KNOWLEDGE-BASED SYSTEM FOR AUTONOMOUS CONTROL OF INTELLIGENT MASTICATION ROBOTS

Ramin Odisho

A thesis submitted in fulfilment of the requirements for the degree of

Master of Engineering in Mechanical Engineering

The University of Auckland

March 2016

Abstract

The study of the mechanical and chemical properties of food is commonly known as *Food Texture Analysis* (FTA); it is an important area of research in fields such as the health and food sciences. The majority of FTA methods employ human sensory panels because they produce realistic results from 'in-vivo' experiments; however, due to the subjective nature of sensory evaluation, the results obtained from different panel members can be inconsistent. This inherent variability in experimental outcomes can lead to difficulties when interpreting and comparing results and is the focus of this research.

Prior contributions to this research led to the development of a mastication robot; the motive being that robots are not inherently subject to the variability associated with human sensory perception and preference. The robot is capable of emulating human mastication by performing a family of rhythmic chewing motions that approximate the trajectories of human molar teeth. However, the robot was unable to adapt its chewing behaviour to food properties as they varied during the mastication process; this had a negative impact on results as mastication is a complex, dynamic process and cannot be accurately emulated by a static system.

The aim of this project was to develop a *Knowledge-Based System* (KBS) that could learn how human chewing patterns change during mastication and as a result, enable the robot to adapt its actions to changes in food properties. To accomplish this, the KBS was designed to store data regarding changes in mastication parameters with respect to varying food properties; the data would then be analysed (via machine learning algorithms) to discover mathematical relationships (knowledge) within the data. The KBS uses these relationships to control the robot (supervisory) based on feedback from the robot regarding the properties of the food.

Under KBS control, the robot autonomously emulated human mastication and produced results that were both, realistic and consistent; this indicates that the KBS has successfully enabled the robot to better approximate human masticatory behaviour. While this is a positive outcome, significant work is still required before such systems can be considered as viable alternatives to traditional sensory panels.

Acknowledgements

The author would like to thank the following people for their help and support during this project:

My family, for their love and support; especially my dearest mother, Olivia, to whom I owe my every success.

Professor Peter Xu, for his valuable supervision and guidance during the project.

Professor E. Allen Foegeding and his colleagues at NCSU, for their advice and data contributions.

Dr. Marie-Agnès Peyron of INRA, France, for her advice and data contributions to the project.

Esra Cakir-Fuller, for her data contributions.

Ashani Perera, for her assistance as a student intern.

Publications

1) Work undertaken during this project has been presented at the *Food Structures Digestion and Health* (FSDH) conference in Wellington, New Zealand during October, 2015.
2) The work presented in this thesis was accepted by the *International Symposium on Industrial Electronics* (ISIE) for presentation at the ISIE 2016 conference.

Table of Contents

List of Figures

Glossary

CNS	Central Nervous System
CPG	Central Pattern Generator
CSV	Comma Separated Values
DLL	Dynamic Link Library
EMG	Electromyography
FTA	Food Texture Analysis
GUI	Graphical User Interface
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KBS	Knowledge-Based System
KDB	Knowledge Database
MDB	Mastication Database
MDS	Multidimensional Scaling
MLR	Multiple Linear Regression
OKF	Object-oriented Knowledge Framework
PID	Proportional Integral Derivative
PSD	Particle Size Distribution
SQL	Structured Query Language
TPA	Texture Profile Analysis
TPM	Texture Profile Method
UML	Unified Modelling Language
USB	Universal Serial Bus

