic Manipulators, 1st ed. West Sussex, UK: Wiley, 2019, ch. 1&2, pp. 9-39.

[4] J. Denavit and, R. S. Hatrtenberg, (1955). “A Kinematic Notion for Lower-Pair Mechanisms based on Metrics”, *Trans.\ ASME E Journal of Applied Mechanics*. Vol 22, pp. 215-221. Available: https://doi.org/10.1115/1.4011045

[5] Y. Amit and, P. Felzenszwalb, (2014, January 10) “Object Detection”. University of Chicago. Available: htttps://doi.org/10.1007/978-0-387-31439-6_660

[6] R. Padilla, S. L. Netto, and, E. A. B. da Silva, (2020, July 21) “A Survey on Performance Metrics for Object-Detection Algorithms”, *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. pp. 237-242. doi: 10.1109/IWSSIP48289.2020.9145130

[7] M. Bollini, S. Tellex, T. Thompson, N. Roy & D. Rus, (2013), “Interpreting and Executing Recipes with a Cooking Robot”, *Experimental Robotics*. Vol 88, pp. 481-495. Available: https://doi.org/10.1007/978-3-319-00065-7_33

[8] M. Wise, M. Ferguson, D. King, E. Diehr and, D. Dymesich, (2016), “Fetch & Freight: Standard Platforms for Service Robot Applications”, *Fetch Robotics Inc*. Available: https://fetchrobotics.borealtech.com/wp-content/uploads/2019/12/Fetch-and-Freight-Workshop-Paper.pdf

[9] J. Weng, P. Cohen and, M. Herniou. (1992) “Camera calibration with distortion models and accuracy”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol 1: Iss. 10. pp. 965-980.

[10] L. Tzuta, (2021) “LabelImg”. Available at: https://github.com/tzutalin/labelImg

[11] M. Abadi, et. al. (2016) “Tensorflow: A system for large-scale machine learning”, *12th USENIX Symposium on Operating Systems Design and Implementation (OSIDI 16)*. pp. 265-283. Available at https://research.google/pubs/pub45381/

[12] T. Mingxing, P. Ruoming and, V. Le Quoc. (2020) “EfficientDet: Scalable and Efficient Object Detection”. *Google Research, Brain Team*. Available: https://arxiv.org/pdf/1911.09070.pdf

Appendix

7.1 Python Scripts

7.1.1 rename_img.py

```python
import os

def rename_images(folder):
    for count, filename in enumerate(os.listdir(folder)):
        newname = f"img-{str(count)}.png"
        img_path = f"{folder}/{filename}"
        newname = f"{folder}/{newname}"

        os.rename(img_path, newname)
        print(f"file[{str(count)}] renamed from {filename} to {newname}")
```

7.1.2 robot.py

```python
from servo import Servo
from time import sleep
from math import acos, asin, atan, degrees, sin, cos, sqrt

class Robot:
    def __init__(self, servo1_pin, servo2_pin, servo3_pin, link1, link2, link3):
        self.servo1 = Servo(servo1_pin, 50)
        sleep(1)
        self.servo2 = Servo(servo2_pin, 50)
        sleep(1)
        self.servo3 = Servo(servo3_pin, 50)
        sleep(1)
        self.cur_pos = [0, 0, 0] # x, y, z
        self.link1 = link1
        self.link2 = link2
        self.link3 = link3
```

```python
    def returnToOrigin(self):
        self.servo1.moveAngle(1)
        sleep(1)
        self.servo2.moveAngle(1)
        sleep(1)
        self.servo3.moveAngle(1)
        sleep(1)
    def moveToPosition(self, x, y, z):
        # set current position to x, y, z
        self.cur_pos = (x, y, z)

        # compute angle for joint 1
        theta1 = degrees(atan(y/x))

        # compute angle for joint 2
        a = (x^2) + (y^2) + (z^2) - (self.link1^2 * self.link2^2)
        b = 2 * self.link1 * self.link2
        theta2 = degrees(acos(a/b ))

        # compute angle for joint 3
        r = sqrt( (x^2) + (y^2) + (z^2) )
        c = self.link2 * sin(theta2)
        d = self.link1 + (self.link2 * cos(theta2))
        theta3 = degrees(asin(z/r) + atan(c/d))

        # move robot to position x, y z
        self.servo1.moveAngle(theta1)
        sleep(1)
        self.servo2.moveAngle(theta2)
        sleep(1)
        self.servo3.moveAngle(theta3)
        sleep(1)

        # return to robot's original position
        self.returnToOrigin()
```

7.1.3 camera.py

```
import sys
import cv2
from object_detector import ObjectDetector
from object_detector import ObjectDetectorOptions
```

```
def run(model, camera_id, width, height, num_threads):
  # Start capturing video input from the camera
  cap = cv2.VideoCapture(camera_id)
  cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

  # Visualization parameters
  text_color = (0, 0, 255)  # red
  font_size = 1
  font_thickness = 1

  # Initialize the object detection model
  options = ObjectDetectorOptions(
      num_threads=num_threads,
      score_threshold=0.9,
      max_results=3,
      enable_edgetpu=False)
  detector = ObjectDetector(model_path=model, options=options)

  # Continuously capture images from the camera and get predictions
  while cap.isOpened():
    success, image = cap.read()
    if not success:
      sys.exit('ERROR: Unable to read from webcam. Please verify your webcam
settings.')

    image = cv2.flip(image, 1)

    # Run object detection estimation using the model.
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    detections = detector.detect(rgb_image)

    # Draw bound boxes on input image
    for detection in detections:
        category = detection.categories[0]

        xmin = detection.bounding_box.left
        ymin = detection.bounding_box.top
        xmax = detection.bounding_box.right
        ymax = detection.bounding_box.bottom

        start = xmin, ymin
        end = xmax, ymax

        cv2.rectangle(image, start, end, text_color, 3)
```

