School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

The Robot Shopkeeper: An Integrated System for Robotic Manipulation and Perception

Kieran Marshall Haden

Submitted in accordance with the requirements for the degree of BSc Artificial Intelligence

Session 2015/2016

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final printed report (x2)	Report	SSO (10/05/16)
Online report	PDF report	VLE (10/05/16)
Github code repository	Software code and other deliverables	SSO (10/05/16)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

The aim of this project is to produce a piece of software using the robotic operating system (ROS) that will allow Baxter to interact with people and give them sweets within a shop-like setting. The personal aim of this project is to learn the algorithms and software systems used in autonomous systems whilst doing this project. Practically, the aim is to allow the School of Computing to be able to use this system as an interactive shopkeeper demonstration on Open Days.

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Mehmet Dogar, who gave me very useful advice throughout all of our meetings and really inspired me in many stages of the project. We met on a very regular basis and it helped a lot to be able to discuss ideas and theories with him to help me get a more clear idea of ways I could implement things. I would also like to thank Dr. Anthony Cohn, who acted as a secondary supervisor. He has helped guide me and Mehmet throughout this entire process. Another person I couldn't have done without was Muhannad, one of the PhD students in the robotics lab. His knowledge of Baxter, the Kinect and robotics in general was key when I started this project. Without his help I would've been stuck on some features for a lot longer.

I would also like to thank Dr Mark Walkley, who is my assessor for this project. Throughout the scoping and planning document and the presentation, he gave invaluable insight and opinions on my ideas for the project.

I thank my amazing fiancée for putting up with me over this three month period, especially towards the end, where I have no doubt put her through many nights of stress and worrying.

Finally, I need to thank Baxter for not malfunctioning too badly throughout the project. I have only had one broken Kinect stand and a stuck gripper motor but overall, he has stayed in one piece.

Contents

Chapter 1

The Problem

The main task for this project was to develop a method for a Baxter robot to run a sweet shop and serve customers. This task was not as simple as it sounds, as it involved multiple computing areas and many practical issues along the way. Overall, a satisfying product was made that allowed Baxter to interact with a customer, listen for what they want and give them the required sweets. Throughout this report, I will discuss the process in which the project was developed, from initial conceptions to the final product.

1.1 Aim

The overall aim of this project was to produce a piece of software using the robot operating system (ROS) that would allow Baxter to interact with people and give them sweets within a sweet shop setting. The personal aim of this project was to learn the algorithms and software systems used in autonomous systems whilst doing the project. Practically, the aim was to allow the School of Computing to be able to use this system as an interactive demonstration on Open Days.

1.2 System Logic

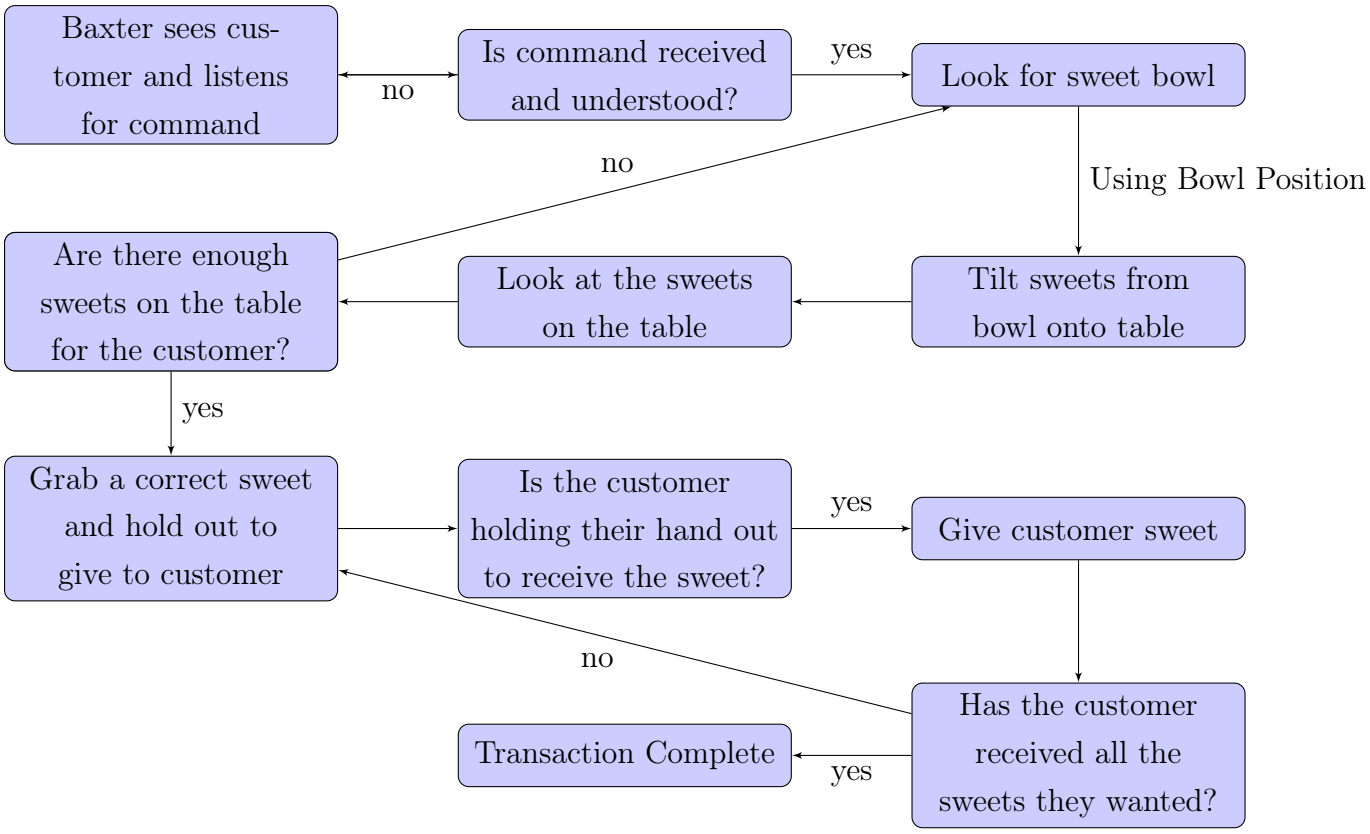
This project was a hugely ambitious one, with an infinite number of approaches and solutions to each part of the interaction. To split the project into a series of simpler steps, an example interaction between Baxter and the customer was proposed. To develop the system logic, it was easier to look at the simple interactions needed between a shopkeeper and their customer. Firstly, human interaction would determine what sweets the shopkeeper would need to get. Secondly, the shopkeeper would get some sweets from the bowl and individually pick out the ones the customer wanted. The shopkeeper would give those sweets to the customer and then ask for payment of some sort to complete the interaction. During the development of the project, more clear and precise logic steps came from development, shown in the **Flow Chart** below.

1.3 Approach

To develop an initial approach for the project development, the plan was to separate the shopkeeper-customer interaction into separate areas of research. With the perception of the environment and looking for objects, machine vision was important. Robotic

manipulation techniques were important for Baxter's manipulation of the different objects - the bowl and the sweets. Human interaction was important to help make the experience more believable. All these areas were further researched before moving on to separating the project into different tasks, as explained in further chapters.

1.4 Flow Chart



Chapter 2

Background And Literature Review

The idea of this project was to produce a fully functioning piece of software that allows Baxter to converse with customers in a sweet shop environment. To do this, background research was taken into the main areas I would need knowledge of in this project: robotic systems, machine vision, manipulation and human interaction. To develop a robotic architecture, it was necessary to look at some other robotics systems to see the main steps they took to plan and build them. The Herb 2.0 paper [1] separates the robot's architecture into several main components that make up the overall system: Robot Behaviour, Manipulation Planning, Planning Under Clutter and Uncertainty, Interacting with People and Perception. These aspects are were further researched below.

2.1 Robotic Behaviour

Robots use a behaviour engine to model their reactions to the environment in a logical way. This simple logic can be transferred to the shopkeeper project by Baxter having different behaviours to carry out in the shop. Those could be listening to a customer's command or picking up a particular sweet. All these behaviours and the logic along with them should be developed for make a robust set of behaviours for Baxter to use.

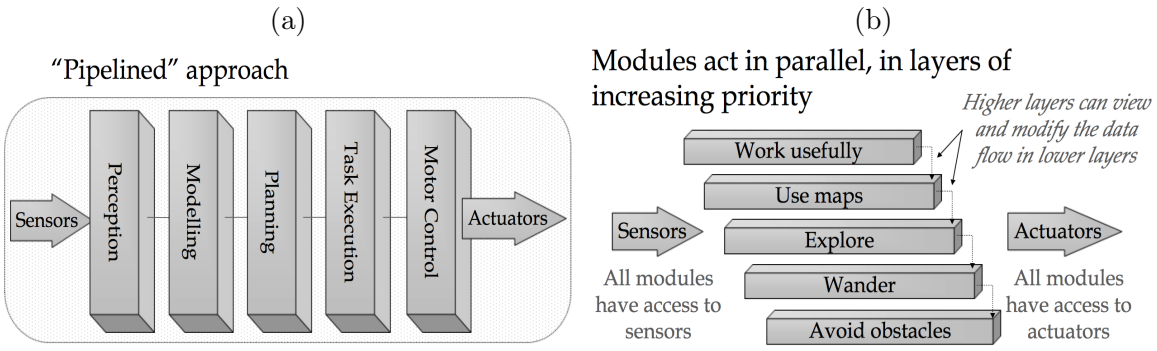


Figure 2.1: Robotic architectures: (a) Classical approach. (b) Subsumption approach.

Two main architectures exist to develop robotic behaviour have been used in multiple robotic systems: a classical approach and a subsumption architecture [2]. A classical architecture is based on a pipeline-like approach, where first perception occurs, building up a model of the world, then from that model, planning is made then tasks are executed, shown in **Figure 2.1a**. A subsumption architecture works in more of a layered model, where multiple tasks run at the same time, so the layers act in parallel.

That means that tasks such as exploration and wandering would run together, both with access to the robot's motors (shown in **Figure 2.1b**).

The simplest approach to the shopkeeper's architecture seems to be a classical pipeline approach. It doesn't make sense for a subsumption architecture, since Baxter will need to re-analyse his environment before planning any manipulation tasks. Therefore a basic logical behaviour architecture could be theorised for a simple task of picking up sweets. In an example task, Baxter would want to pick up a sweet from a pile of sweets on the table. This could be split into a logical order of behaviours: recognise the sweets on the table, plan which sweet he wants to pick up, then pick up the sweet. This has to be done in this order, as Baxter will need to build up a model of his environment before he can perform the task.

2.2 Machine Vision

Machine vision is the area of computer science where images can be analysed and understood. This is often used in robotics to help robots recognise and understand objects in their environment so they can react and respond appropriately. In the sweet shop, Many robotic vision systems are used in conjunction with manipulation and other aspect's of the robot's behaviour, so a robot will recognise and understand it's surrounding and interpret that in real-time to feed the information back to other processes. For example, this technique could be applied to Baxter looking at the table and then telling the manipulation process where a sweet is located to pick it up.

2.2.1 Image Segmentation Techniques

There have been multiple approaches to object detection in machine vision. The approaches analyse pixels by segmenting images into the desired edges/distinguishing features. The main object detection methods found in preliminary research that could be applied to sweet recognition in the project were active contour methods and template matching.

There are multiple methods of active contour detection in images, all based around the principle of detecting object boundaries, iteratively moving towards the ideal border using image processing and user input [3, 4]. Contours work by assigning a general snake-like shape to roughly form a border around the target object. After, internal and external spline energies are calculated to determine where the border should be attracted to, whether it be a strong edge gradient (a defined edge on an object), a dark ridge or a termination of an edge. This energy potential is calculated using the edge, line and term terms using the formula below:

$$\varepsilon_{\text{image}} = \omega_{\text{line}}\varepsilon_{\text{line}} + \omega_{\text{edge}}\varepsilon_{\text{edge}} + \omega_{\text{term}}\varepsilon_{\text{term}}$$

Here, ε refers to energy, ω represents the weight of the line, edge and termination features. The contour 'evolves' by using this formula to minimise the energy, which moves inwards towards a contour that perfectly outlines an object, shown by the moving 'snake' contour in **Figure 2.2a**. The fact that the contour energies need to be minimised means that it is helpful to initialise a contour external to the object's border. Active contour methods are commonly used in situations where a defined border exists to clearly view an object, which could come in handy when recognising sweets on a table later on.

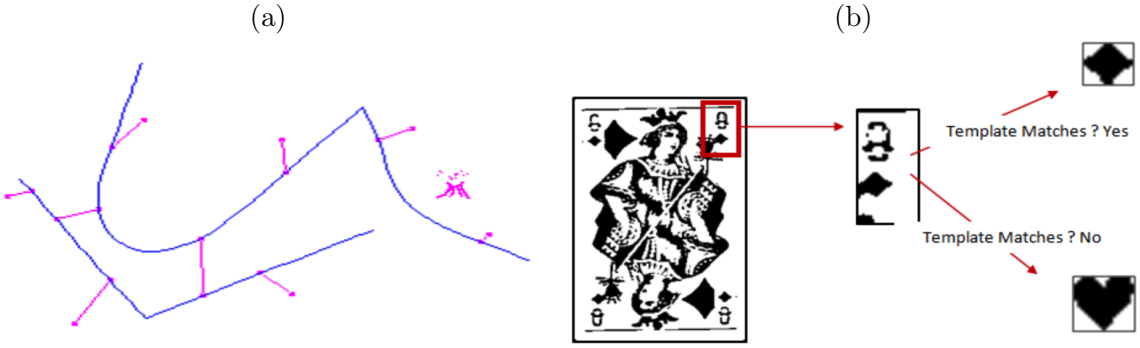


Figure 2.2: Image processing techniques: (a) A snake contour being attracted to a dark edge. (b) An example of template matching on a playing card.

Another method which could be applied to recognising sweets is template matching, where the shape of a sweet could be learnt to recognise them in different orientations. Template matching uses an approach using the template image (for example, a picture of a sweet) and tries to match that image to an area of the target image in a particular orientation. An example of template matching is shown in **Figure 2.2b**, where numbers and suit template images can be detected on the playing card. The key aspects of this technique is establishing a way to measure the error metric, so the template and image can be compared. Also, a successful search technique needs to be devised to search where the template matches occur and at what rotation. A common error metric is the least squares method, where each x, y pixel is compared to an u, v position in the target image [5]. Search methods need to be developed appropriately too, as an exhaustive search is inefficient to search all orientations. Some solutions to this problem include limiting the space to search over, such as hierarchical motion estimation, where an image pyramid is created, searching first at more coarse levels, then each of those searches is used to initialise a more local search at the next level [6]. This method is also a viable method to search and locate sweets on a table image.

2.2.2 Pointcloud Analysis

Vision techniques used in a 3D Pointcloud have a slightly different approach. Instead of segmenting and processing the pixels in an image, object analysis is more concentrated around grouping of points that are close in orientation and colour.

Pointcloud's are analysed in a different way to regular 2D images. However, the same basic image processing principles still apply, segment out the area you want to analyse and use a recognition technique to provide further recognition. A main method of initially segmenting pointclouds is to use RANSAC [7], which can be used to find co-planar points. RANSAC (or random sample consensus), is a technique first developed as an alternative to a least median square method. To find points on a particular plane, a targeted sample of vertices from the pointcloud are analysed and checked whether they mathematically lie near enough to each other to belong to the same plane. These planes can then be used to find manmade objects within the 3D cloud, with flat, parametric surfaces. This technique is commonly used to segment objects from unwanted surfaces, for example, separating objects on top of the shop counter from the table itself. RANSAC can also be used to identify simple 3D shapes such as cylinders by searching over points that fit a mathmatical model, as shown in **Figure 2.3a**.

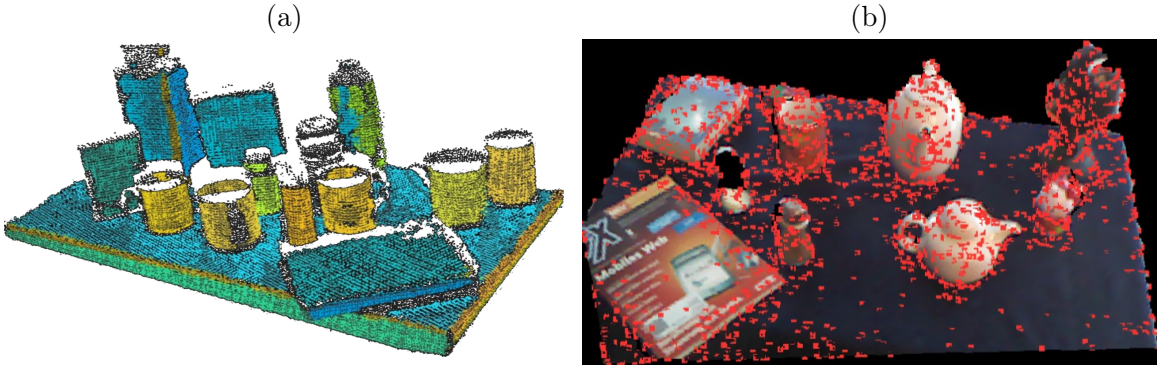


Figure 2.3: Pointcloud processing techniques: (a) RANSAC detection of cylindrical objects. (b) 3D-SIFT detected keypoints in an object pointcloud.