Chapter 1

Introduction

1.1 Project Background

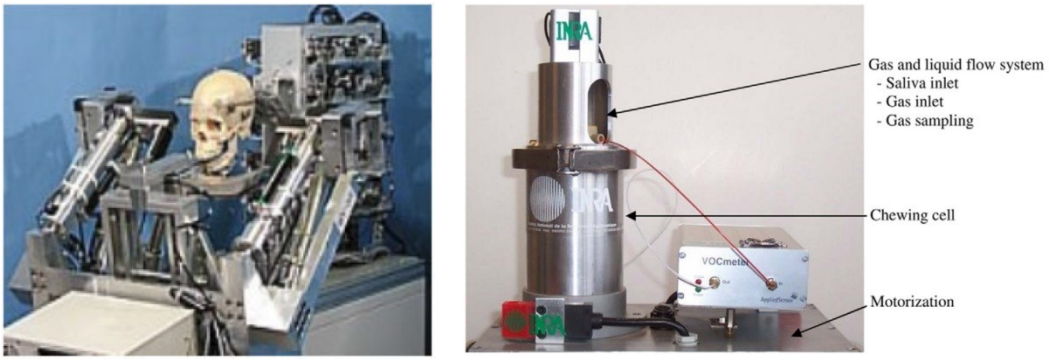
The study of the mechanical and chemical properties of food is of particular importance in many areas of science and research. In food science and other related fields, the study of these properties is generally known as *Food Texture Analysis* (FTA). [1] One of the main goals of FTA is to provide qualitative feedback regarding food properties in terms of descriptive words such as 'crunchy', 'crispy', 'chewy', 'hard' and 'soft' etc. [2] There are two main methods that researchers employ to conduct FTA, namely, 'instrumental' methods and 'sensory evaluation'. [3, 4] Instrumental methods use devices designed to measure the physical properties of a given food sample and then relate these measurements to the descriptive terms mentioned above. [4] An example of such a device is the *TA.HD Plus* texture analyser shown in Figure 1.1.

Sensory evaluation methods employ trained (or in some cases, untrained) sensory panels consisting of human participants; the panel members consume a set of food samples then grade and label each sample's texture based on their own sensory perceptions. [5, 6] The sensory panel approach produces realistic results because experiments are performed 'in-vivo' however, these results can be inconsistent due to the subjective nature of sensory evaluation. [4] This inherent variability in results stems from many factors such as a participant's incentives/motives or 'tokens' [6] and the participant's 'sensory-liking' [7] of the food being consumed. While there are techniques that have been developed to mitigate these discrepancies, [8-11] variations

can still occur and their effects can make comparing and analysing results difficult; this variability in experimental outcomes is the main focus of this research.

Figure 1.1: *TA.HD Plus* texture analyser (reproduced from [12]).

From the above, it can be seen that sensory methods produce realistic but inconsistent results, whereas the opposite is true for instrumental methods. Based on this seemingly 'complimentary' relationship between instrumental and sensory methods, an instrument capable of emulating human mastication can be a potential solution to the variability problem. There are a number of robots available in the fields of food science and dentistry that may be suitable for this purpose; [13] however, in their current state, most of these robots are either too complex to be practical, or too simple to produce useful results. The *Waseda-Yamanashi Jaw* series of dental training robots (please see Figure 1.2a) are good examples of robots that replicate human mastication accurately, [14] but are too complex to be used effectively for an application such as FTA. Instruments such as the *Chewing Simulator* (please see Figure 1.2b) are easy to control, [15] but cannot reproduce human mastication with accuracy that is comparable to sensory panels. This shows that there is a need for an instrument that can emulate human mastication with sufficient accuracy, while being relatively simple to control and operate.



(a) (b)

Figure 1.2: Robots capable of emulating human mastication (a) *Waseda-Yamanashi Jaw* (b) *Chewing Simulator* (reproduced from [13] and [15] respectively).

1.2 SDOF Mastication Robot

Prior contributions to this research resulted in the development of a single-degree-of-freedom (SDOF) mastication robot. The robot uses a four-bar linkage to trace a family of trajectories that can accurately approximate the chewing trajectories of human molar teeth. The four-bar linkage is driven by a single motor (hence SDOF) making the robot relatively simple to control. The initial prototype of the robot was designed as a linkage-based device to perform standardised chewing for use in food evaluation. [16] The main design features of that prototype (please see Figure 1.3) consisted of:

- Four-bar linkage (extended to six-bar to maintain orientation) for trajectory tracing with adjustable ground link for trajectory setting
- Food retention mechanism
- Shock absorber to prevent excessive chewing forces
- LabView interface for control

While that prototype was successful in approximating human chewing trajectories, it could not be controlled autonomously; the operator was required to either manually control the robot, or to specify parameters/goals (number of cycles, opening/closing time etc.) to be met by the robot.