7.1.4 object.py

```
import math

class Object:
    def __init__(self, name='object', bnd_box=[0,0,0,0], score=0.5):
        self.name = name
        self.bnd_box = bnd_box
        self.score = score
        x = (self.bnd_box[0] + self.bnd_box[2]) // 2
        y = (self.bnd_box[1] + self.bnd_box[3]) // 2
        self.center = [x, y]
```

```
    def get_object_position(self, known_width=10): # known width in cm
        x = self.center[0]
        y = self.center[1]

        width_pix = self.bnd_box[2] - self.bnd_box[0] # xmax - xmin
        # camera parameters after calibration with open cv
        # x & y focal lengths of the camera
        fx = 1.01228658 * math.exp(3)
        fy = 1.01955224 * math.exp(3)

        # x & y optical center points of the camera
        cx = 6.51698659 * math.exp(2)
        cy = 3.83238934 * math.exp(2)


        z_world = (fx * known_width) / (width_pix)

        x_world = ((x-cx)*z_world)/fx
        y_world = ((y-cy)*z_world)/fy

        return x_world, y_world, z_world
```

7.1.5 menu.py

```
from camera import run
from robot import Robot
from object import Object
from time import sleep
import cv2
import RPi.GPIO as GPIO
```

```python
def get_menu_option():
    try:
        menu_msg = menu_msg_intro

        for optn in menu_optns:
            menu_msg+= optn

        print(menu_msg)
        menu_option = input()

        return int(menu_option)
    except Exception as e:
        print(f'Input Error: {e}')
```

```python
def get_chicken():
    name = 'chicken'
    detections = run(model_path, camera_id, frame_width, frame_height,
num_threads)
    for d in detections:
        category = d.categories[0]

        xmin = d.bounding_box.left
        ymin = d.bounding_box.top
        xmax = d.bounding_box.right
        ymax = d.bounding_box.bottom

        if category.label == name:
            break
    obj = Object(name, [xmin,ymin,xmax,ymax], round(category.score, 2))
    x,y,z = obj.get_object_position(12)
    Robot1.moveToPosition(x, y, z)
    sleep(1)
    Robot1.returnToOrigin()
```

```
def get_chicken():
    name = 'chicken'
    detections = run(model_path, camera_id, frame_width, frame_height,
num_threads)
    for d in detections:
        category = d.categories[0]

        xmin = d.bounding_box.left
        ymin = d.bounding_box.top
        xmax = d.bounding_box.right
        ymax = d.bounding_box.bottom

        if category.label == name:
            break
    obj = Object(name, [xmin,ymin,xmax,ymax], round(category.score, 2))
    x,y,z = obj.get_object_position(12)
    Robot1.moveToPosition(x, y, z)
    sleep(1)
    Robot1.returnToOrigin()
```

```
def get_ketchup_bottle():
    name = 'ketchup_bottle'
    detections = run(model_path, camera_id, frame_width, frame_height,
num_threads)
    for d in detections:
        category = d.categories[0]

        xmin = d.bounding_box.left
        ymin = d.bounding_box.top
        xmax = d.bounding_box.right
        ymax = d.bounding_box.bottom

        if category.label == name:
            break
    obj = Object(name, [xmin,ymin,xmax,ymax], round(category.score, 2))
    x,y,z = obj.get_object_position(12)
    Robot1.moveToPosition(x, y, z)
    sleep(1)
    Robot1.returnToOrigin()
```

```
def assemble_plate():
    option = get_menu_option()
    try:
        while option != 7:
            print(f'Starting plate assembly for option {menu_optns[option-1]}')
            if option == 1:
                get_chicken()
            elif option == 2:
                get_potato_fries()
            elif option == 3:
                get_chicken()
                get_ketchup_bottle()
            elif option == 4:
                get_potato_fries()
                get_ketchup_bottle()
            elif option == 5:
                get_chicken()
                get_potato_fries()
            elif option == 6:
                get_chicken()
                get_potato_fries()
                get_ketchup_bottle()
            else:
                print('Invalid option selected, try again\n')
            print('-------------------------------------------\n')
            option = get_menu_option()

        print('Exiting Plate Assembly Script')
    except Exception as e:
        print(f'Menu Error: {e}')
```

7.2 Training Notebook

The python notebook where the model was trained can be found https://console.paperspace.com/larkuo/notebook/rncthelresxae8i.

7.3 Images and Annotation Files

All training files (images and annotations) can be found https://drive.google.com/drive/folders/1YtVByGrUQsYqhoLIwHVkW40qeQBHMYY0?usp=sharing.

All validation files (images and annotations) can be found https://drive.google.com/drive/folders/10a4BlyLBIEDgVBWC_ttRKQXRhKuF2iKj?usp=sharing.

7.4 CAD Model

The CAD assembly can be found on Onshape from https://cad.onshape.com/documents/f5559b3a8d1dfa5452cc9610/w/2c8fd2f6f4ca79821d800561/e/9566d325d534fbadf0bd8f40?renderMode=0&uiState=6269d6de24af24506098aa80.

7.5 GitHub Link

Find all scripts, notebooks, data and, CAD Model on GitHub at https://github.com/Larkuo/undergrad-capstone.