Another method to recognise objects within a pointcloud exists as a 3D expansion similar to a template matching method, where a 3D model can be built of a target object [8]. To create a single model representation of an object, multiple images can be combined into one 3-dimensional representation, allowing the object to be recognised from multiple viewpoints. Once the model has been built, SIFT (Scale Invariant Feature Transform) features can be retrieved from a 2D RGB view of the object, which can retrieve a vector of feature points invariant to translation, scaling, rotation and illumination, shown in **Figure 2.3b**. The SIFT features can then be matched to their nearest neighbours with the closest Euclidean distances, which can then determine if

that matches the target object at a specific 3D orientation. This would then mean that the object would then be recognised and at what orientation in the pointcloud. This can be used as an approach to find 3D objects on the table, such as a bowl holding the sweets.

2.3 Manipulation Planning

Manipulation planning is a key aspect of robotic systems, where the hardware of the robot and the limitations need to be considered before integrating manipulation techniques. Manipulation tasks in the sweet shop include tasks such as picking up sweets, giving them to the customer, which will all require planning for the position and movement of the grippers.

2.3.1 Inverse Kinematics and Planning

Inverse kinematics is one of the common movement planning techniques in robotics. This involves a kinematic analysis of the current state of the robot's system, looking at the all of the joint positions and angles. Forward kinematics works out where the endpoint of a manipulator is, given the angles between each limb (i.e. joint angles). Inverse kinematics does the exact reverse of this, where you specify a coordinate endpoint of the limb and the joint angles between each link will be worked out. Baxter's inverse kinematics requires a specified x, y, z coordinate for an effector's endpoint, along with a quaternion for the rotation of the arm.

After the inverse kinematics calculations have been done, a planning algorithm has to be used to decide which is the best way Baxter can move his arm to the calculated joint angles. One of the main issues with planning is that commonly, there are multiple paths to take and it is not always clear which one is the best to take. The most obvious solution to choose would be to select the one which contains the least overall movement to get to the right location. The problem with the multiple solutions is that sometimes, unexpected routes can be taken instead of the desired one. This usually tends to occur because a perceived human solution to a joint position will not be the closest solution in the joint's coordinate system, rather than the world coordinate system [9].

2.3.2 Grasping Methods

In the shopkeeper-customer interaction, there will be multiple parts of the interaction that require Baxter to grasp objects in different ways. One task that needs to be done is for Baxter to grasp sweets from a table. Multiple techniques could be used to do this, but a sensible approach would seem to be using a PCA (principal component analysis)

to get the correct grasping angle to approach the sweet [10]. Once the angle was obtained, Baxter could grasp the sweet from the correct angle to pick it up.

Further manipulation issues could be encountered if Baxter wants to exchange money between him and the customer. This would be due to notes being soft-bodied, providing issues in separating notes and grasping them to begin with. There are multiple areas of research into soft-bodied grasping and manipulation, which is an unsolved problem in robotics as of yet as the object changes shape while being grasped. One approach to do this is done using robot towel folding using cloth grasping [11].

2.4 Planning Under Clutter

Planning under clutter could be an important aspect of the project depending on how the sweets are placed on the table. If they were grouped together, Baxter would have to plan to separate them from the clutter. Also, other objects on the shop's counter could cause clutter issues with recognition and manipulation planning.

One task that needs to be done, is for Baxter to be able to grasp and separate sweets. Separating the sweets out on the table with grasping methods could be done in multiple different ways, such as picking the sweets up and dropping them from a height onto the table [12], or scooping some sweets up with the gripper and looking at the sweets in Baxter's hand at that time. Multiple planning methods need to be incorporated to separated groups of sweets on the table, using a vision system that can separate the pile into graspable and non-graspable sweets.

Other tasks that could involve planning under clutter could be perception tasks in clutter. With multiple items on the shop's counter, such as sweets, containers and possibly other items, the vision systems have to be accurate enough to ignore other objects and clutter and concentrate on the desired shape/size of the target object.

2.5 Interacting with People

Human interaction is key to making the robot shopkeeper a realistic and comprehensive experience. By testing with human customers, this system could be made robust and efficient. This is important for Baxter to understand the customer's request and interact with them accordingly in the shop.

Voice recognition is a useful feature to integrate robot and customer interaction in the shop. Multiple techniques have been developed over time for speech recognition [13],

some of which are Python's NLTK module. Using these principles, speech can be converted to text and then analysed for commands the customer can give to Baxter.

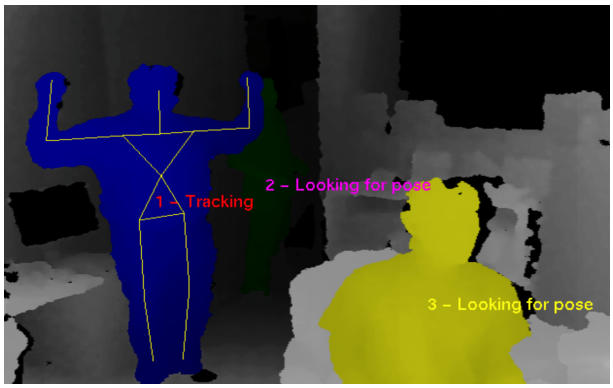


Figure 2.4: Openni_tracker tracking skeletons and poses from a Kinect depth image.

Another useful feature in Baxter interacting with the customer is skeleton recognition, which would allow Baxter to recognise the gestures the customer is doing, such as reaching for a bag of sweets or reaching out with some money. Different approaches have been used for skeleton recognition, but a relatively easy implementation of this is to use the Kinect's in-built skeleton recognition to detect people and their positions in front of the camera. **Figure 2.4** above shows an example of how simple poses and skeletons can be tracked using Openni_tracker[1]. This technology could ideally be used to detect someone holding their hand out to exchange money with Baxter in the shop.

[1] ROS Wiki: Openni_tracker. `http://wiki.ros.org/openni_tracker`

Chapter 3

Objectives

3.1 Objectives

After doing the appropriate research, some main objectives were laid out below as an initial order of tasks/features to implement:

1. Develop a vision system to recognise the container that the sweets are in and a manipulation method to retrieve them from the container.

2. Include a vision system to recognise the sweets in their environment and a manipulation method to separate/singulate sweets into the desired amounts.

3. Integrate methods of human interaction between Baxter and a customer to allow them to order some sweets from the shop.

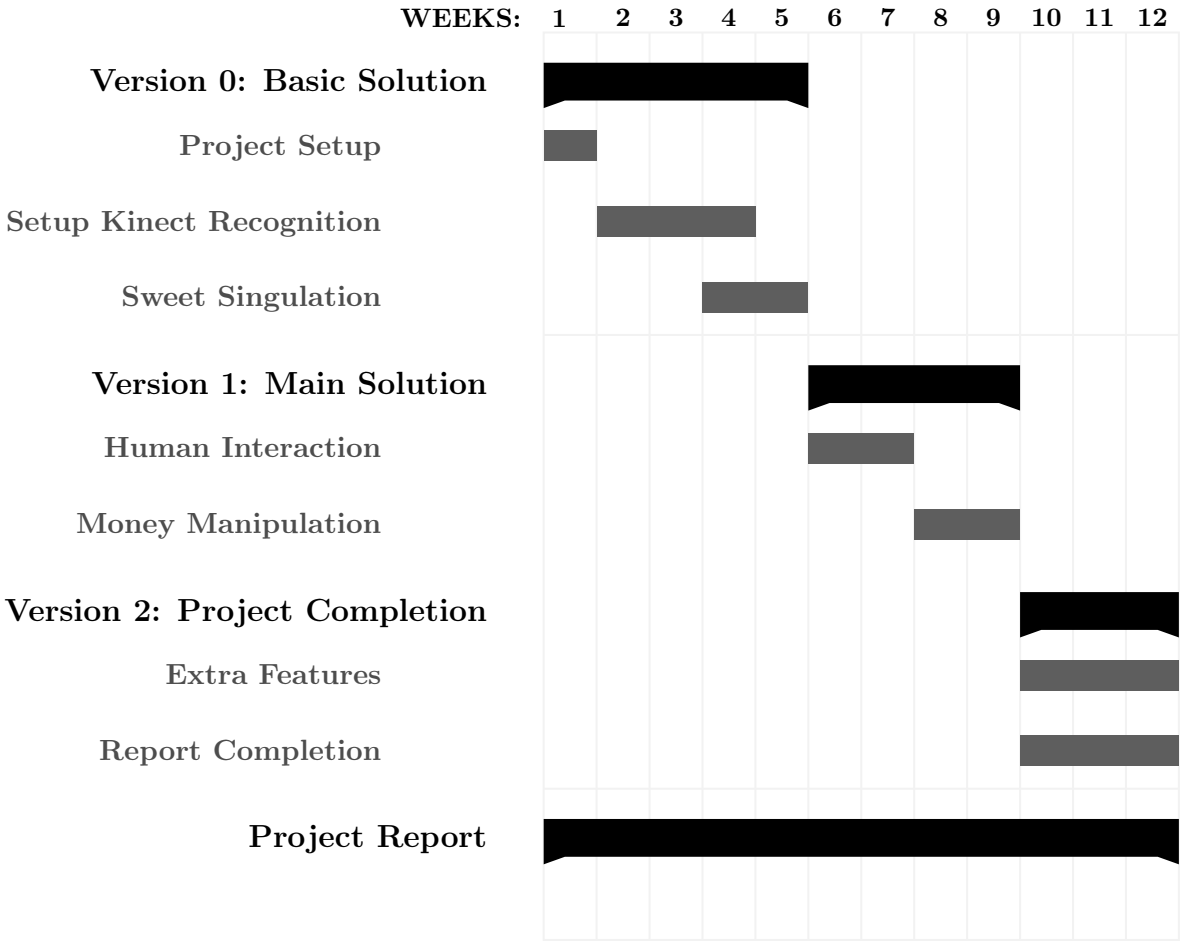
4. Create a piece of software that would allow Baxter to implement all the previous steps into one integrated architecture. This would allow Baxter to interact with a customer in the sweet shop and give them whatever sweets they asked for.

3.2 Methodology

Since this project involved a lot of different areas of robotics integrated into one system, a somewhat modular approach had to be taken. It was decided that each area (vision, manipulation, human interaction) would be developed in a sequence. The method therefore has a very testing-centric agile approach, where for a task, there would be planning for multiple approaches, development sessions and then testing. Once an efficient solution had been found (determined with testing), the next task could then start being developed. Since the methodology was so linear, by looking at the large amount of tasks in the **Gantt chart** to complete, multiple contingencies had to be taken to make sure that certain tasks didn't take too long or were too complex.

As you can see from the **Gantt chart**, there are multiple target versions for the project, in case some tasks run over their time limit. The basic solution was aimed to be complete in 6 weeks, but the tasks for version 1 and 2 could've been seen as extra objectives if there wasn't time to implement them all. Another contingency would also be the heavy planning and supervisor discussions before each task, meaning I knew what to read beforehand and some of the common approaches to solving each problem.

3.3 Gantt Chart



Here are each of the tasks explained in more detail:

- **Project setup** - Setup the ROS system and look learn the basics of the software. Complete ROS and Baxter tutorials and make some example programs to test and move Baxter in different ways.
- **Setup Kinect recognition** - Setup the Kinect with Baxter and look into different computer vision methods to recognise the bowl and sweets and their positions in 3D space.
- **Sweet singulation** - Trial multiple object manipulation algorithms to allow Baxter to grab and select specified sweet combinations.
- **Human interaction** - Create some form of human interaction between Baxter and a human to allow the human to ask for specific sweets. This could be gestures or voice control based.
- **Money Manipulation** - Develop some vision system to recognise paper money or change and then work on algorithms for Baxter to handle/count out money to give

to a person.

- **Extra Features** - Allow time to further improve upon previous steps or pick one dedicated topic to further go into and develop.

- **Report Completion** - Allow the last three weeks to further improve upon the project report.

- **Project Report** - At each week, take a look at project report deadlines that need to be met. Make sure parts of the report are written at the time they are completed.

3.4 Changes from the Initial Plan

The **Gantt chart** above, whilst reasonably useful to lay out the project, there were multiple flaws that were found throughout the time spent on it. Firstly, as you can see, there were multiple tasks that only allowed 2 weeks to complete, where that time would have to include planning, research, coding and testing. This was unrealistic for some tasks as they either were more complex than anticipated or there were unexpected technological issues that added more time on overall.

The vision tasks had a lot more challenges and issues than initially expected, so a lot of time had to be allowed for them. Manipulation tasks, whilst they were a bit quicker to develop, physically testing those took longer, as the manipulation tasks had to run many times to record the results.

As predicted, the tasks in version 1 and 2 could not be fully completed due to time restraints. Therefore, human interaction only was done partially and money manipulation was too complex of a task to implement in the time I had left. Therefore, looking back, it would have been better to remove that task and make some of the previous tasks allow three weeks of time instead of two to get a higher quality solution.

In general however, the Gantt Chart did a reasonably good job of laying out a basic plan. Whilst some tasks took longer than expected, some took a shorter amount of time, so the task times tended to average out. The idea to research, plan and discuss before starting each task really helped speed-wise, allowing a focused approach to be used on each. By the end of the project, a reliable piece of software had been produced that included a large percentage of the planned tasks and others that weren't even considered before. Other changes to the objectives and planning are explained in the later sections of the report.

3.5 Determining the Quality of the Solution

The quality of the solution could be determined by the testing of each stage of the project. The idea of this approach meant that once each feature was implemented into the system, it could be tested to determine its efficiency. If the efficiency of the feature was not high enough, then that feature could be re-implemented via another method. By doing this testing for each feature, it meant that by the end of the project, when integrating the whole system together, the system would be reliable enough to be a reasonably robust piece of software. After the testing of methods and determining when one was efficient enough to include in the final system, it was decided the results of those tests would be used for evidence and deliverables to be included in the report.

To determine the quality and efficiency of the final system, it was decided that human testing would be the most appropriate way to do this. If you could test the system with multiple people walking up to Baxter and ordering their sweets, the quality of the solution would be determined by how many interactions produced the desired request and how many had errors.

Chapter 4

Materials and Software

After planning the report, the first week was used to introduce myself to the main components of the project. It was necessary to get used to the new software and materials so that I could find out how to use them for the required tasks.

4.1 The Robot

A Baxter robot is a programmable automaton which is already being integrated into workforces around the world [**?**]. This robot (**Figure 4.1a**) is currently very popular to use at universities as a safe robot for research purposes and this project will use the one at Leeds.

Figure 4.1: Baxter robot specifications. (a) Full torso of Baxter. (b) Seven labelled arm joints, present on each limb.

Baxter has three cameras to use, a right hand, a left hand and a head camera, each of which can output different resolution images for use in machine vision. He has two electronic grippers which can be customised to grip objects of different sizes, with pressure sensors to detect whether he has grabbed something. Seven joints are located on each arm (shown in **Figure 4.1b**), with humanoid movement, which can be controlled by specifying specific joint positions or torques at each joint.

4.2 Kinect

The Kinect has two cameras, used to detect objects with depth detection. It has been utilised in many areas of robotics as a more accurate vision method, creating 3D objects

by looking at the Kinect's pointcloud, an array of points with an x, y and z coordinate, containing RGB values for each one. The Kinect in the robotics department was set up to work with Baxter. It was calibrated with a rectangular checkerboard to set up the Kinect drivers, which could then set up the ROS transform (explained in **Section 4.4.2** below) between Baxter and the Kinect. Baxter could then know his own coordinate system with respect to the Kinect.

4.3 Software

There were multiple pieces of software that I had not used before, so some research needed to be done into them. The most helpful tutorials and files on most of the software were located in their respective online documentation.

4.3.1 OpenCV

OpenCV is a very useful open source implementation of vision techniques and functions. Having not used OpenCV before, I decided to write the OpenCV code in this project in Python (as there is a C++ and Python version of the module). It was very easy to pickup from the documentation and tutorials and most of the features were well explained online. It is written in an optimized C/C++ library, which turned out to be very quick and efficient to use on image analysis, even from Baxter's camera video feeds outputting high resolution images.

4.3.2 PCL Library

The PCL library is a C++ library which is open source and written in C++. It allows you to process and analyse pointclouds. It is rather poorly documented and there weren't many examples to find online so it was very much a trial and error to learn the functions and processes in this library. One of the main problems is it required a lot of manual manipulation and conversions of pointcloud types. There were multiple types of pointcloud data such as PointCloudXYZ, PointCloud2 or PointCloudXYZRGB, which would have to be converted and switched between depending on the function you wanted to use. Manual manipulation of pointclouds was also difficult, as you needed to use C++ boost iterators to access the indices of the points before then implementing the custom algorithms over the whole cloud.

4.4 ROS

ROS (or robot operating system) is a very interesting piece of software that is open-source and very widely used to program robots. It is a very useful, adaptable piece

of software that is written using it's main features of nodes and topics. Nodes are pieces of code that can be constantly run in a terminal window and be in contact with other nodes. Topics are custom data items that are constantly published as information from Baxter to the terminal. ROS nodes can access these topics to get information from Baxter or send information to him. Custom topics can be created and published via ROS to publish custom information between nodes.

For example, in Baxter's vision system for the shop, Baxter uses his right hand camera to view the sweets on the table. To carry out the structure of this task in ROS, a sweet vision node is created. That node would accept the right hand camera image topic to retrieve the sweet image to analyse. Then, after analysing the image, the node would then publish a list of the sweet locations on a custom topic, with a different node receiving those positions, so Baxter knew where to pick them up.