Given the limitations of the initial design, further work was undertaken to explore various control strategies that would enable the robot to function autonomously. [17, 18] In addition to improving the control methods, several modifications were made as follows:

- Three-dimensional (3D) force sensor added to enable bite/chew force measurement
- Automatic food manipulation mechanism added to reposition food sample between chewing cycles
- Shock absorber replaced with spring-mass damper system

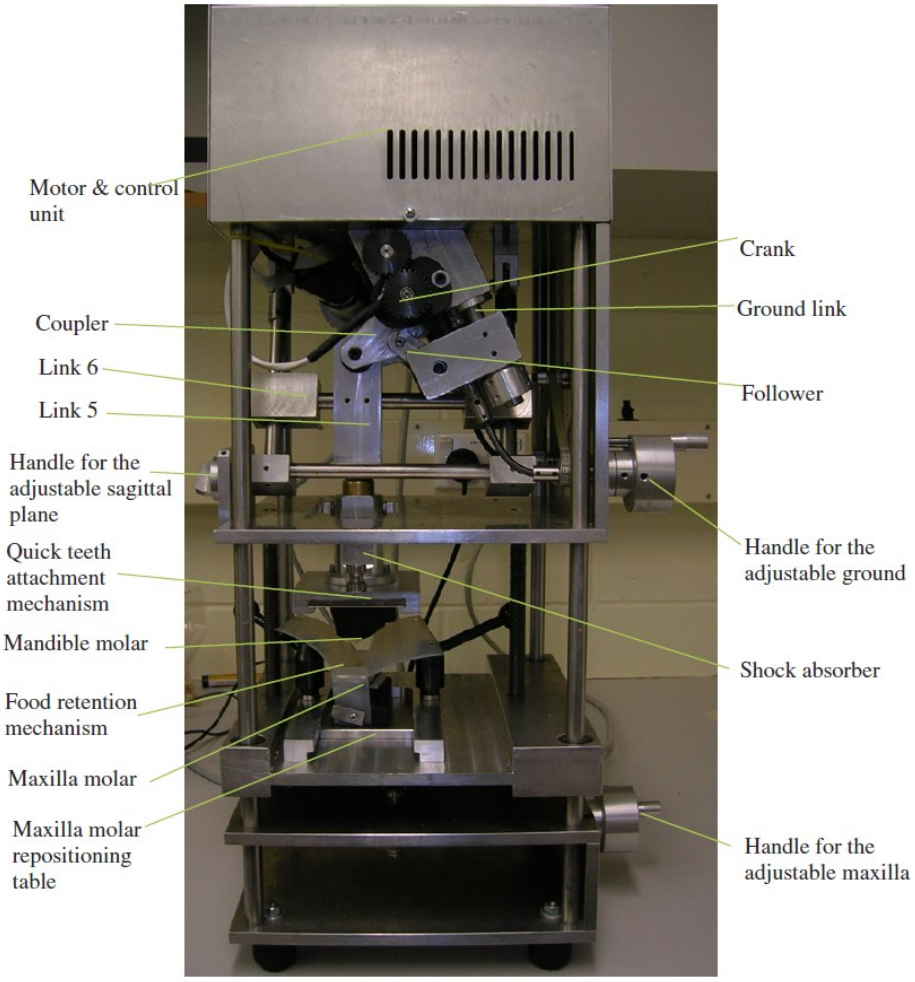


Figure 1.3: Initial prototype of the SDOF mastication robot (reproduced from [16]).

The control system for this augmented prototype was implemented as an adaptive *Fuzzy Logic* controller, designed to adjust motor velocity based on chewing force measurements. Although the velocity control was successful, other factors such as the number of chewing cycles to perform were not controlled. Furthermore, the controller did not emulate human masticatory patterns but rather, used a 'numerical' approach where the output velocity is scaled in response to changes in chewing force. The robot was later developed into a commercial product (please see Figure 1.4 - Figure 1.6), the version that was used during the course of this current project.

Figure 1.4: Current model of the SDOF mastication robot.

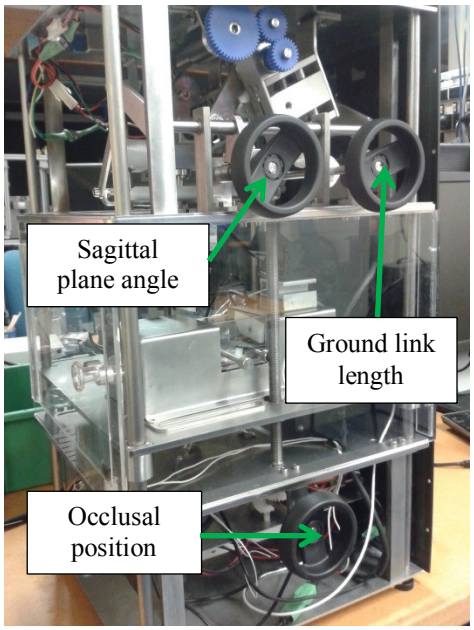


Figure 1.5: SDOF mastication robot showing adjustment knobs.