4.4.1 RViz

RViz is a very useful ROS package that was used in almost every aspect of the project. It can be used to visualise the many topics within ROS. The idea of RViz is that if any information is published as a topic, then RViz should be able to access and display that information in real-time. Baxter's robot model can be imported into the software and it's movements will be constantly updated, as shown below in **Figure 4.2**.

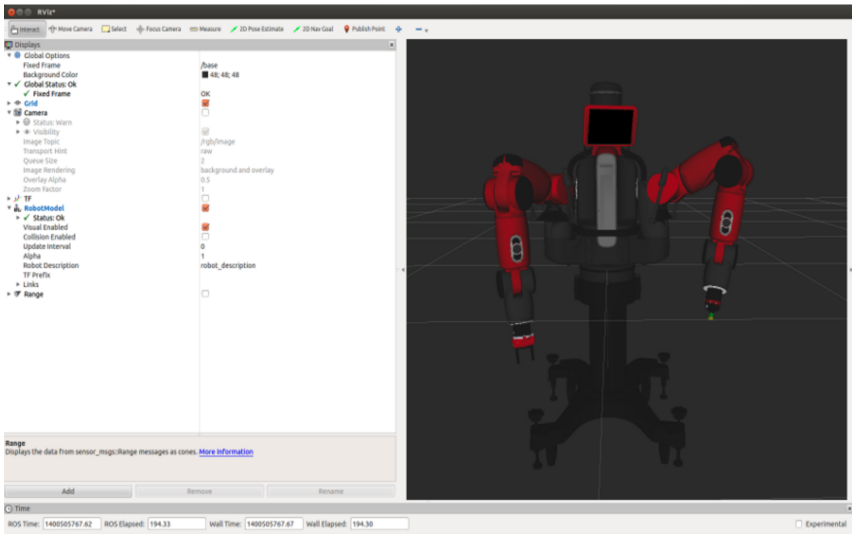


Figure 4.2: Baxter's model imported into RViz.

From the left hand toolbar, a multitude of extra visualisations can be added to view alongside Baxter: the Kinect's depthcloud, axes transforms for any fixed frame and more. Throughout the project, if a vision system needed to detect something, RViz could be used to make sure it was correctly aligned with Baxter and the Kinect. This technique was used and can be seen many times in the Methods section of the report.

4.4.2 Fixed Frame Transforms

A key aspect of creating the integrated movement system was understanding the concept of fixed frames and transforms. In ROS, Baxter contains information for multiple transforms between every joint and sections of his body. These are known at all times to help know where a point is relative to one part, say the hand, against Baxter's torso. This method works similarly for the Kinect, where Baxter's torso coordinate system, is linked to the Kinect's coordinate system via a transform, which can be accessed by certain methods within ROS.

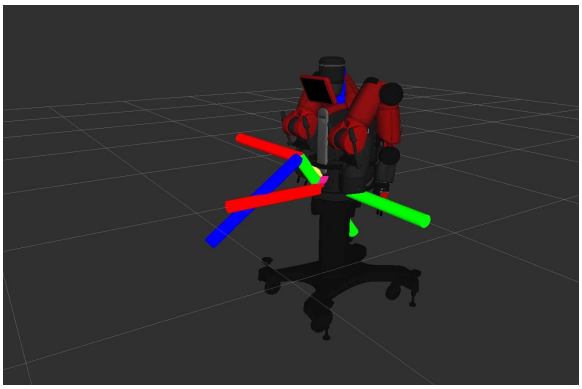


Figure 4.3: Image showing axes transforms between the Kinect and Baxter's torso frame, with the red, green and blue lines representing the x, y and z axes respectively.

Fixed coordinate systems were constantly used in testing the vision systems in this project, mainly within RViz. Everytime a coordinate or shape was recognised within the vision system, they could be published to RViz using a PointStamped object, which can be published in a particular coordinate system. Therefore, by publishing in one frame, transforming then publishing it another, RViz can show whether a point is correctly detected and whether it is in the correct frame or not.

This principle was used in the bowl recognition system, where after the centre of the bowl was obtained, Baxter needed to know where the centre of the bowl was in it's main torso coordinate system before a movement command could be made. The system then looked up the transform between the Kinect and Baxter's torso to transform the 3D coordinate between coordinate systems. In **Figure 4.3**, RViz shows the axes for the Kinect's frame and Baxter's torso frame.

4.4.3 Custom Service Requests

The idea that Baxter's movements would all be based on the current states of his vision system - depending upon where both the sweets and the bowl were, meant there needed to be a method developed for Baxter's movement system to request information from the vision system. This was implemented via Services, which is a technique in ROS that

allows a custom message to be sent, and then received between two nodes.

Custom services were extremely useful as a method to send information back and forth between the nodes, which needed to be utilised a lot when integrating the whole ROS system together. Whilst I had implemented and coded multiple nodes for each aspect of the system, they only worked on their own. Without one main node being able to retrieve all of the information for each system, the system wouldn't be able to work together. For example, a custom service request is shown and explained below.

```
string reset
...
float[] sweetPositions
```

This request was stored in a .srv file and was used in the shop's integrated system. The main shop node uses this service file as a request by first, publishing the service on a new node and sends a 'string reset' item within that request. Then, the sweet vision system has a constant client listening out for this service message and then, when it received it, it calculated the current sweet positions and sent back the message with those positions stored in the 'float[] sweetPositions'. Then the main shop node can receive and process these positions. As you can see, this approach came in very handy and was used quite often for the main node to retrieve information for the shop.

4.5 Github Repository

All of the code files mentioned throughout the upcoming chapters of the report are stored within the src folder of the Github repository, so they can be cloned directly into a created ROS workspace. The src directory is divided into multiple catkin packages - perception, interaction and manipulation to make the files easier to navigate. There is a testing and videos folder, providing evidence for the testing and data collected. Finally, there is a report folder, containing multiple draft versions of the report during it's development, from the initial version 0 to the final draft. Overall, the repository has been well maintained and the commit messages are self explanatory in terms of their purpose.

Chapter 5

Methods

This chapter discusses the methods and approaches implemented for each part of the shopkeeper project, along with the data and testing information. I have tried to include reports of changes and limitations of each feature where appropriate

5.1 The Bowl

The bowl is a container Baxter uses to store his sweets in. The eventual aim of using the bowl was for Baxter to be able to recognise it on the table, scoop some sweets out and move on to giving them to the customer.

5.1.1 Recognition

The first thing Baxter needed to do in his shop is to be able to recognise the bowl. This section shows how Baxter uses the Kinect to find the bowl on the table, identifying it from a range of other objects. The Kinect produces a pointcloud, a set of 3D points in the Kinect's coordinates system. The first task to do when recognising the bowl, was to pick a type of bowl that would be easily visible in the pointcloud. Multiple bowl types were trialled, until it was decided white polystyrene bowls would be a good overall choice. This is because, unlike other bowls, the rim was always clearly visible in the cloud and polystyrene is an easy material to grab and manipulate. Plastic and foil materials were too reflective to show up constantly.

5.1.1.1 Segmentation and Recognition

To recognise the bowl on the table, the vision system first separates objects from the table, then separates those points into individual clusters. To eliminate noise, a colour segmentation is performed to find only the white objects on the table. The bowl can then be found by looking for the rim as a circle in 3D space. The overall process is carried out by multiple algorithms, shown in the images and explained below.

1. Tabletop Object Detection/Segmentation - This method works by detecting the table in the pointcloud by finding the dominant plane using RANSAC. Points above this plane are then considered to be objects on top of the table, which are then segmented from the points that belong to the table. This results in a pointcloud without any interference from the table. **Figure 5.1a** shows an example of the processed pointcloud with the bowl and sweets separated from the points on the table. This

separation from the tabletop was developed using the *pr2_object_manipulator* ROS package, located and explained here[1].

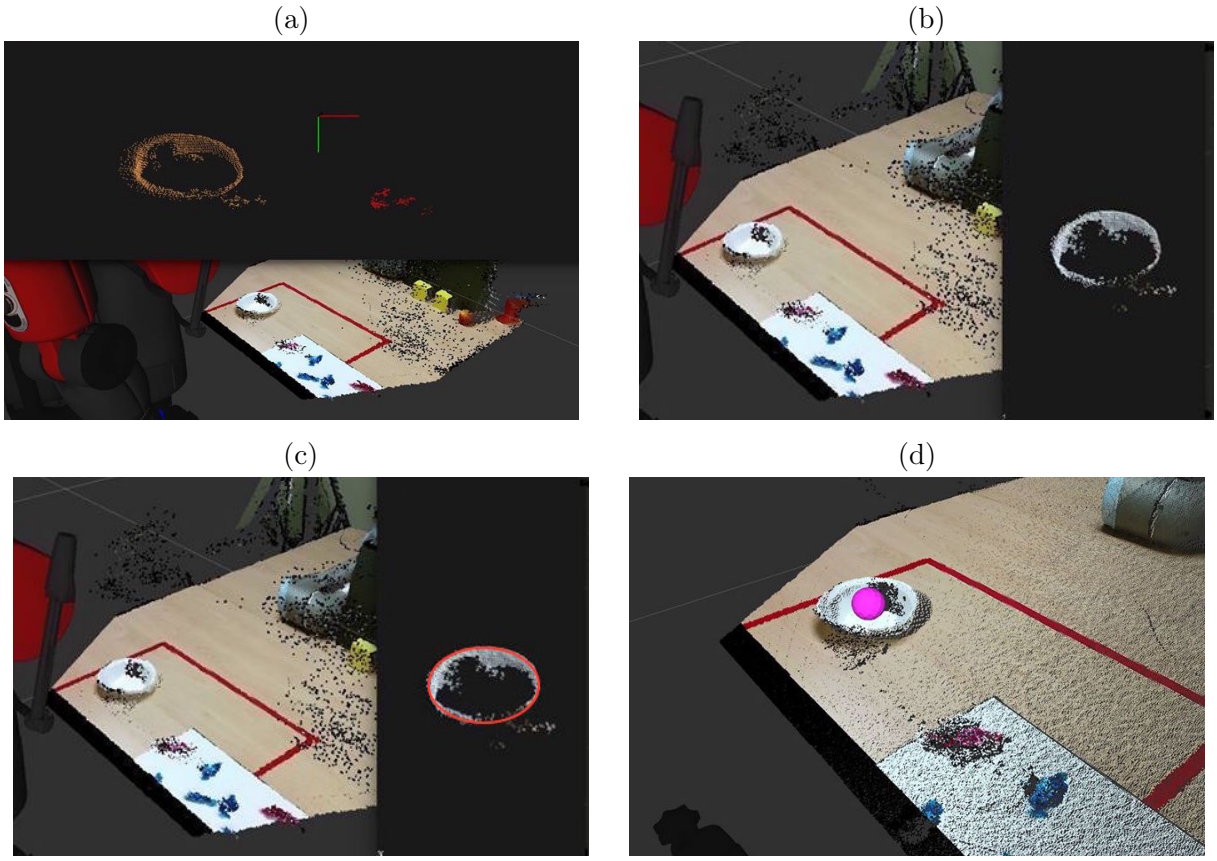


Figure 5.1: Overall bowl recognition process. Each of the images has a view of the processed pointcloud and a view of the actual cloud from the Kinect in RViz. (a) Object clustering in the pointcloud. (b) RGB colour segmentation. (c) 3D circle detection using RANSAC. (d) Visualization of the bowl centre in RViz.

2. Object Clustering - The purpose of this algorithm is to separate the points in the pointcloud into clusters, ones which are close enough to represent the individual objects on the table. The theory behind this algorithm is it takes in the indices and estimated normals of the pointcloud and for each possible region, calculates a K-nearest neighbour search over the indices. During this search, it compares each point with another, checking first for a specified smoothness constraint, then if the deviation between normals is within a specified angle. If that is satisfied, then the difference in curvature is tested, allocating the points to the appropriately separated clusters. This algorithm uses the region growing segmentation implementation located in the PCL ROS package[2]. Whilst using this algorithm, different values were tested. Variables such as number of neighbours, smoothness and curvature constraints were tested until the resulting cloud produced reliably clustered objects. In **Figure 5.1a**, you can see the points representing

[1] Tabletop object detector documentation. `http://wiki.ros.org/tabletop_object_detector`
[2] Region growing segmentation documentation. `http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php`

the bowl have been separated into one cluster and the sweets into another.

3. Colour-Based Segmentation - Whilst testing the bowl detection methods, it became clear that with the Kinect, there were clear noise issues caused by certain objects in the Kinect's view. Baxter's arms especially caused severe black noise to appear around them, making it harder to segment the bowl out from other objects. Therefore, a colour segmentation method was used to separate the white bowl from the interfering black and other coloured clusters. This algorithm was written myself as the PCL library didn't provide a good solution for this method. This was done by looping over the points in each cluster and averaging the RGB values of each one. Then, only the clusters with a high enough R, G and B threshold were used, so that the resulting pointcloud contained only the points for the bowl and any other white object on the table. This method could be expanded to recognise other coloured bowls relatively easily, by finding the correct thresholds for other colours. **Figure 5.1b** shows the white segmented pointcloud with no black points/noise.

4. Detection of the Bowl Rim - The most significant part of the bowl that tended to show up in the Pointcloud was the bowl's rim. From that information, it was decided that the simplest solution was to find the rim by searching for a specific sized circle within a plane of the 3D pointcloud. This method was implemented by using the RANSAC algorithm to detect the points that best fit a circle shape within any particular plane. This method in the PCL library is especially good at allowing variation in it's detection[3]. By varying distance thresholds and minimum/maximum circle radius values, the bowl was found successfully, even with gaps in the rim (which could occur when Baxter's hands went in front of the bowl). The successful rim detection is shown in **Figure 5.1c**. After the circle model had been found in the white Pointcloud, the x, y and z centre coordinate of the bowl's rim was then known in the Kinect's frame coordinates.

5. Averaging and Transformations - After the bowl was found in the Kinect's frame, when publishing in RViz, it was clear whilst the centre point was always within the bowl, it wasn't a reliably constant bowl centre. Therefore, a new ROS node was created that took in the previous result, averaged the centre value over 20 frames and then outputted a new averaged centre point. This point was a lot more stable for Baxter to use. After, using the averaged bowl centre, that point was transformed into Baxter's torso frame and published, so the values could be used for bowl manipulation. To make sure the correct centre of the bowl was detected, it was visualised in RViz, shown in **Figure 5.1d**.

[3] RANSAC model example. `http://docs.pointclouds.org/trunk/group_sample_consensus.html`

Improvements - The basic approach to recognising the bowl is described above. However, as the system began more rigorous testing, when the basic shop system was being put together, the detection was not accurate enough for Baxter to always be able to grip the edge of the bowl. Therefore, a few improvements and changes were added to the system.

Due to increased noise in the Kinect's Pointcloud recordings, the averaging over 20 frames was not reliable enough to keep a fixed centre. To counteract this, a cumulative average was taken instead, meaning that if there was interference, the recognised bowl centre would not be affected as much. Another improvement was to reduce noise at the segmentation stage. To do this, a larger minimum clustering size was used when clustering the objects, so less areas of noise were included in the bowl rim detection. This resulted in a stable centre for the bowl in multiple positions on the table, as shown in the testing below.

5.1.1.2 Testing

To determine whether the recognition was reliable enough, multiple tests were taken for the bowl recognition and noted down in a spreadsheet, found in the Github repository[4]. The main method of testing whether the bowl recognition was accurate enough was devised by getting Baxter to grab the edge of the bowl using the bowl's centre coordinate found through the vision system. Since grasping the bowl is a key aspect in retrieving sweets from the bowl, it made sense that the recognition system would have to be accurate enough for Baxter to be able to reliably repeat this feat.

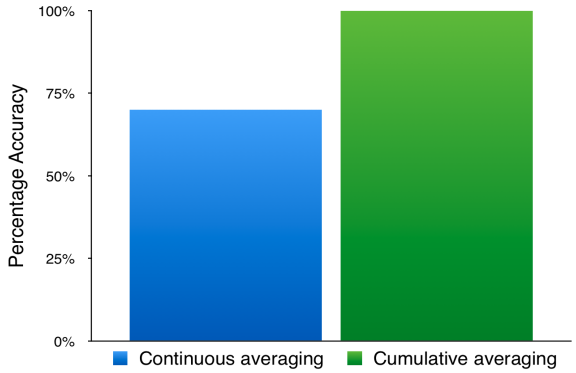


Figure 5.2: Comparison of repeated grasps using different techniques over time.

For the initial rolling average over 20 frames and the cumulative averaging methods (mentioned in the sections above), Baxter was tasked with grasping the edge of the bowl 10 times with each method. The efficiency was then calculated as a percentage of the bowl grasping accuracy, shown above in **Figure 5.2**. As you can see, the cumulative

[4] Github repository: Bowl Recognition Tests. `https://github.com/um10kh/baxter-project/blob/master/Trials/Bowl%20Recognition%20Tests.xlsx`

averaging obtained a 100% accuracy over 10 tests and therefore seemed the more reliable and sensible choice to use in the overall system. The only issue with the cumulative average method is after the bowl has been moved, if the bowl had previously been in place for a long time, the centre will end up being a mixture between the two positions. Therefore, a reset option had to be implemented so that when Baxter moved the bowl, a new cumulative average was taken when looking for the moved bowl.

Limitations

Limitations on the bowl recognition system seemed to be based around common recognition issues within computer vision. The rim detection could get confused if two light-coloured, round-shaped objects were on the table, meaning that in object clutter, the cumulative, detected centre of the bowl would be offset by other, falsely recognised 'bowl rims'. Solutions to this limitation would be complex and would possibly need a 3D model built using the bowl so that Baxter could accurately recognise and distinguish it from other objects. Issues also occurred when Baxter's hand obscured the bowl significantly, preventing the bowl's circular rim from being detected. The simple way to get around this limitation was to only check where the bowl's location was when the system knew Baxter's arm was out of the way.