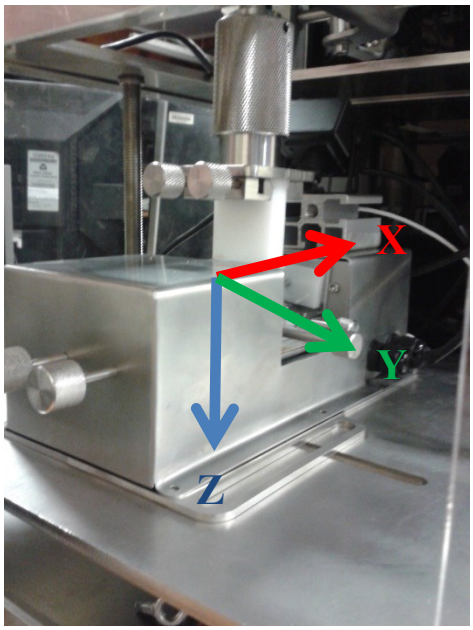


Figure 1.6: SDOF mastication robot showing coordinate system used.

1.3 Knowledge-Based System

As discussed, the methods that were used to control the robot had several shortcomings; the purpose of this project is to address those shortcomings through the development of a *Knowledge-Based System* (KBS).

Knowledge-based systems are realised in various forms and find applications in diverse fields ranging from patient diagnosis in medicine, [19] to risk and crisis

management in the travel and tourism industry. [20] Although the specific implementation of a KBS may vary between applications, the core architecture of a typical KBS will consist of three main aspects:

1) *Data Acquisition* (DA)
2) *Knowledge Discovery* (KD)
3) *Decision Making* (DM)

The KBS developed in this project has been implemented in the form of a *Graphical User Interface* (GUI) based software application, that contains tabs for each of the above aspects (please see Figure 1.7). The following sections will discuss each of these aspects in more detail and outline their implementation in this project.

N.B. In the following sections, the term 'target' is used to refer to the target of the KBS output; this is usually a human operator or 'expert' however, in this project, the target refers to the SDOF mastication robot.

1.3.1 Data Acquisition

The function of the data acquisition feature of a KBS is to extract and store data that may contain useful knowledge; data may be stored in a general purpose database however, a database that is specific to the application is generally preferred.

In this implementation, data is presented to the KBS in the form of a spreadsheet containing time-varying mastication parameters and food properties. The KBS then displays data to the user via the 'Data Acquisition' tab of the GUI. To extract the data from the spreadsheet and into the KBS, the user 'maps' the column headings of the spreadsheet to database fields; this is done via the GUI and will be discussed further in section 3.2.1.

1.3.2 Knowledge Discovery

Once the KBS has been populated with sufficient data, the user may examine the database for knowledge (typically in the form of mathematical relationships) within the data. The general approach to 'discovering' the knowledge, involves applying machine learning algorithms to the data of interest.

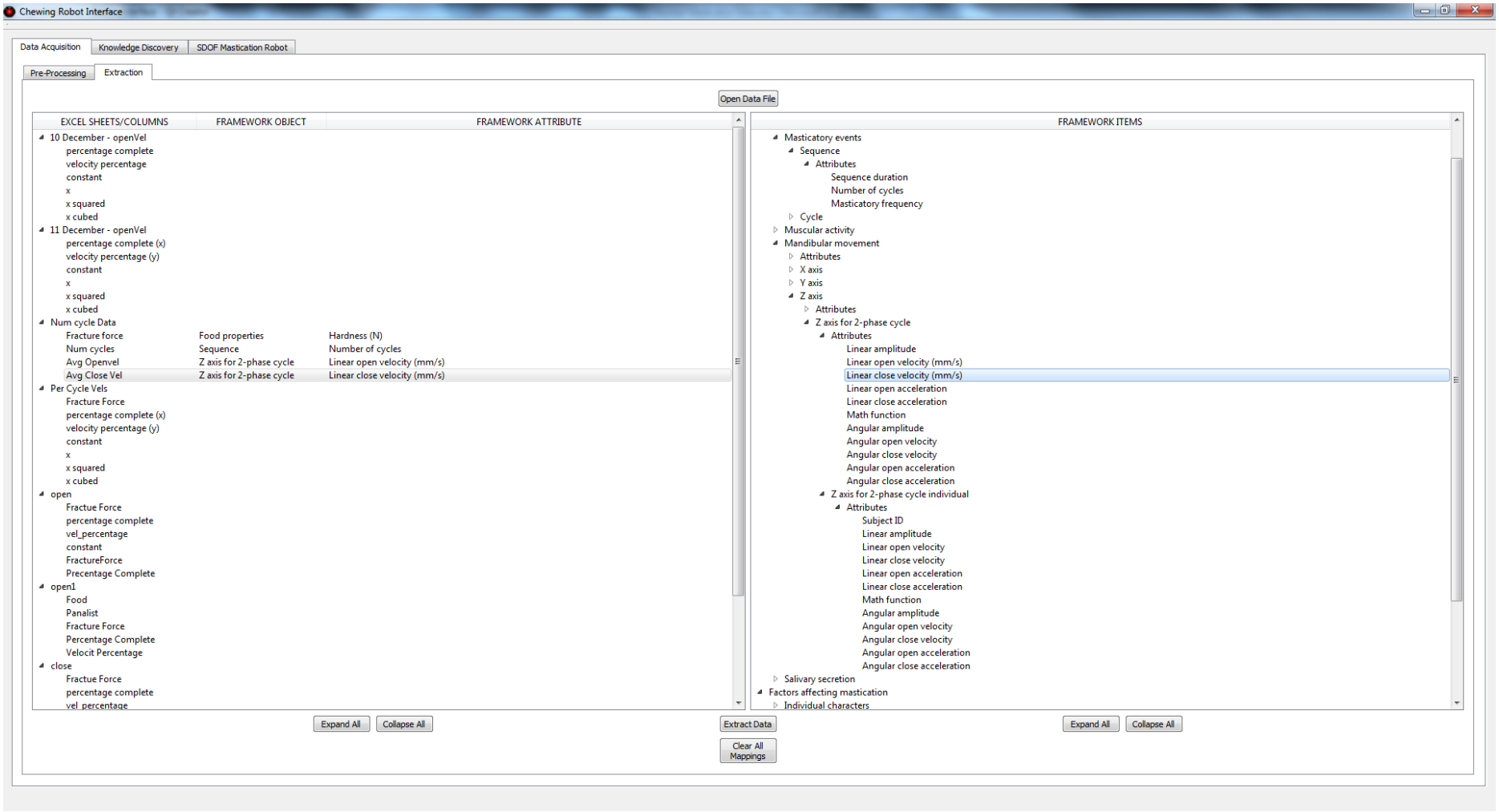


Figure 1.7: Screenshot of GUI showing the various tabs.

The 'Knowledge Discovery' tab of the GUI contains tools to facilitate the application of machine learning algorithms such as *Multiple Linear Regression* (MLR). The user can perform MLR by selecting a 'Target Variable' from the database, along with one or more 'Explanatory Variables'; the KBS will then attempt to find a relationship between these variables in terms of regression coefficients. MLR has been implemented because it is well suited to human mastication data; however, many other algorithms such as *Clustering* and *Artificial Neural Networks* may be applied depending on the application and nature of the data in question.

1.3.3 Decision Making

The purpose of the decision making aspect of a KBS is to generate information regarding the future state of the environment (application); i.e. given the current state of the environment, the KBS will specify the future state that needs to be achieved. To generate this information, the KBS uses knowledge or 'facts' (stored in its KB) about its environment, in combination with feedback from the target regarding the current environmental parameters or state. This information will enable the target to take actions to transition the environment to the required state.

The GUI contains a tab labelled 'SDOF Mastication Robot'; this tab contains widgets (GUI elements) that allow the user to interact with the robot. KBS control of the robot has been implemented as follows:

1) The user initiates autonomous control via the GUI
2) The robot starts by reading its sensors to provide feedback to the KBS
3) The KBS uses its knowledge and feedback from the robot, to send new parameters for the robot to follow
4) The robot adjusts its actuators according to the new parameters set by the KBS

The above sequence of events is repeated (without user intervention) for the duration of the mastication process.

The software implementation of the robot is contained in a *Dynamic-Link Library* (DLL) which is separate from the KBS software (please see Figure 1.8). The main advantage of using a DLL is that the robot-specific control (i.e. the low-level actuator commands) is not handled by the KBS; this means that the robot (or any target in

general) appears as a 'black-box' to the KBS. This architecture has several other noteworthy advantages and will be discussed later in section 3.6.