5.1.2 Manipulation

With the bowl having been recognised, the next step was for Baxter to retrieve sweets from the bowl so he could separate them out on the table. Two main methods for this were trialled, first scooping methods were tested, trying different ways for Baxter to scoop out sweets with his grippers. Secondly, Baxter tipped the bowl onto the table at different angles to get the sweets out. Both methods are explained in more detail here:

5.1.2.1 Scooping Methods

Firstly, several methods of scooping were trialled for Baxter to scoop sweets out of the bowl with his gripper. These methods were developed through theories and trials and then tested to see which methods were the most efficient. Different variables to consider were things like gripper distance - the distance between the grippers when closed, angle of entry to the bowl and gripper movement before grasping. Testing for the reliability of these methods are shown in the **Scooping Tests** section and videos on all of the different scooping methods are located on the Github repository[5].

Grasp from Above - The grasp from above method was the simplest, first approach taken, where Baxter retrieves the centre of the bowl, goes slightly above it at a fixed

[5] Github repository: Videos - Bowl Scooping Methods. `https://github.com/um10kh/baxter-project/tree/master/videos/Bowl%20Scooping%20Methods`

height, then moves slowly down into the bowl and closes his gripper. **Figure 5.3a** shows this grasping method in action. To an extent, this method was one of the best at gripping individual sweets from a full bowl however, the problem was the approach. During this method, Baxter would try and go to a fixed depth and then grab sweets. Unfortunately, without analysing the sweet's orientation in the bowl first, the end of Baxter's grippers would get caught on them before reaching the specified depth. Then the sweets underneath the grippers would be crushed, which was seen as a problem for a shopkeeper.

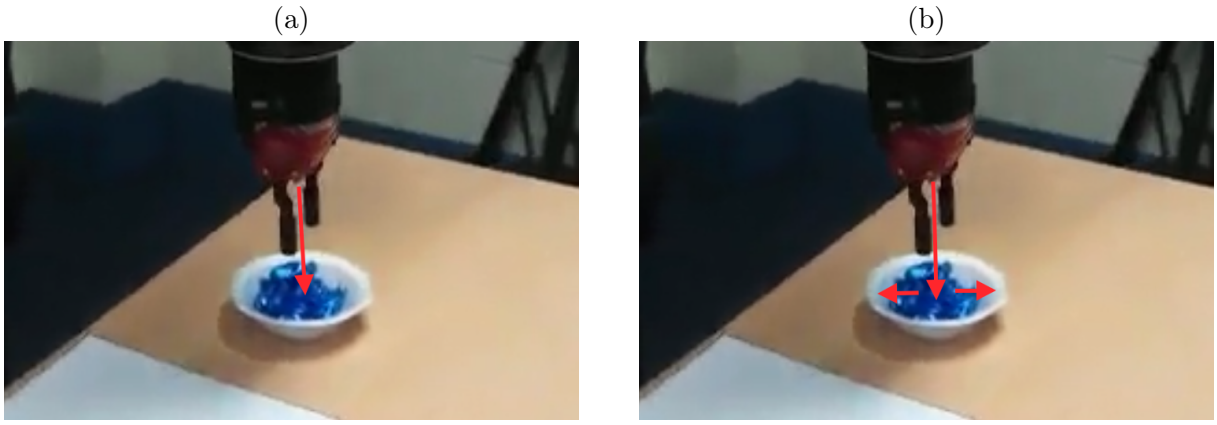


Figure 5.3: Grasping methods: (a) Grasp from above. (b) Grasp and shake

Grasp and Shake - A second version of the grasp from above method used a slightly adapted method of bowl entry, where Baxter went near the rim, then lowered his hand moving from side to side. The basic idea of this grasp is shown in **Figure 5.3b**. In theory, that would reduce the likelihood of hitting and crushing sweets on the way into the bowl as the hand would be able to shake itself in. However, yet again, if the grippers approached on top of some sweets, they would be crushed and then moved from side to side, staying stuck. Another fault of this method came from trying to empty the bowl. Due to the position being approached at the centre, only sweets in the centre could be retrieved, pushing other sweets to the side of the bowl. That spawned the idea for creating a scooping method that tilted the bowl in future trials.

Scoop from the Side - This method worked very much how it sounds, using Baxter's gripper to approach the bowl at an angle to try and scoop some sweets up. **Figure 5.4a** shows a diagram of this method. The downside to this approach came apparent after multiple trials, where Baxter ended up pushing sweets to one side of the bowl so on repeated attempts, it could no longer reach them. This method did however improve Baxter's ability not to crush sweets, as the angled entry meant he was less likely to get stuck on a sweet, as he could just push it to one side. Due to the sweets being pushed aside, the idea of tilting the bowl again was theorised to be able to retrieve more sweets from the bowl over time.

(a) (b)

Figure 5.4: Grasping methods: (a) Scoop from the side. (b) Tilt and scoop.

Tilt and Scoop - This method came by combining ideas from the previous methods to develop a more reliable method of scooping sweets. Firstly the left hand would grip the side of the bowl and tilt it so sweets fall down to one side, then the right hand gripper would come in and try and scoop some sweets up. **Figure 5.4b** shows this grasping method in action. This process, while reliable on tilting the sweets, was unreliable on it's consistency, as the approach sometimes managed to pick up one or two sweets, but most often it missed them on a random approach.

5.1.2.2 Scooping Tests

In the Github repository[6], there is a spreadsheet showing trials of each type of scooping method mentioned above. For each method, there was a trial regarding it's efficiency from grasping sweets from a full bowl (the less grasps Baxter could get the customer's sweets in the better) and repeated trials were carried out, to see how well they performed over time after the initial scoops.

In **Figure 5.5a**, the y axis shows the number of sweets grasped and the x axis shows the current grasp attempt number. The graph shows the tilt and scoop method was the most reliable grasping method over time. This suggested that the tilting was beneficial and meant it prevented sweets from being pushed to one side after initial grasps. The problem with all methods in this figure however, is that sweets were not grasped particularly efficiently. There was a maximum of around 3 sweets retrieved after 8 attempts, meaning the customer would have to wait a very long time if they wanted 3 or more sweets (which ideally would be possible).

[6] Github repository: Grasping Methods Tests. `https://github.com/um10kh/baxter-project/blob/master/Trials/Grasping%20Methods%20Tests.xlsx`

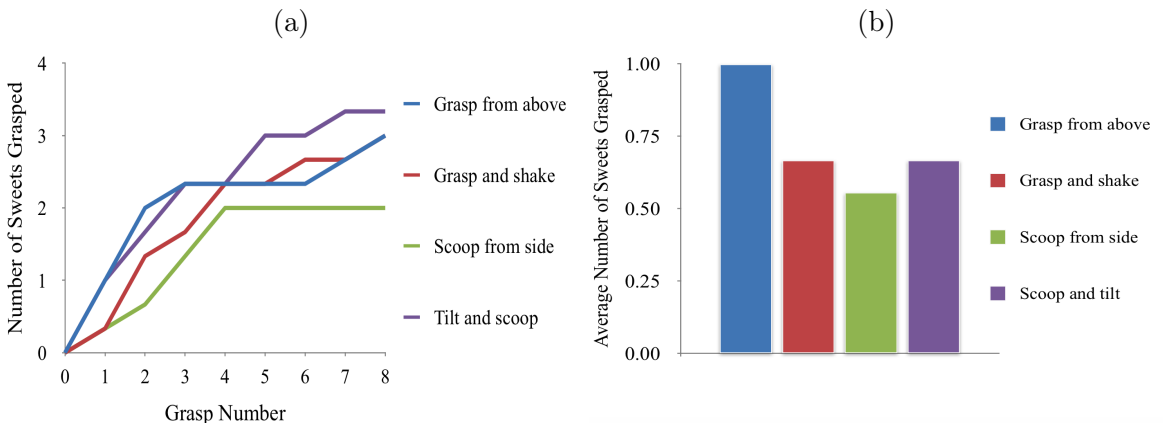


Figure 5.5: Scooping test graphs: (a) Comparison of repeated grasps using different techniques over time. (b) Comparison of the average sweet number grabbed from a full bowl.

In **Figure 5.5b** you can see that, on average, from a full bowl of sweets, the grasp from above method was the most effective, averaging one sweet grasped over ten attempts. The main thing to point out here was that the maximum average grasped from a full bowl was only one sweet. In a shop-like scenario, a customer will most likely want more than one sweet, so if Baxter had to grasp one at a time, the process would take a very long time. After testing these grasping methods, it was determined that no one scooping method would be efficient enough for Baxter to use. This was due to the reliability of these methods not being sufficient enough for Baxter to grab multiple sweets at a time. By taking inspiration from the tilt and scoop method, it was decided that tipping sweets from the bowl onto an area on the table would be a possible solution to get more sweets in a shorter number of attempts. This was trialled in the **next section**.

5.1.2.3 Tipping Methods

After deciding scooping the sweets was not a reliable enough method, some bowl tipping methods were trialled to attempt to achieve a better efficiency. Each method was again developed through manipulation trials. In each trial, the sweets were attempted to be tipped and dropped onto a rectangular area designated for sweet singulation. This brought in several new factors to consider such as whether the sweets landed successfully in the area, how many sweets fell next to each other (the more often that happened the harder for Baxter to separate them later) and where in the area should the sweets be tipped.

Different variables considered in these methods included factors such as tipping height, tipping angle, tipping location and tipping speed. Tipping added another issue to the manipulation because once sweets had been tipped out of the bowl, it was harder to get the remaining ones out, meaning an incremental approach had to be taken, increasing

angles and speed to retrieve them. Testing for the tipping methods are shown in the **Tipping Tests** section and videos of the methods are located on the Github repository[7].

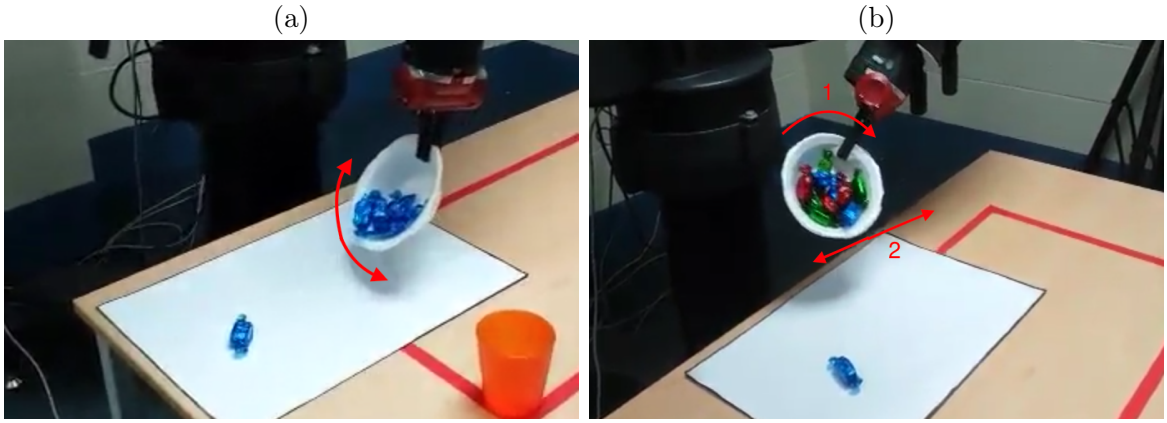


Figure 5.6: Tipping methods: (a) Vertical tilt. (b) Horizontal tilt.

Vertical Tilt - A vertical tilting method was the first tipping method trialled, where Baxter would grasp the bowl from the side, then raise the 'elbow' of his arm so that the bowl was tilted at an angle. Then Baxter would rotate up and down, shaking the sweets out of the bowl onto the table, as shown in **Figure 5.6a**. The optimal number found through testing was for Baxter to tip the sweets out once in the centre-right of the sweet area and once in the centre-left so there was a reasonably even spread of sweets tipped out on both sides of the area. This method caused issues in reliability over multiple tilts. Sometimes, when there were sweets down in the bottom of the bowl, no matter how high the tilt angle was, they were hard to shake out without completely tipping the bowl upside down. The horizontal tilt method was developed from the flaws in this method, in theory to gain some extra control over the tilting angle by using Baxter's gripper rotation instead of the 'elbow' rotation.

Horizontal Tilt - This method was similar to the vertical tilt as mentioned above but instead, Baxter tilts the bowl using his gripper so there was more control over the tipping. Another factor added to this method was that instead of shaking the bowl up and down, Baxter instead shook it in a upward diagonal motion, shaking sweets upwards and out of the bowl at the same time, shown in **Figure 5.6b**. Whilst this technique added a worry of throwing sweets too far out of the bowl, it did help get the sweets towards the bottom of the bowl. The efficacy of this method against the vertical tilt is discussed in the testing below.

[7] Github repository: Videos - Bowl Tilting Methods. `https://github.com/um10kh/baxter-project/tree/master/videos/Bowl%20Tilting%20Methods`

5.1.2.4 Tipping Tests

To test the tilting methods, a similar approach was taken to the bowl grasping tests. In these tests, two different versions of the horizontal and vertical tilts were observed. The 1st iteration of these tilts had a smaller tilting angle and a smaller shaking height and the 2nd iterations had higher values for these. Each of these methods were judged on their ability to tip sweets onto the page from a full bowl and their ability to separate the sweets on the table to be easily graspable.

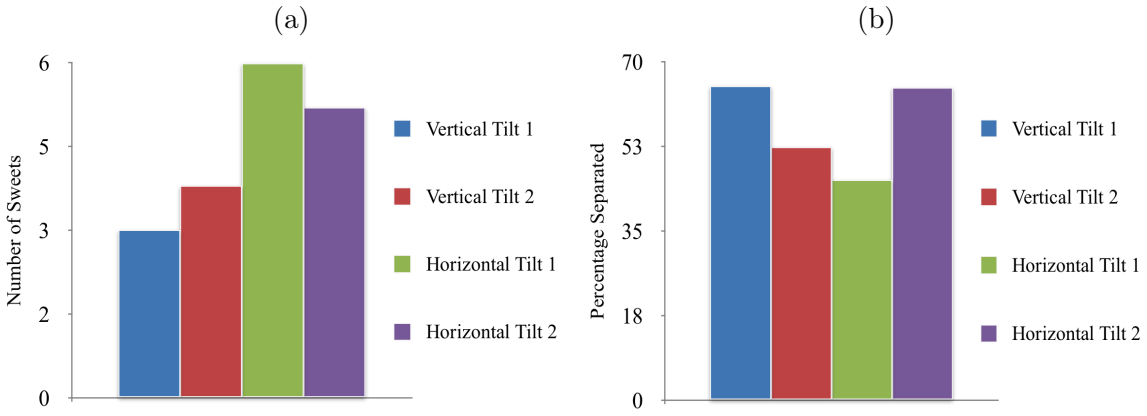


Figure 5.7: Tipping methods graphs: (a) Average number of sweets tipped onto the table. (b) Percentage of sweets separated on the table by each method.

As you can see from **Figure 5.7a**, the horizontal tilting methods tipped more sweets onto the table in one tilt over the vertical tilting methods. By looking at **Figure 5.7b**, the percentage sweets separated was highest using Horizontal Tilt 2 and Vertical Tilt 1. By combining the best results from both tests, it was decided that the second horizontal tilting method would produce the most desirable results for tipping and separating sweets on the table. The recorded spreadsheet for this test is located here[8].

5.2 The Sweets

Once Baxter had seen the bowl and retrieved sweets from it, the next task was for Baxter to recognise the sweets and be able to individually pick the requested sweets up for a customer. This task was split into recognition, singulation and manipulation methods.

5.2.1 Recognition

The task of Baxter recognising the sweets involved him being able to look at an area of the table with sweets retrieved from the bowl on and determine how many sweets there

[8] Github repository: Tipping Methods Tests. `https://github.com/um10kh/baxter-project/blob/master/Trials/Tipping%20Methods%20Tests.xlsx`

were in that area along with what type/colour each sweet was. The problem with recognising the sweets using the Kinect was that the sweets were such small objects, the noise in the Kinect meant a significant number of sweets weren't picked up after segmenting objects from the table. Therefore an alternative method was proposed using OpenCV. The idea behind this method was for Baxter to capture an image of the table with the sweets on using his hand camera. Then from that image, OpenCV image processing techniques could be applied to separate the sweets into individual objects, from which the shapes, centres and colours could be obtained.

5.2.1.1 Sweet Background Detection

The first task in separating the sweets was to have Baxter to only look in the area the sweets were placed on the table, so any other objects/sweets on the table would not interfere. It was decided the easiest way to segment this area out was to use a white piece of paper as the background, making it easier to segment out the rest of the image. The piece of paper used was A3 in size and had a black edge drawn on, to help detect the border of the page. Multiple vision methods were trialled to try and recognise this sweet placement area, explained below.