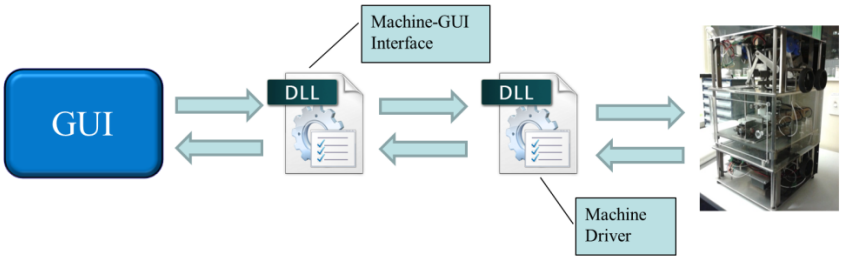


Figure 1.8: General system architecture.

To summarise, the KBS, when implemented as described above, can enable the robot to exhibit more human-like chewing behaviour. This is accomplished in the following way:

1) The KBS stores data regarding changes in mastication parameters with respect to varying food properties
2) The data is analysed (via machine learning algorithms) to discover mathematical relationships (knowledge) within the data
3) The KBS uses these relationships to control the robot (supervisory) based on feedback from the robot regarding the properties of the food

The specific software implementation of the KBS and the algorithms used will be discussed in greater detail in chapter 3.

Chapter 2

Literature Review

This chapter will discuss the current literature that is relevant to this project; literature regarding the background of the project and the proposed solutions will be reviewed. The chapter is organised according to the following sections:

2.1) *Food Texture Analysis* (FTA)

This section will focus on the following two points:

a) The definition of food texture and its associated terminology.

b) Discussion and comparison of the main measurement techniques used to quantify food texture.

2.2) *Sensory Panels*

This section will discuss human sensory panels and will focus on the following two main areas:

a) Typical experimental procedures used to conduct sensory analysis/evaluation.

b) Human variability as it pertains to sensory evaluation and its effects on FTA.

2.3) *Central Pattern Generator* (CPG)

This section will discuss CPGs with focus on three main areas:

a) The biological function of CPGs in humans.

b) The role of CPGs in human mastication.

c) The use of CPGs in *Bio-Mimetic Robots* to enable human-like behaviour.

2.4) *Knowledge-Based System* (KBS)
This section will review typical KBS design and discuss various applications that employ KBSs for automation and control.

2.5) *Object-oriented Knowledge Framework* (OKF)
This section will focus on OKFs and typical applications in which they are used.

2.1 Food Texture Analysis

This section will discuss FTA by first reviewing how food texture is defined and why it is important. Human perception of food texture and the physiological and psychological effects it has will be discussed. Finally, the main methods used in the field of FTA will be reviewed.

2.1.1 Definition of Food Texture

The properties of food play a vital role in maintaining good health, happiness and overall wellbeing in human beings. With regards to health, the effects that food properties have on the ability of humans to extract and process nutrients from food, are of particular importance and have been studied extensively. In addition to the effects of food on the biological and physiological aspects of health, food properties also have a significant impact on mental and emotional health. The ability to masticate and enjoy food is a basic human need; [21] its lack thereof not only contributes to malnutrition, but also has adverse effects on psychological wellbeing. Studies have shown that the emotional state of an individual is related to his/her eating patterns; a common finding among these studies is that individuals tend to consume more food while in a negative emotional state. [22] This reinforces the fact that mastication and eating are fundamental human activities required to maintain a healthy physical and psychological state; therefore, an understanding of foods and their properties is crucial for the development of foods that best meet these needs. The various foods that humans consume generally consist of highly complex structures with many diverse and varying properties; of these properties, ‘texture’ was not recognised as a standalone property until the 1920s when it was realised that the texture of food plays an important role in human perception of food quality

(Figure 2.1 shows texture as part of a finite continuum representing human sensory perception of food quality). [23]

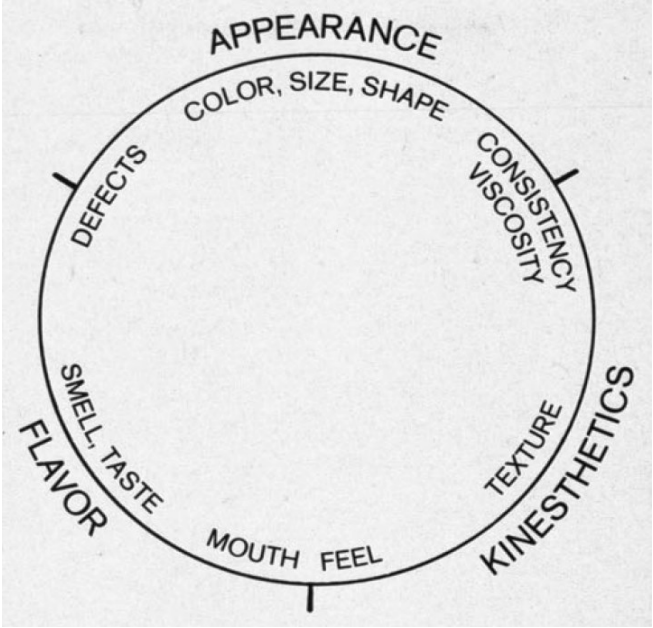


Figure 2.1: Human sensory perception of food quality as a continuum of senses (reproduced from [23]).