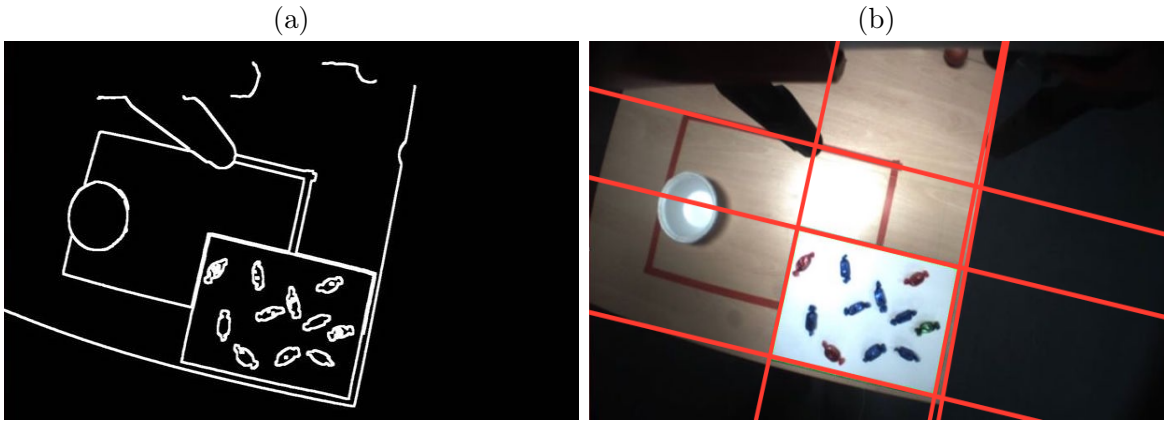


Figure 5.8: Sweet background detection techniques: (a) An example of Gaussian blurring, Canny Edge detection, then image opening. (b) Hough line transform used to detect the edges of the sweet area.

Image Pre-processing - Before the image from the camera could first be used, some pre-processing was taken place to reduce noise and make it easier to detect the area on the table. A Gaussian Blur was performed with a small kernel to firstly blur the image to reduce possible noise. Then, Canny Edge Detection was used to detect edges within the image, along with dilation and erosion to close any incomplete edges, resulting in a reasonably successful edge detection for the entire image, shown in **Figure 5.8a**.

Hough Line Transform - Firstly, to find a rectangle in the image, Hough Line Transform was attempted to be used to find the individual edges of the paper. This

technique was somewhat successful when attempting to distinguish between the edges however, by varying and optimising the line detection threshold, it was difficult to detect all four edges of the paper consistently. Either one or two edges kept being detected then undetected or too many edges were found (shown in **Figure 5.8b**). This resulted in reliability issues finding the rectangular area, so other methods were explored.

(a) (b)

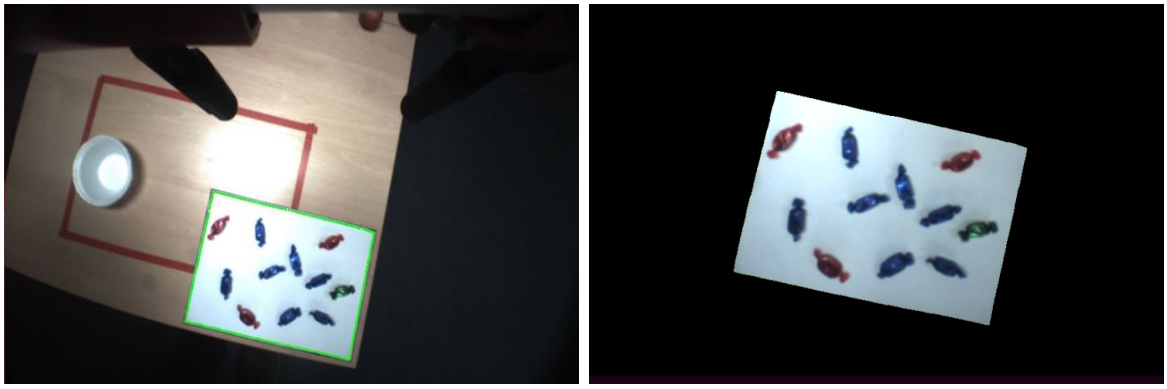


Figure 5.9: Sweet area detection using contours: (a) Green contour correctly wrapped around the sweet area. (b) Masked sweet area after contour segmentation.

Contour Detection - Once the black border on the area was drawn thickly enough to be consistent during canny edge detection, a contour successfully managed to wrap itself round the four edges of the paper. Due to there being many contours detected in the image, certain constraints had to be put on the detection to segment out the sweet area contour from the others. The main method of segmenting out the rectangular area then was to first eliminate the smaller contours by size (using the in-built OpenCV countourArea method) and then approximate a polygon for the countours to detect which countour was rectangular in shape. The eventual correct contour was selected and identified, shown in **Figure 5.9a**. This contour could then produce a mask, which could segment out the rest of the image from the sweet area, shown in **Figure 5.9b**, which meant that the vision system had a separated area to analyse the sweets in.

5.2.1.2 Sweet Recognition

Once the sweet placement area was reliably segmented out from the image, multiple methods were used to attempt to try and identify the individual sweets. These methods are explained below.

Hough Circle Transform

For the simple, round sweets initially used in trials, a Hough Circle detection algorithm seemed like a sensible way to detect the simpler, round sweets. However, like the Hough Line detection used earlier, it was hard to get a constant solution, with circles being

undetected throughout various received image frames (shown in **Figure 5.10a**), therefore other methods needed to be tried to get a more reliable vision system.

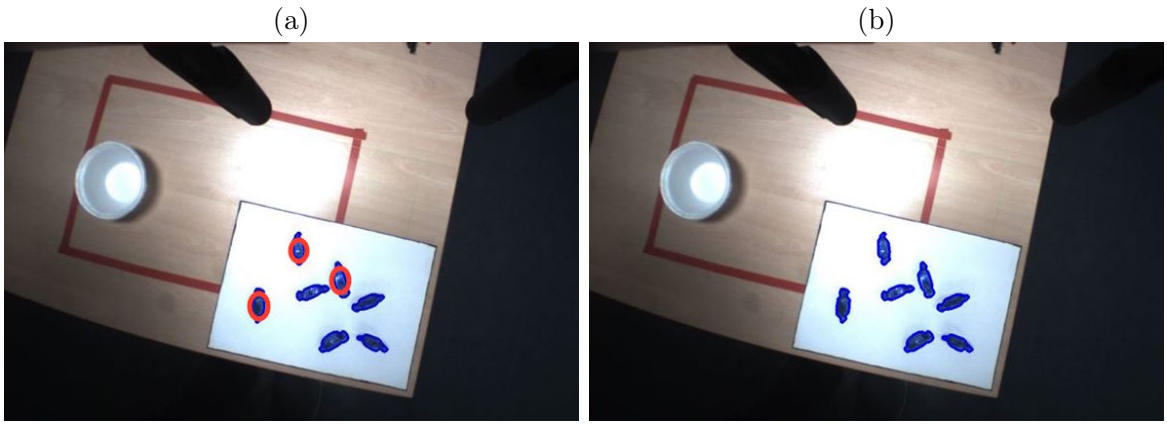


Figure 5.10: Sweet recognition techniques: (a) Sweets detected using Hough circle detection. (b) Correct blue contours found for each of the blue sweets.

Contour Detection

A better method was to do some image processing, Gaussian blurring, closing and opening to reliably produce a mask of a white background with some black sweets in front. The problem with using this method is due to reflections on the sweets wrappers, which caused an issue of multiple broken contours within an individual sweet, whereas a preferred method would have been to capture the whole sweet with one contour, as Baxter needs to be able to count each sweet once.

HSV Colour Segmentation with Contour Detection

A more accurate method was found by using an existing tool called objectfinder[9]. This tool uses multiple sliders to segment an image and produce a mask using lower and upper bounds for RGB values. Since each sweet wrapper had different identifiable colours and they were placed onto a separable white background, using a range of possible HSV values could be used to identify main sweet wrapper colours - blue, green, red etc. An example of this working is shown in **Figure 5.10b** above. The only limitation of this approach was that similar colours under light could be mixed up, for example, red and pink wrappers had overlapping HSV RGB ranges, and therefore could not be both separated and identified by this method. It did however result in very clear contours for the significantly different colours so this method was initially used for the sweet recognition. Possible improvements could have been made with shape recognition then colour analysis, which may have been able to identify a typical sweet shape and then separate by colour after.

[9] Github repository: Misc files - objectfinder.py. `https://github.com/um10kh/baxter-project/blob/master/miscFiles/objectfinder.py`

Improvements on Colour Detection

Whilst the HSV detection method was reasonably accurate, there were issues, which often occur in vision techniques, when the lighting changed. At different times of day, the HSV colour recognition failed, as the blue/green and green/red colourspaces overlapped in light or dark conditions. A couple of solutions were proposed as improvements on the colour detection to rectify this.

Euclidean Distance - Since the HSV range method varied in the light, an attempt was made to capture some of the light variation. Instead of detecting a range of HSV values, an averaged HSV value was calculated using multiple samples of each colour. By analysing the mean colour of the contour multiple times for each colour, an average RGB value for red, blue and green was calculated. Then, when trying to check what colour a sweet was, the RGB value with the smallest Euclidean distance to the average value would be the detected colour.

Neural Networks - Neural networks are a relatively new development in AI, used to learn certain tasks by knowing the needed results, training the network to produce a set of weights to produce the desired results. Since the RGB colourspaces of the green, blue and red sweet wrappers overlapped in their values, it was decided a neural network could take some example values of the sweet's RGB values, learn them, and produce a more reliable colour recognition method.

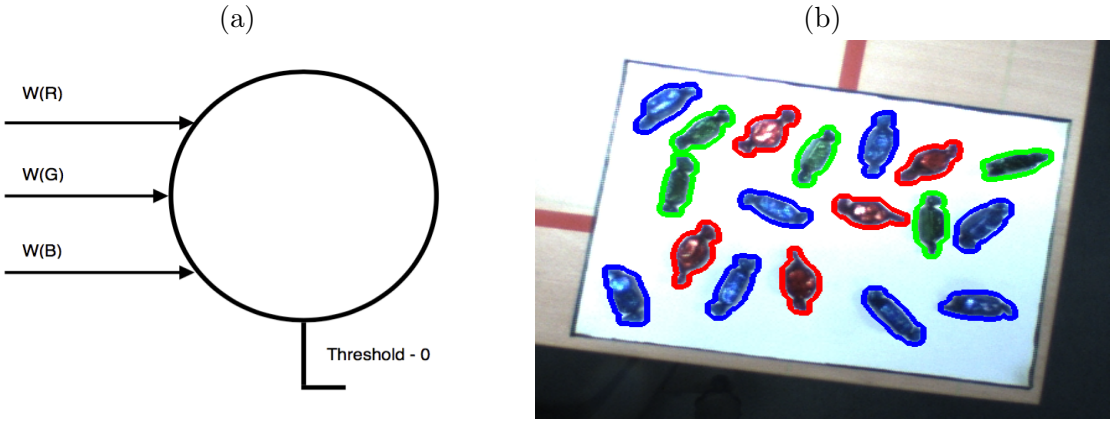


Figure 5.11: Neural network images: (a) An example perceptron of the neural network with a threshold of 0. (b) An example of the correct sweet colour classification using the neural network.

This was done by first collecting a sample of 50 of each coloured sweet's RBG values. 10 images were taken of five green, red and blue sweets respectively with sweets in different positions related to the light source and at different times of day. Then the RGB values from within each sweet contour were written to a text file along with the desired result, for example: "R: 75, G: 80, B: 100, blue". This colour dataset was then used to train a

simple neural network. The simple neural network design was based on the basic perceptron learning algorithm, which used a basic perceptron design, shown in **Figure 5.11a**. More information about the basic principles of neural networks can be found here[14]. This perceptron would ideally train using the dataset by finding four weights that would give a less than zero value if the sweet RGB values matched a colour and above zero if they didn't match. After training the perceptron weights on the dataset for each colour (ie. green vs non green values), three sets of weights were returned to classify the colours. These weights could then be applied to any RGB average value to determine whether a sweet was red, green and blue. In neural networks, there can be some convergence issues, meaning there would be no set of weights to classify all three colours however, seeing as the weights converged without any errors, that meant that red, green and blue sweets could be classified using the neural network with no overlap in classification. The neural network worked very well in classifying the colours, as it took into account variations in light for classification and can be seen working above in **Figure 5.11b**.

5.2.1.3 Testing

After testing multiple colour detection methods, the best method was decided by testing between the three main ones developed: HSV colour segmentation, Euclidean distance and neural networks[10]. The testing consisted of using each of the three methods on the sweet area and seeing which one had the most efficient colour detection. Five red, five green and five blue sweets were laid on the page in ten different orientations and tested with each method and the results are shown below.

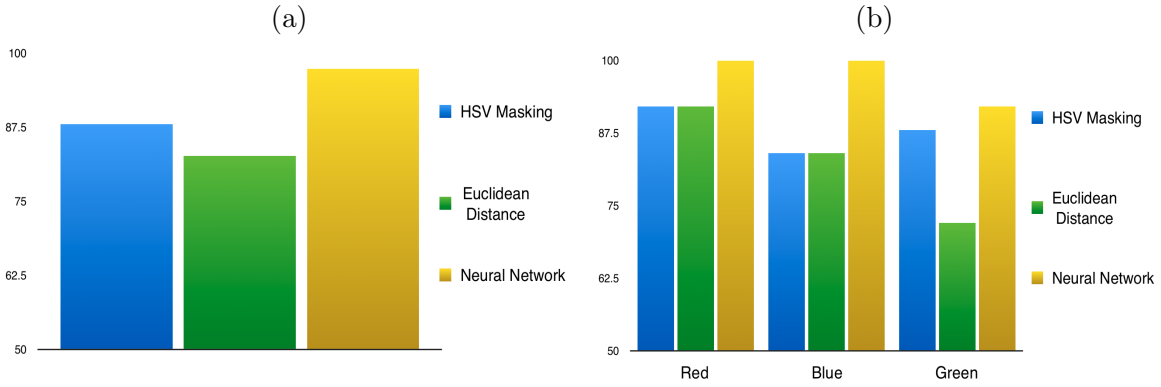


Figure 5.12: Graphs for colour detection methods: (a) Percentage efficiency of colour detection across multiple methods. (b) Percentage efficiency of RGB detection across multiple methods.

As you can see from the RGB tests in **Figure 5.12b**, the green and blue sweets were

[10] Github Repository: Colour Recognition Testing. `https://github.com/um10kh/baxter-project/blob/master/Trials/Color%20Recognition%20Testing.xlsx`

more difficult to identify than the red sweets. This is because there was a larger overlap in their colourspace. As you can see from the correctly identified sweets previously in **Figure 5.11b**, some of the green sweets definitely have a dark blue-ish tint to them, so it is even difficult for a human to recognise the correct colour. However, from using a large sample of data to train the neural network, the neural network does seem to do a reasonable job of differentiating between the two. It is clearly shown that the neural network not only has the largest percentage efficiency of colour recognition overall (**Figure 5.12a**), but it was the most efficient at recognising the different colours as well. This is why it was decided that the neural network was the best colour detection method to use overall.

Limitations

Whilst the neural network did provide an efficient solution to recognising the sweet's colours, the problem was that this technique was only applicable to a specific brand of sweet with unique blue, green and red coloured wrappers. If any other colours or makes of sweet were used, a new training set would have to be created and trained before they could be recognised. A suggestion to solve this would be to create a user interface to allow the user to add their own coloured sweet by taking a sufficient number of photos. The lack of time towards the end of the project meant I did not have time to implement this. Another issue still arose when the colourspaces overlapped too much. For example, the neural network couldn't correctly train itself to tell the difference between pink and red sweets in different lights. To solve this, a more complex network could be developed to allow extra hidden layers for a more robust method for recognising complex colour separations.

5.2.2 Manipulation

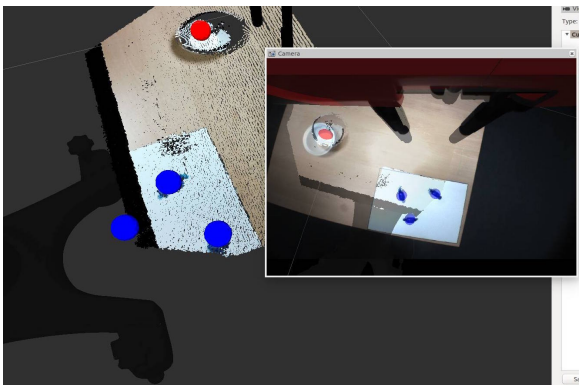


Figure 5.13: RViz showing the transformation between the detected sweet centres and the actual coordinates.

Now there was a method for the sweets to be detected, by extracting moments from the sweet contours, the 2D pixel coordinate could be retrieved from the 2D image. The

problem then was converting the 2D points in that image into 3D world coordinates. This was done using an altered version of a pinhole camera method[11], where the u, v coordinate within the 2D image could be converted to a 3D world coordinate using Baxter's in-built calibrated camera matrix. The equation uses the x and y camera offset values, with the focal lengths to scale the initial point. Then using the distance from the camera to the table (which is fixed), the points were then converted into 3D world coordinates. This was tested using rViz and Baxter to make sure it worked, shown above in **Figure 5.13**. This method was a difficult concept to grasp, but after some sound mathematical theory in deducing the correct formula, I was pleasantly surprised at how efficient the transformation was.

5.2.2.1 Grabbing Methods

Basic Grasp

Once the sweet positions were known to Baxter, a basic grabbing method was developed initially. This was done by creating a function where the xyz coordinate was known for the centre of the sweet, then Baxter would move his hand directly above the sweet at the default horizontal gripper rotation. Then Baxter slowly moves his hand down and closes his gripper when he reaches the height of the table. This was a very simple approach to grasping sweets that, whilst reasonably reliable, when the sweet's orientation was at an awkward angle, the lack of rotation would cause the sweet to slip out of the gripper on grasping.

Grasp with Rotation

(a) (b)

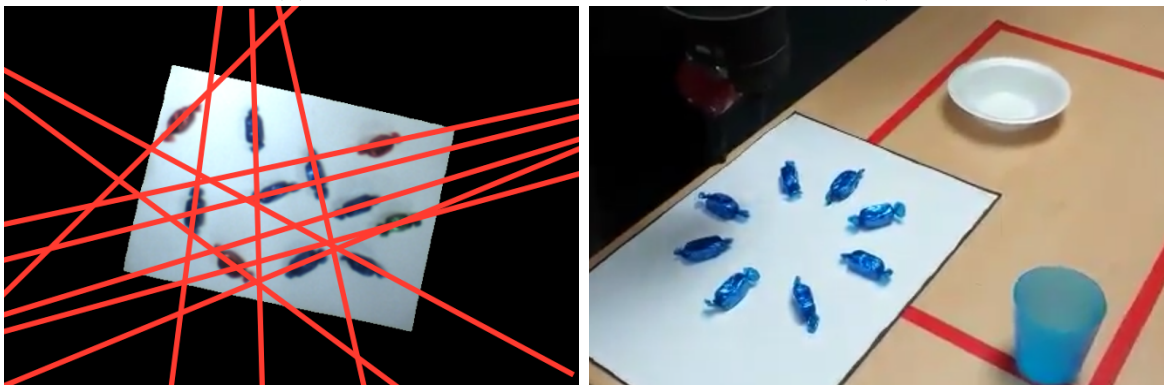


Figure 5.14: Grasping sweets with rotation: (a) Principle axes of each sweet on the page. (b) An image showing Baxter grasping sweets placed at multiple angles.

This grasp used a very similar approach to the previous grasp however, it contained a very important update, it included the rotation of Baxter's gripper. This however,

[11] OpenCV - Camera calibration and 3D Reconstruction. `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`

meant that an improvement on the sweet's vision system was needed to work out the angles the sweets were placed at. To do this, a PCA (principal component analysis), was calculated on the contour for each sweet. After the principal axis was calculated for each sweet (shown in **Figure 5.14a**), the grippers could be placed perpendicular to that axis to grab the sweets at the correct rotation. Therefore, after finding those perpendicular lines, the angle could be calculated that the gripper needed to rotate by using the formula below. This formula was used with the perpendicular line values against a the horizontal axis to find out the angle Baxter needed to grasp at related to it's default rotation along that axis.

$$cos\theta = \frac{(x_2 - x_1) \cdot (x_4 - x_3)}{|x_2 - x_1||x_4 - x_3|}$$

Here, x1 and x2 are the endpoints of the calculated line and x3 and x4 are endpoints of a horizontal line representing the x axis. After calculating the angles, they were sent along with the sweet coordinates so Baxter could know the angle required to pick up each sweet. Once this grasp was working, the pick up efficiency of the sweets became a lot higher and Baxter could grasp sweets at all angles, shown in **Figure 5.14b**. A comparison between the basic and rotation grasping techniques can be seen using the videos on the Github repository[12].

Improvements

Whilst the grasp with rotation method was very efficient at picking up sweets, it did sometimes only grab a small edge of the sweet, then the sweet would slip slowly out of the gripper and fall onto the table. After, Baxter would try and put nothing into the bag, assuming it had grabbed a sweet. The problem with this was Baxter thought he had grabbed a sweet there, whereas ideally, it would've been nice for Baxter to know when he missed a grasp or not. This lead to an alteration using torque-based sweet detection.

Baxter contains a pressure sensor on the inside of each of his grippers. To detect the pressure, by subscribing to the gripper's state topic, the torque could be retrieved from the gripper at any time. This meant that Baxter could tell whether a sweet was in his hand by detecting whether there was zero torque (no sweet) or some torque (sweet currently in the gripper). By using this technique while Baxter was grasping the sweet, he would know whether or not the sweet was in his gripper. Therefore the improvement was made that he would try and regrasp another sweet if he missed grasping a sweet rather than continuing to put nothing into the sweet container.

[12] Github repository: Videos - Sweet Grasping Methods. `https://github.com/um10kh/baxter-project/tree/master/videos/Sweet%20Grasping%20Methods`

5.2.2.2 Testing

To test the grasping methods, they were both measured trying to grasp 10 sweets at various angles from the table (tests found here[13]). To make the tests fair, the sweets were placed in the same orientation for each method. As you can see from **Figure 5.15**, the variables that were recorded were whether Baxter missed grabbing a sweet completely, whether he partially grasped but dropped the sweet or whether he correctly grabbed the sweet.

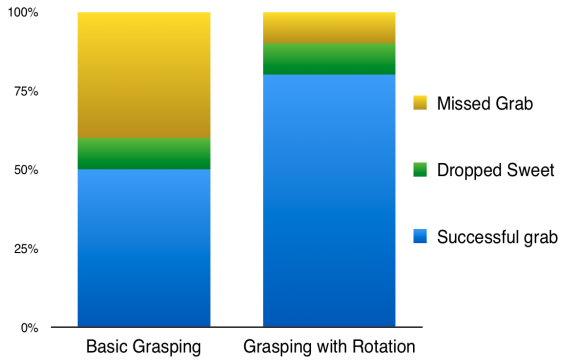


Figure 5.15: A graph showing the difference in efficiency between the developed sweet grasping techniques.

The graph shows that the grasping with rotation severely reduced the occurrences where Baxter missed the sweets. This showed that adding rotation to the grasping method really increased the efficiency and it was a fairly reliable method to use in the overall system. Some limitations of the approach could be that the approach still misses/drops the occasional sweet. If there was time, the grasping method could be further improved for Baxter to iteratively move towards the sweet and use the vision system to analyse the exact position between the gripper and the target, to make sure it stayed exactly in the centre of the gripper.

5.2.3 Singulation

A major problem with the tipping sweets from the bowl is that it could result in sweets which are too close together to be recognised properly by Baxter's perception system. Without them being properly recognised, Baxter ignores them and therefore can't include them in the overall system. This results in problems say, if the customer wants three red sweets and there are only four in the bowl. If two red sweets are overlapping when tipped onto the table, Baxter could only possibly recognise the other two. This section explains methods used to recognise overlapping sweets and methods to separate them/pick them up from a pile to rectify this problem.

[13] Github repository: Sweet Grasping Tests. `https://github.com/um10kh/baxter-project/blob/master/Trials/Sweet%20Grasping%20Tests.xlsx`

5.2.3.1 Detecting Overlaps

Since the previous sweet recognition system only recognised sweets which were separated from each other on the table via a contour method, large contours of two or more sweets could not be easily recognised. The main problem here was that, due to the sweets not being separable by colour reliably (the colour spaces overlapped), there needed to be a custom vision method developed to separate them. This section discusses the multiple approaches on trying to analyse these larger contours and split them into the respective separate sweets.

Separation by Case

An initial attempt to split these contours into separate sweets was to code the contour splitting manually by analysing the position, shape and angle of the contours and splitting them the correct way. This method was attempted by first looking at the shape of the contours, the area of them and the ratio between the height and width.

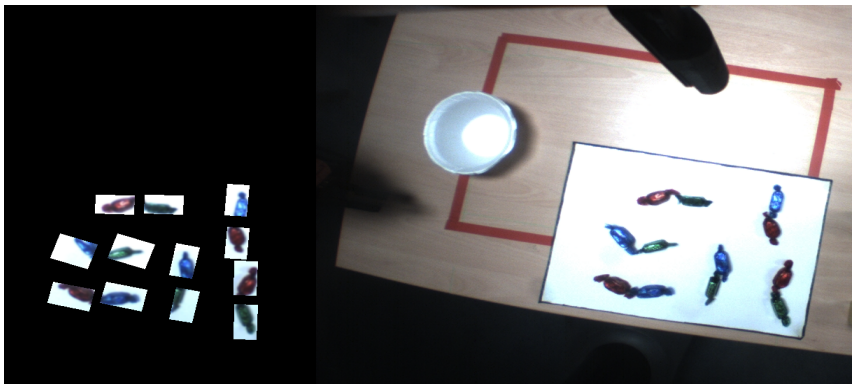


Figure 5.16: An example of splitting contours using a simple case.

Initially, the first case attempted to solve was the one in **Figure 5.16**, where the sweet contour had a rectangular shape with an area the size of two sweets. As you can see in the image, this case was easily separable and therefore the sweets were recognisable. However, the problem came when attempting to apply this technique to multiple cases. The number of different separable cases were too large and varied to use this technique in practice, so other techniques had to be developed.

Morphological Analysis

Another attempt to do this was to try and split the sweet contours via morphological analysis. This was done by a few processing methods to dilate and erode the masked sweet images to split them apart into separate elliptical shapes. This approach worked for the most part aside from when two sweets fell next to each other, sharing one of their sides together.

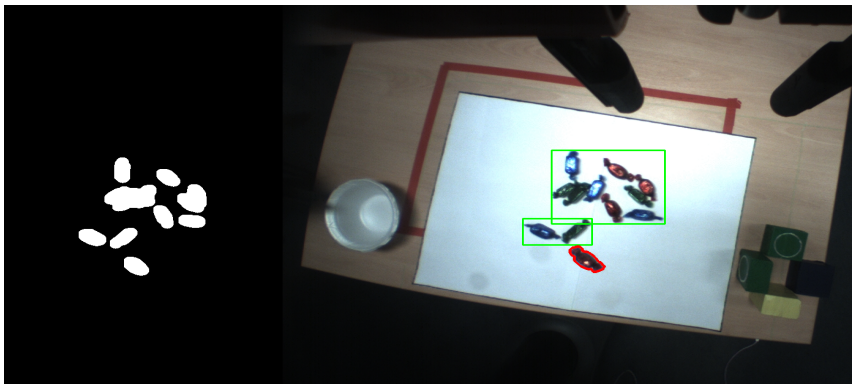


Figure 5.17: An example of a failed morphological analysis on groups of sweets.

This situation could not be split using this method as the sweets were too close to each other to split via this method. An example of this method failing is shown above in **Figure 5.17**, where some of the sweets that were very close together couldn't be separated.

Rectangular Search Method

The main method used in the final version of the software was the rectangular search method. This method used a combination of vision techniques to process the sweets and separate them within the grouped contour. This process has multiple steps, explained below.

(a) (b)

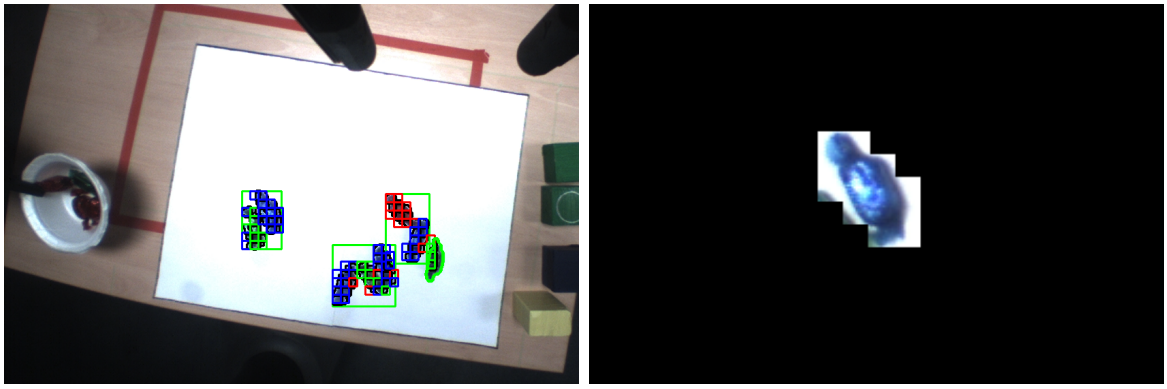


Figure 5.18: Steps for the rectangular search method: (a) Square coloured areas detected within groups of close sweets, (b) A separated sweet mask using the detected blue squares.

1. Splitting into coloured squares - Firstly, an algorithm splits the whole contour up into small squares and looks at the colour of each square using the neural network developed in the sweet recognition system. Due to the fact the green, blue and red colourspace has some overlaps in the RGB values, these square colours were not always accurate, but this method was accurate enough to recognise the majority of the sweet's correct colour values (shown in **Figure 5.18a**).

2. Region Growing Algorithm - Secondly, a custom region growing algorithm was developed to loop over the coloured squares to grow out the regions. This algorithm checked the horizontal, vertical and diagonal neighbours of each colour and determined the colour was correct if it had two or more of the same neighbours of the same colour. This meant that any anomalous colour recognition, like the anomalous, incorrect colours above, were not included in any further analysis.

3. Masking and Recognition - After the region growing, the individual sweet areas were converted to a mask, which could then be detected as a whole sweet using the regular image processing techniques normally used in the existing vision system, shown in **Figure 5.18b**.

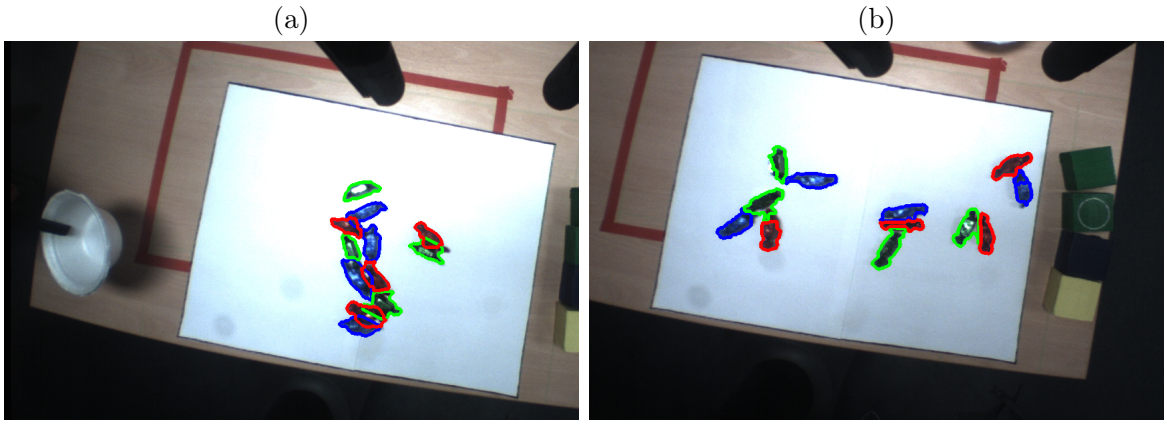


Figure 5.19: Detection of overlapping sweets: (a) Sweets detected using convex hulls to expand the detection area. (b) A working example of the vision system detecting overlapping sweets.

4. Convex Hull/Non Convex Hull Recognition - Occasionally, the sweet mask missed some of the edges of the sweets, due to the colour edges not being properly recognised. In this case, convex hulls could be taken of the mask, to slightly expand the mask allowing for the whole sweet to be detected. However, convex hulls did have a downside, overlapping the detected sweets as shown in **Figure 5.19a**. It tended to be dependent on the scenario whether the convex hull recognition or non-convex hull recognition was more efficient, so the non-convex hull method was used for the time being, shown in **Figure 5.19b**.

Limitations/Improvements

Whilst this approach was reasonably efficient, there were some colour detection issues in the earlier stages, leading to slightly inaccurate sweet edge detection. Another limitation on this approach was the speed of this system. This process required multiple steps, multiple loops over areas and the large amounts of time to process and with more time,

the process could be more streamlined.

This process did have issues when sweets of the same colour were next to each other, as they would be recognised as the same contour/sweet. A quick solution was used for this by using a slightly modified version of the separation by case method mentioned before to separate these sweets, provided that there were only two of the same colour sweet touching each other. Since there were only 4 of each colour sweet in the bowl, it was found that three of the same colour sweet touching on the page was not a very common case.

5.2.3.2 Singulation Methods

Once the overlapping sweets had been recognised, Baxter needed a much more accurate manipulation/singulation system to be able to grasp these sweets. When trying to manipulate sweets in a pile, multiple issues were introduced over picking up a singular sweet, such as grabbing two sweets accidentally instead of one or getting the gripper stuck on a nearby sweet, meaning the grippers couldn't reach the table level and close properly.

Gripper Vision System

Using the detected sweets from the vision system, an approach was proposed where two circles representing the ends of Baxter's electric gripper would be placed either side of a sweet. Then, if those circles didn't collide with a contour of another sweet, then the gripper would in theory be able to go to that location and pick up the sweet. This approach was implemented by doing PCA, similar to the sweet manipulation detection previously. After getting the perpendicular axis to the sweet, the circles representing the ends of the gripper were placed along that axis either side of the sweet as shown in **Figure 5.20**.

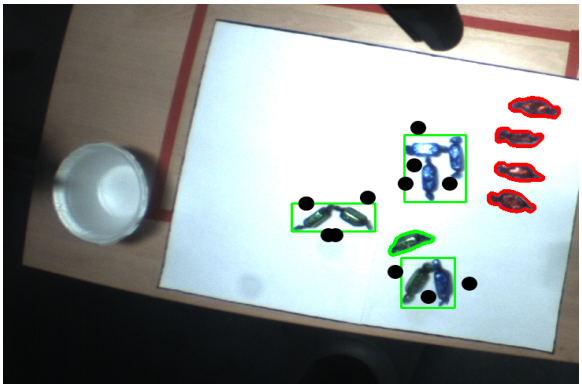


Figure 5.20: An image showing the ends of the gripper projected onto the image where it was possible for Baxter to pick them up.