The texture of food is a subjective, sensory property and as such, its interpretation can vary based on individual consumer perception; [24] this leads to difficulty when attempting to establish a formal, universally accepted definition for the term, and is an on-going issue within the scientific community. Despite the ambiguous and subjective nature of food texture, a generally accepted definition has been published by the *International Organization for Standardization* (ISO) as follows: "all the mechanical, geometrical and surface attributes of a product perceptible by means of mechanical, tactile and, where appropriate, visual and auditory responses"; [25] from this definition, it is evident that the sensory perception of food texture is a complex process that can involve a multitude of sensory functions. The texture of a food is typically quantified using terms such as 'crispy', 'crunchy', 'chewy', 'hard' and 'soft' etc. [2] These terms are measured via two main techniques as will be discussed in the following section.

2.1.2 Measurement Techniques

Food texture is a sensory property and can be measured directly via appropriate human senses; while this is possibly the most accurate measurement technique, it is considered subjective and varies depending on the characteristics of the individual.

[23] Objective measurement techniques are possible using indirect rheological methods; while such methods suffer the disadvantage of being only as useful as their similarity to human senses, they are not susceptible to sources of discrepancies such as drift and fatigue. [23]

Rheological methods for food texture measurement yield objective results through the use of properly calibrated instruments; in order for such instruments to be effective, certain procedures must be followed as shown below: [23, 26]

1) Perhaps the most important factor contributing to the effectiveness of instrumental methods is how clearly and precisely texture terms are defined. Having a standard against which measurements are made and compared is essential if the results obtained are to be meaningful and comparable.
2) Because food texture is a combination of sensory properties, it is possible to use multiple methods/instruments to achieve a combined result that is more accurate than a simple test using a single instrument. This is generally achieved via the use of *regression analysis* with the aim of determining which combination of methods/instruments will best predict the sensory property being measured.
3) When using instrumental methods, validation is an important step in the measurement process; it is achieved by repeating tests and using different samples to cover the range of food that is being tested. Validation is required due to variability among samples of a particular food type; without proper validation, relationships obtained may not be generally applicable to the food but rather, a 'chance' relationship local to the samples under test.
4) Due to multiple methods being used to measure specific textural parameters as mentioned above, it is possible that certain methods contribute significantly to multiple texture parameters. This can be exploited via multivariate analysis such as *multiple linear regression* (MLR) to integrate various methods to achieve a better overall prediction of texture.

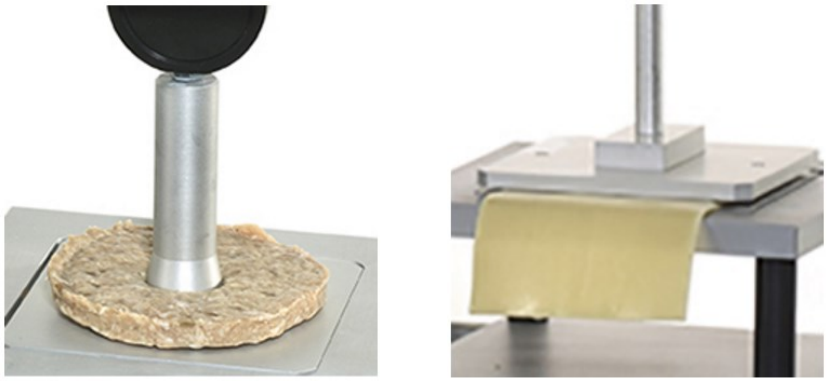
There are various instruments used to conduct objective texture analysis; typically, each instrument focusses on a specific parameter to be measured that can then be used to predict (or contribute to the prediction of) the sensory quality in question as

mentioned above. The following will discuss some common instruments used in industry ranging from manual to fully software controlled instruments.