Then the algorithm checked for collisions of the gripper. If the points for the gripper collided with another sweet contour, then the gripper position would be moved up/down the sweet incrementally to check for an open position. If there was no possible grasping position for the gripper then that sweet was ignored until the pile had been further separated. The idea then was that the vision system would send the respective angles of the sweets that could be separated, Baxter would separate them then re-view the pile to try and grab the other sweets in the next sorted pile. Whilst the theory behind this idea was solid, it was a lot harder to implement the manipulation side of this algorithm (setting custom gripper distances and precise positions caused a lot of issues), so this singulation algorithm was only partially implemented by the end of the project, therefore, definitive testing could not be carried out on this technique.

5.3 Human Interaction

After the main sweet manipulation and recognition methods were initially developed, the final system to implement was human interaction one. Since the human interaction methods were the least important aspect of the overall system (Baxter could get sweets via the command line without any interaction), the human interactions were the least developed at the end of the project, due to time constraints.

5.3.1 Voice Recognition

Voice recognition was a feature that seemed key to implement.within the system. The idea of this feature was for a customer be able to approach Baxter, say a voice command for the requested sweets and then Baxter would get those sweets for them. A couple of approaches were taken to voice recognition, to get the valid accuracy required for an order. The idea was for program to eventually recognise a complex command such as "I would like two green sweets, three blue sweets and one red sweet" and parse the command appropriately.

5.3.1.1 Python's NLTK

Firstly, Python's NLTK module was proposed to be used to analyse voice commands. An in-department microphone was used to record voice commands using some in-built Python microphone recognition/recording modules (the linux alsaaudio module amongst others). This method proved to be difficult to set up and depended on the linux machine and the currently installed drivers. When the microphone installation was set up, a ROS node was created so that the microphone first averaged the background noise to cancel it out. Then the microphone would record until a timed period of speech occurred.

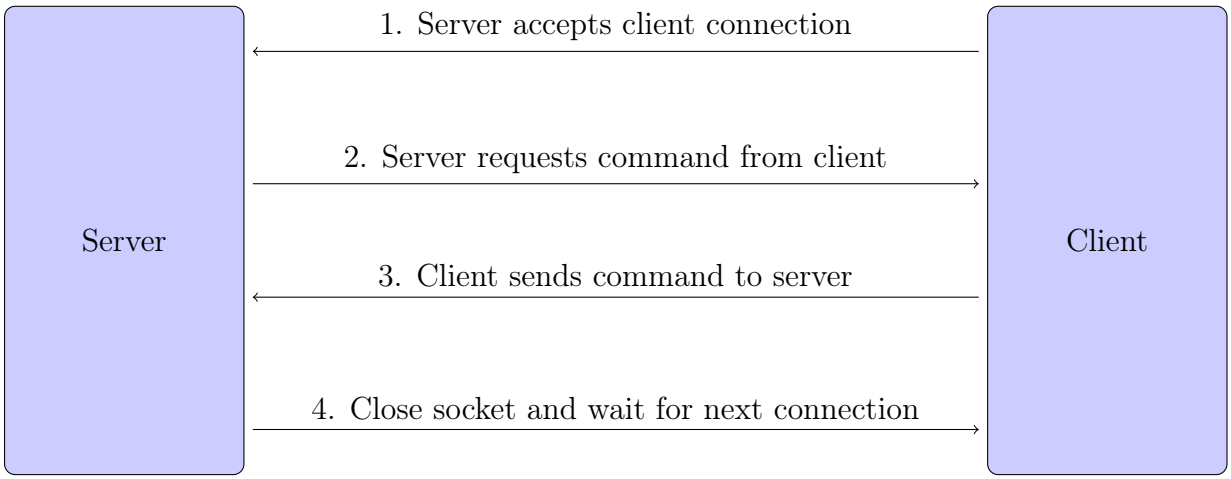
After the recording was made, the Google API was queried with the audio file, to

convert the speech to a line of text. Unfortunately, with noise issues and microphone efficiency issues, there was some difficulty in recognising some words due to the person needing to be a very specific distance from the microphone (otherwise varying noise levels from different distances would interfere with the voice to speech detection). After the speech had been converted to text, the text was analysed to determine the person's command. Since the voice recordings weren't too reliable, the text could be parsed for numbers within the command. The Python module therefore listened three times - for a number for blue, red and green sweets respectively. However, since this didn't recognise full sentences very well, it was decided an Android device could be used to provide a better recording device for more complex, longer commands.

5.3.1.2 Android's Google Voice Recognition

Due to complications with the microphones detecting longer commands, it was decided that an Android device with built-in voice recognition could be a more accurate approach. The only problems with the Android approach is the fact that it assumes an Android device would be available to use alongside Baxter and a small server-client application would have to be developed to run this. Here are some details for the development of this application.

Server-Client Application



The easiest way to connect an Android app with the computer running Baxter's software was to connect a ROS node written in Python to a Java by means of a server-client architecture. The idea being that the Python node would run a server via a Python socket on the university's wireless network and then when a user needs to provide a command, the server would send a command to the client prompting the user to speak or enter a touch command.

The application client was set up so that it only asks the user for a sweet command when the Python server prompts the client for it. The Python server uses an open '0.0.0.0' IP address on the machine running it and then the client uses the specified wireless IP for the server machine. This is inputted into the app on first opening to ensure that the correct IP address is being used for socket connections between the two devices. The server-client logic is shown above.

User Interface - The user interface was designed to be simply understood by the user and only require a small amount of setup for the person running the software. The idea of the app is that it would connect to the server on opening, show the main menu and then when Baxter wants a command, the server would send a request to the Java app client, the app would open up the command page, which would prompt the user to record a voice request or enter the sweet numbers they want via touch arrows. The main menu and sweet command page are shown in the images below.

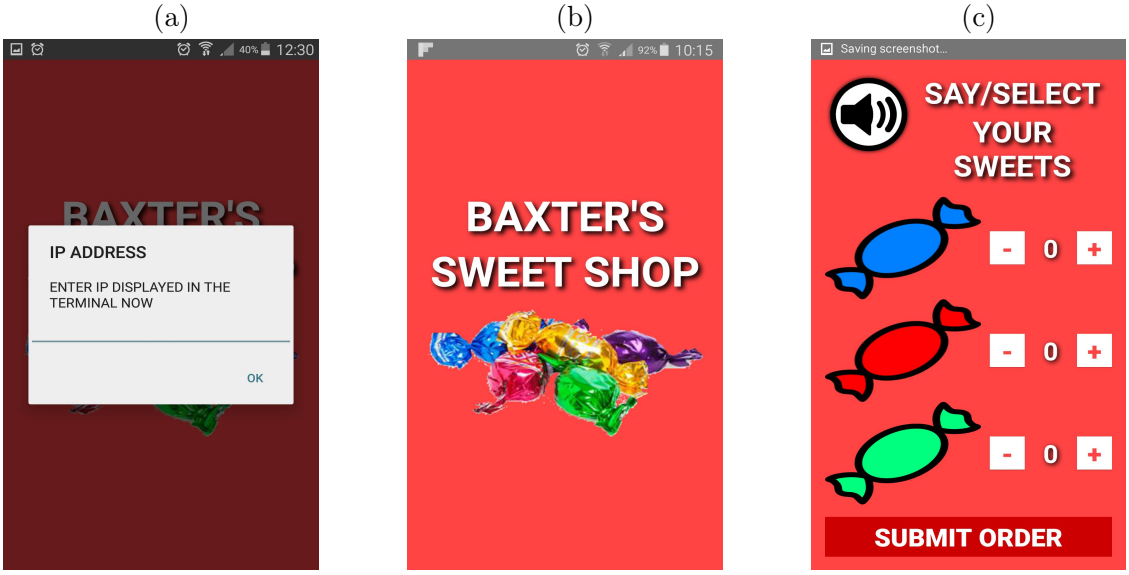


Figure 5.21: Multiple application pages: (a) Enter IP page. (b) Main menu page. (c) Choose sweets page.

When the app was opened, the IP address is entered from the command line printed by the server node, entered in **Figure 5.21a**. Then the main menu page (**Figure 5.21b**) is viewed until the server wants to receive a customer's command, which would go on to the command page (**Figure 5.21c**).

Analyse Command - After the speech had been converted to text using the Google voice recognition API on the device, the text needed to be parsed for a command. The parsing works in a relatively simple way, where it first looks for whether someone has said the words 'blue', 'green' or 'red'. There is also a basic fuzzy search method where it also looks for other words that sound very similar to them. After, the locations of those

words are found and then the words before the colours are analysed, to see what number they are. If they are recognised as 'one', 'two', or 'three', then the parser knows how many of each colour the customer wants. This approach works reliably enough and the more people that tested the system, the more changes and extra anomalies were caught in the voice recognition.

5.3.1.3 Testing

To test the voice recognition approaches using the Python NLTK module and the Android device's voice recognition, firstly, each approach was set up and ten commands were spoken. Each test involved a complex command, which contained a number for blue, red and green sweets for example 'Can I have three green, three blue and two red sweets'. The request numbers were varied from one to three and then it was recorded whether all three of the numbers were correctly detected, or whether some were incorrectly recognised.

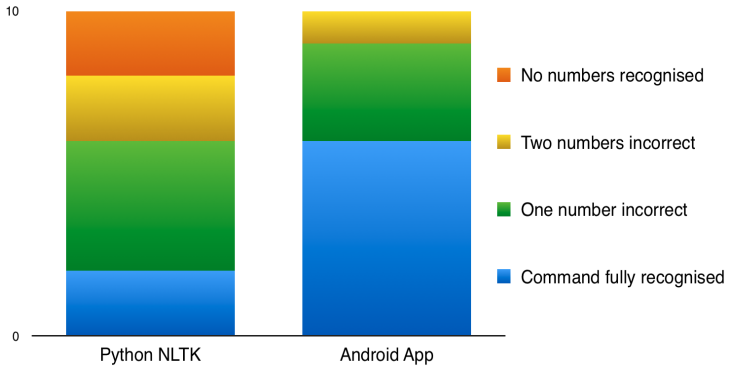


Figure 5.22: Voice recognition tests testing the efficiency of the Python NLTK module against the Android voice recognition.

As you can see from **Figure 5.22**, the android application fully recognised or had one number incorrectly recognised 90% of the time, as opposed to the python NLTK module, which was far less accurate. This seemed to be due to the noise issues with using a USB microphone rather than a phone microphone. It seemed that varying distance between the customer and a normal microphone seemed to cause a lot more issues with recognition than varying the distance from a phone with the app installed. Therefore, the obvious choice for voice recognition for the overall system was to use the android device.

5.3.2 Customer Recognition

The next implementation of human interaction was to recognise whether a customer has approached Baxter or not. Then if the customer had approached Baxter for some sweets, Baxter would know and could therefore start the conversation and other aspects

of the interaction. The initial idea for the approach was to know when a customer enters and exits the scene, to ideally be further expanded to recognise certain actions, like reaching for the sweet bag and exchanging money.

5.3.2.1 Detect Person

The first method to detect a person was a very simply devised method, based around background subtraction without any shape recognition on the person whatsoever. Firstly, Baxter would request a ROS node to look for a person. Then, Baxter would move his arms out of the way so the head could turn and see a customer enter. The current frame would then be used as the background frame for subtraction. A person would be considered to have entered the scene when a large contour was found entering the scene, seen in **Figure 5.23b**. Due to lots of dilation and other pre-processing methods being performed on the image, to make sure the person moving in the scene was recognised as a filled shape, you can see there was some accuracy issues with the contour. However, it was decided that it worked well enough for the basic purpose needed.

Figure 5.23: Person detection methods: (a) Masked person contour differencing from background. (b) Detected person contour in image.

After the person was found, a key part of the recognition would be to make sure the person was in front of Baxter for a set period of time. After the person existed in front of the camera for 30 frames, the node recognised the person as wanting sweets and sent a message back to Baxter to start the interaction. However, if a person did not exist for 30 frames, the program would recognise the contour exiting the scene and therefore carry on waiting for another customer.

This system also had to recognise the opposite feature, when a person left the frame. By keeping the initial background subtracted frame it was possible to find the person contour until it became too small and left off one side of the camera. Then, once the person's contour was not detected anymore, a boolean variable was triggered that

allowed Baxter to know that the person had moved away from the camera. This was useful in the process of waiting for one customer leave and another to approach in the shop scenario.

5.3.2.2 Skeleton Recognition

Due to time constraints, a working implementation of skeleton recognition could not be implemented by the end of the project. This was partially due to technological issues. To get a skeleton recognition system working, the system would need to implement a second Kinect depth camera. The problem with a second Kinect is that it was difficult to find a place to put the Kinect, due to one already being on the torso of Baxter looking at the table. This meant the only place to put a second Kinect was on a tripod away from Baxter, meaning that if Baxter was moved at all on open day, that Kinect would have to be recalibrated each time. The decision to not implement skeleton recognition was then made from a time perspective, as it takes a long time to set up an extra Kinect and calibrate it to work with the system. Due to only having 4 weeks left at the point of discussing this, it was decided it was better to focus on other key features of the project.

5.3.3 Human Characteristics

A section that became more important towards the end of the project was adding human-like characteristics to Baxter. With most of the system's features implemented and the human interaction not engaging the customer much, features like giving Baxter a face and a voice added a lot to the overall product.

5.3.3.1 Speech System

For Baxter's speech, a TTS (text-to-speech) module was incorporated to allow Baxter to speak commands and react to certain actions made within the shop. The TTS system used here was the open source module MaryTTS[14]. The basic concept of this system is that a Java HTTP server would be hosted, which could convert a client's text request to an output audio file. Whilst there was not an immediately obvious way to translate a Java server/client TTS system to ROS, it helped that a module called *strands_ui* [15] existed, which included user interfaces for robots. After installing strands_ui onto the ros workspace, the server could then be started. Then, a ROS bridge connection was made, allowing a call to me made from the main shopkeeper node to the 'ros_mary_tts' node, which would then post a HTTP request to the TTS server, get an audio response and send that audio to Baxter.

The TTS system came in handy from translating the shopkeeper program from a mute

[14] The MARY Text-to-Speech System. `http://mary.dfki.de/index.html`
[15] Strands_ui Github repository. `https://github.com/strands-project/strands_ui`

terminal-based piece of software to a genuine conversation between Baxter and the customer. It helped when Baxter needed to instruct the customer, such as when he needed the customer to speak to place their order, or even just to voice his frustration when he dropped a sweet onto the table.

5.3.3.2 Facial Expressions

Whilst only a small addition to the overall project, different faces were used for Baxter to show different emotions at different interactions in the shop. Multiple images[16] were used to represent a few facial expressions for Baxter. These were mainly to help accentuate the different features of the system and helped in testing to identify what was going on when away from the computer monitor in front of Baxter. A change from a bored default face to a happy face would occur when a customer went in/out of Baxter's sight. A frustrated face would occur when Baxter accidentally dropped a sweet and a happy face would happen at the end of the successful interaction. Small touches like these faces helped to further the human characteristics of Baxter in this interaction, along with adding a little humour as well.

[16] Github repository: Baxter's Facial Expressions. `https://github.com/um10kh/baxter-project/tree/master/src/manipulation/src/images`

Chapter 6

The Integrated System

Throughout the methods being developed and tested, they had to be added and integrated together into an overall system. This chapter describes the process taken to integrate all of the features together by using service communications, roslaunch files and basic logic to create the final product.

6.1 Feature Integration

After the features had been developed from the Methods section, a major challenge was to integrate all of these features to work together in the shop environment. This was done by first moving the code from all the manipulation tasks into one file[1] to allow the shopkeeper to be able to do all the manual tasks required. These manual tasks were commented clearly and placed under the *Shopkeeper* class.

Once, Baxter was able to carry out the manual tasks such as tilting the bowl and grabbing sweets, the next task was to integrate a communications system (using the *Communications* class), to connect the main Shopkeeper node with the vision/other nodes. The main method of doing this was with custom service messages, where the Shopkeeper node would launch a service, sending a custom message, using a .srv file structure. The Shopkeeper node would then wait for an appropriate response returned from another node, before retrieving the information and using it. An example of this would be when Baxter was waiting for a customer to appear. He would look where he expected a customer then send the *find_person* node a message to find the next customer. That node would then wait for a customer to approach Baxter and wait in front of the camera, before sending a confirmation message back to the main node. This principle was used on the other aspects of the system too, such as finding the bowl, finding the sweets on the table and receiving voice commands from the Android application.

6.1.1 Roslaunch Files

Once all of the code and communications had been integrated into one file, a roslaunch file[2] was built to help make the setup of the software more simple. Without a roslaunch

[1] Github repository: Manipulation - shopkeeper.py. `https://github.com/um10kh/baxter-project/blob/master/src/manipulation/src/robot_shopkeeper_nk.py`

[2] Github repository - Launch - shopkeeper.launch. `https://github.com/um10kh/baxter-project/blob/master/src/manipulation/launch/shopkeeper.launch`

file, the system had to be run by starting specific nodes at specific times, with around 9 different terminals running at the same time. The idea of this file was for there to only be one command to launch them all at the appropriate time, for ease of setup and testing. The basic approach would first launch all of the main Baxter-related nodes such as tucking his arms (so they always start in a fixed position) and opening the right hand camera and head camera with their correct resolutions. Then, the main vision systems were started, looking for the sweet area, the bowl on the table and the customer entry. After that, the application server would be started on the node and the application on the device, meaning the main shopkeeper node could then be run.

6.1.2 System Logic

The roslaunch file meant that all the nodes could run at once however, to make the manipulation and vision tasks fully integrated, some system logic was developed for Baxter to be able to constantly run these nodes, looking for new customers and running the 'shop' more intelligently.

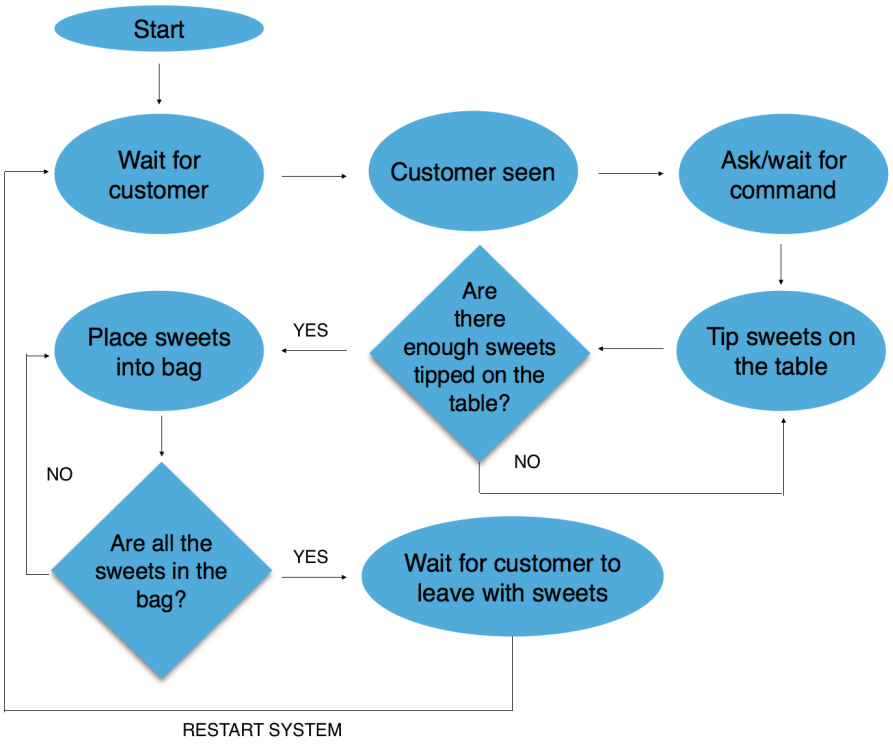


Figure 6.1: A flow chart showing the overall system logic used to build the integrated system.

The key way to implement the logic was to develop the code around the major decision points. The basic way to do this was to implement while loops with logic checks. For example, a while loop would start, checking the sweets on the table using the sweet vision system. Then if there weren't enough sweets, Baxter would keep looping the code, tipping the bowl continually onto the table until there were enough sweets, which

would break out of the loop. This principle was used for all the major decision points in the logic system and a 'while True' loop was placed round the whole system so once a customer had left, the whole system would start again.

6.1.3 User Setup

Once the logic was implemented and the roslaunch file was written, setup options were added to the start of the program to make it easier for the user to start the system. Since there was no bag/sweet container recognition developed, I implemented a setup function that would prompt the user to place the right arm over the sweet bag and press enter, saving the joint positions so Baxter knew where to drop the sweets. Other user prompts for user setup were made in the same way. The user was prompted to open the android app and enter the shared IP address from the terminal and to place the gripper onto the table to retrieve the table's z height.

An issue that arose towards the end of the system was when Baxter tried to grab some sweets, the inverse kinematics system caused his arm to knock off and break the clamp/stand for the Kinect attached to his torso. To get by this, a roslaunch file was created to ignore the bowl detection nodes and the user setup prompted the user to move the bowl underneath Baxter's gripper. This helped to test the overall system without having to use the Kinect. The user setup README[3] explains how the user could setup the system with or without the use of a Kinect.

6.2 Integrated System Images

Once the integration had been implemented along with the user setup, it was then time to test the final system. Firstly, a video taken to demonstrate the capabilities of the system. Stills from this video[4] are provided below in **Figure 6.2** and show from (a) to (i) how a generic customer-shopkeeper transaction takes place with Baxter.

In this particular scenario, the customer approaches Baxter and when prompted, asks for one blue, one red and one green sweet. On his first attempt, he tips out the sweets, recognises them and grasps them from the small pile on the left of the page.

[3] Github repository: User README. `https://github.com/um10kh/baxter-project/blob/master/README.md`

[4] Github repository - Complete System Video. `https://github.com/um10kh/baxter-project/blob/master/videos/IntegratedSystemCompressed.mp4`

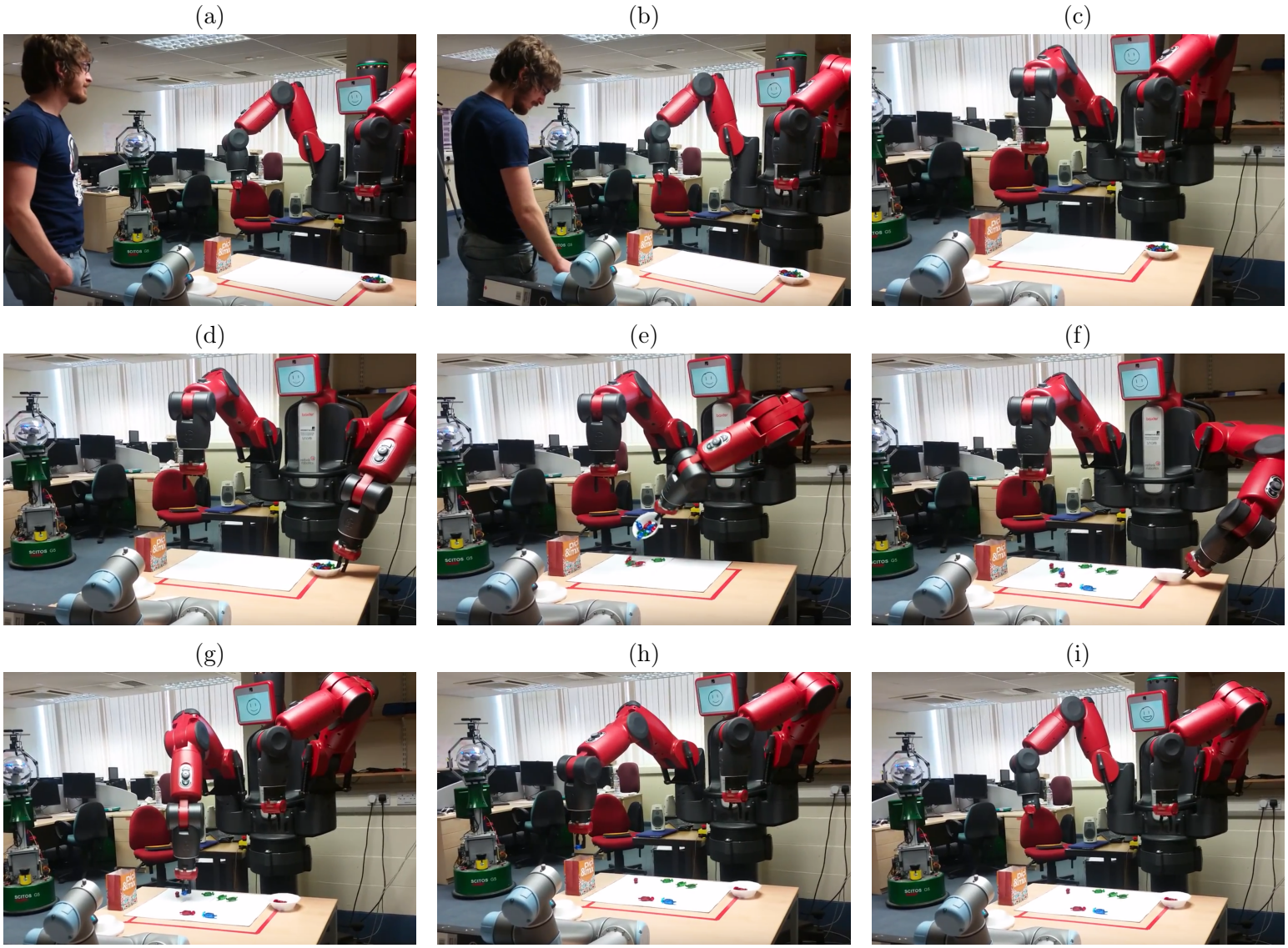


Figure 6.2: An example of the complete working system: (a) Baxter recognises the customer and asks them to provide a voice command for their sweets. (b) The customer says how many blue, red and green sweets they want and confirms on the Android app. (c) Baxter checks the table to make sure there aren't enough sweets on the table already. (d) He looks for the bowl and goes to grab it. (e) The bowl is tipped in two locations above the page. (f) The sweets on the table are scanned, counted and analysed. (g) The requested sweets are grabbed from the table. (h) The grabbed sweets are placed into the sweet bag. (i) Baxter has completed his request, thanks the customer and waits for the next one.

6.3 Evaluating the System

To test the overall system, five different human customers were asked to walk up to Baxter and order some sweets. The reliability of the system was judged on multiple levels: on whether Baxter understood the command, on whether he correctly placed the requested sweets in the bag and on the customer satisfaction with the ease of the

process. After doing the human tests, a few observations were made below on the problems encountered.

Variations in voice/accents provided difficulty for the voice recognition system (although that was expected and more a limitation of the device than the software). Sometimes the voice recognition didn't recognise the colour stated, leaving the rest to be unrecognised too. With more time, multiple accents could be trialled and an increasingly wider search of terms could be accepted for the colours to correct this. Tipping the sweets onto the table, whilst reasonably reliable at separating the sweets, the tilting method still managed to produce hard-to-separate sweet piles. It tended to depend on how the sweets were placed in the bowl initially. As long as the colours and orientations were mixed it seemed OK however, if they were all bunched together in colour order, all facing the same way, a lot of sweets could be tipped out at once. Since the addition of the speech to Baxter, the customers found the interaction reasonably simple to carry out. The face letting the customer know when they were recognised also helped (in case of a slightly inaccurate detection and the customer needed to move about a bit more).

The trials overall, even after multiple customers had approached Baxter, were very good on the whole, and reliable enough that if Baxter did make a mistake, he knew about it and could correct it. If he missed grabbing a sweet, he would try to grab another, if he hadn't tipped enough sweets on the table to recognise, he would continue tipping until there were enough. Occasionally Baxter grabbed two sweets from a pile instead of one and gave them slightly more than they ordered but that wasn't a important issue. The main issues were on the setup of the system (either the TTS or app server had an issue in starting), but after a reboot of the roslaunch file, the system tended to work well again.

6.4 Limitations/Improvements

Overall, whilst most of the actions in the shop had been tested to determine reliability, there were still some issues that existed by the end of the project. Here, I will discuss the issues and limitations demonstrated by the final version of the system and approaches I would like to take to these problems if I would have had more time to do so.

Human interaction

The problem with the final interaction system was that there were some gaps in the conversation and interaction that made it not seem lifelike. The main improvements that could have been made with more time were to use a proper skeleton recognition system, so Baxter could constantly recognise the person's skeleton and perform gesture recognition. I feel like the ability for the customer to exchange items with Baxter, such as exchange money or take the bag of sweets would have added to the realism of the

interaction greatly. Another issue that was mentioned was to implement threading and idle movements. Threading would have helped Baxter move both arms together in more human-like movements (instead of moving them one at a time). Idle movements were discussed as an addition to make Baxter seem lifelike even when he was not talking to a customer. As future tasks, it would be nice for those to be implemented so at the open day, Baxter would not be stood still until a customer approached him.

Sweet Recognition
Whilst this system was reasonably reliable by the end (and a lot more efficient than it was in initial iterations), there were still some problems that needed to be rectified. To be more robust, the sweet recognition system should have been made to work within different shop-like settings. The problem was different light sources would affect the testing data used to train the neural network. Therefore if the shop had a different coloured light source, the sweet colour recognition would be unreliable. I think a good way to solve this problem would be to add a user setup method to train new sweets under new conditions and automate the process. That would mean a user could then run the software, place multiple target sweets onto the table and Baxter would record their data and train the neural network. That addition to the system would certainly make it easier and more robust to be used in different rooms throughout the university for open days.

Sweet Singulation
If there was more time, a more efficient sweet singulation algorithm would be proposed/developed. The problem with singulating the sweets vision-wise and manipulation-wise is that there was no clear way to do this task. I carried out a lot of research and there is no specified way to be able to recognise individual, deformable objects within a cluster, with variable shape and colour. The fact that the sweets could vary in shape so much, meant that most vision techniques using shape recognition would not work - for example, template matching was considered but it could not be reliably implemented. The variation in colour meant that the sweets could not be differentiated easily by colour as all three wrapper colours could overlap in the HSV or RGB colourspace. Therefore, the combined method I used, using colour detection, then other processing methods was the best method I could manage to implement, but with more time, I think there would be a more accurate, computationally efficient approach to solve this.

6.5 Project Reflection

Over the past twelve weeks, I have really enjoyed this project and I think that mainly, it went very well and I am pleased with the overall deliverables produced. The main

problem I think I had was organisation of time. After doing the initial planning report and the supervisor/assessor presentation at week 8, the main issues with the feedback from both those meetings were that the time limit was overly ambitious and at some point, some features would have to be cut out. By the end of the project, there were multiple discussed parts of the project that were not developed: money exchange with the customer and fully developed sweet singulation were some of the key features that I didn't have time to complete. Whilst money exchange with the customer was a stretch goal anyway, the sweet singulation methods turned out to be a lot more of a complex issue than initially proposed.

If I were to do the project again, I think I would have started coding and work on the project earlier. A major issue was that I started developing the bowl recognition system with the Kinect as the first task. It took a long time to get used to the methods and PCL libraries. The problem was that the first main coding task was based on two completely new pieces of software: ROS and the PCL library. I feel that if I had started with the simpler OpenCV/Python based coding tasks first, it would've made the project easier to start and develop and could have prevented getting stuck for as long in the initial few weeks. Possibly the reason why the bowl recognition task took so long is because ROS was quite difficult to pick up and use and there wasn't a lot of great documentation out there to explain the core concepts of it. Once I got to around week 6, I felt like I fully understood the principles of ROS and could therefore do a lot more with it.

Looking back at the initial objectives, they stated I needed to develop a vision system for the bowl, sweets, a manipulation system for them and a some human interaction systems. I believe that I have achieved all of these basic objectives to some extent and am very happy with the quality of the results produced at each stage. I thoroughly enjoyed the overall project and feel like I got a lot out of it. The weekly meetings with my supervisor were very useful, to discuss and develop ideas with but mainly, I feel a sense of achievement that I mainly developed this entire project on my own.

References

[1] S. S. Srinivasa, D. Berenson, M. Cakmak, A. Collet, M. R. Dogar, A. D. Dragan, R. A. Knepper, T. Niemueller, K. Strabala, M. Vande Weghe, and J. Ziegler. Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, 100(8):2410–2428, Aug 2012.

[2] Daniel Toal, Colin Flanagan, Caimin Jones, and Bob Strunz. Subsumption architecture for the control of robots. In *Proc. of the IMC*, volume 13. Citeseer, 1996.

[3] Andrew Blake and Michael Isard. *Active contours : the application of techniques from graphics, vision, control theory and statistics to visual tracking of shapes in motion*. Springer, Berlin, London, New York, 1998.

[4] Eric N. Mortensen. Vision-assisted image editing. *SIGGRAPH Comput. Graph.*, 33(4):55–57, November 1999.

[5] P.J. Huber, J. Wiley, and W. InterScience. *Robust statistics*. Wiley New York, 1981.

[6] L. H. Quam. Hierarchical warp stereo. In *Image Understanding Workshop*, New Orleans, Lousiana,, December 1984.

[7] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[8] D. G. Lowe. Local feature view clustering for 3d object recognition. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–682–I–688 vol.1, 2001.

[9] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1989.

[10] Yan Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–506–II–513 Vol.2, June 2004.

[11] J. Maitin-Shepard, J. Lei M. Cusumano-Towner, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *icra*, 2010.

[12] Megha Gupta, Jorg Muller, and Gaurav S Sukhatme. Using manipulation primitives for object sorting in cluttered environments. *Automation Science and Engineering, IEEE Transactions on*, 12(2):608–614, 2015.

[13] Frederick Jelinek. *Statistical methods for speech recognition*. MIT press, 1997.

[14] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. The basic ideas in neural networks. *Commun. ACM*, 37(3):87–92, March 1994.

Appendices

Appendix A

External Materials

In this project, there were many external materials used for inspiration and development. For this reason, any ROS nodes that are publically available, were not included in this section as the ones used are already mentioned in the report.

Within the project files:

One of the PhD students within the robotics lab has a github: `https://github.com/OMARI1988`. His code provided a lot of the inspiration for mine so parts were taken from his publically hosted projects: **objectfinder.py** was directly taken from the github, as was some of the setup code for Baxter's movement, adapted from the robotics tabletop/board game code.

Other code used externally tended to be from documentation and tutorials, which are referenced in footnotes throughout the project where necessary.

Appendix B

Ethical Issues Addressed

Whilst in this project, there weren't many ethical issues to address, one has to consider the implications of AI in general with this type of task. In the customer-shopkeeper interaction, Baxter 'observes' customers and passers-by throughout the running of the software. In the robotics department, I came across other PhD students having difficulty gaining consent for taking Kinect recordings of skeletons. Whilst Baxter doesn't expressly record them, it could stray into the grey area of whether the person should give their permission to have their images taken/analysed by Baxter in the first place.