Figure 2.2 shows a manual instrument designed for measuring the firmness of fruits and vegetables. This machine is operated manually via a lever or hand wheel and measurements are made via tensile or compressive tests. Such units typically also include a digital force readout that is used to determine peak forces applied during testing; these forces are then translated to the property being measured.

Figure 2.2: An example of a manual testing instrument for measuring 'firmness' (reproduced from [27])

In addition to instruments that measure general properties such as 'firmness', there are instruments that measure more specific textural terms such as 'burger consistency' and 'Lasagne stickiness' as shown in Figure 2.3.



(a) (b)

Figure 2.3: Specialised texture measurement instruments for (a) burger consistency and (b) Lasagne stickiness (reproduced from [27]).

As an alternative to manual testing of food texture, various instruments are available that can partially automate the testing process. A good example of such an instrument

is the *TA1 Texture Analyzer* shown in Figure 2.4; this instrument is capable of producing relatively accurate results and being software controlled, it does not suffer from operator related subjectivity.

Figure 2.4: *TA1 Texture Analyzer*, a fully software controlled food texture analyser (reproduced from [27]).

As mentioned earlier in this section, the effectiveness of instrumental methods for food texture analysis relies heavily on a standard against which testing can be performed. There are various methods that can be used to achieve this however, a popular and widely accepted method is the *Texture Profile Method* (TPM); this method provides a procedure through which a texture 'profile' may be developed for a food from test results (the overall procedure is shown in Figure 2.5).

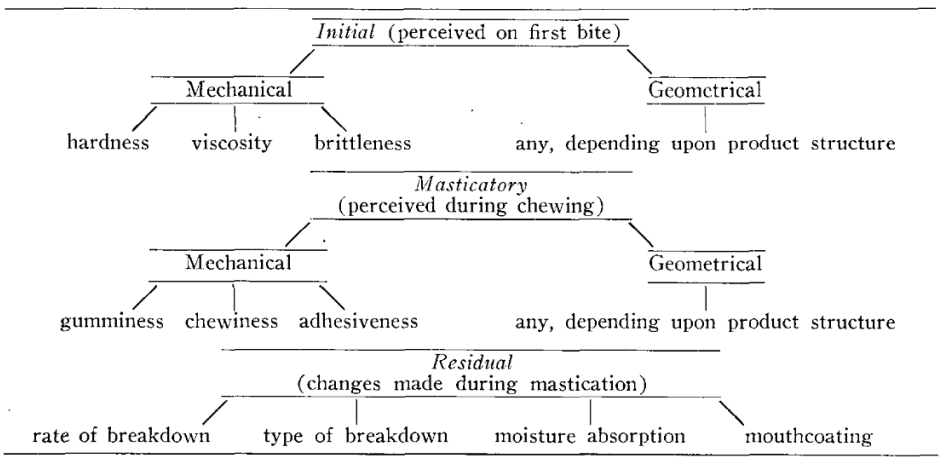
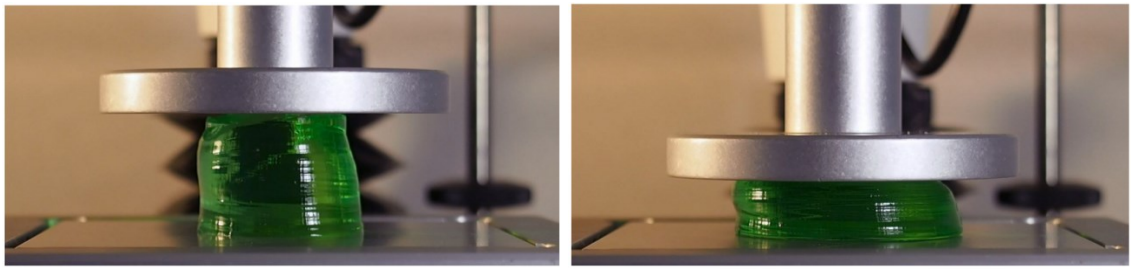


Figure 2.5: General procedure for evaluating texture via TPM (reproduced from [28]).

There are various ways in which instruments may be used to apply TPM to texture measurement; however, a simple yet effective procedure known as *Texture Profile Analysis* (TPA) is commonly used. The TPA procedure involves deforming the food sample via two consecutive compressions and measuring the change in the sample's properties; Figure 2.6 shows a food sample before and after uniaxial compression during TPA. A common property measured using TPA is 'hardness', this is measured as the peak force during compression as shown in Figure 2.7.



(a) (b)

Figure 2.6: Jelly food sample during TPA (a) before compression and (b) after compression (reproduced from [29]).

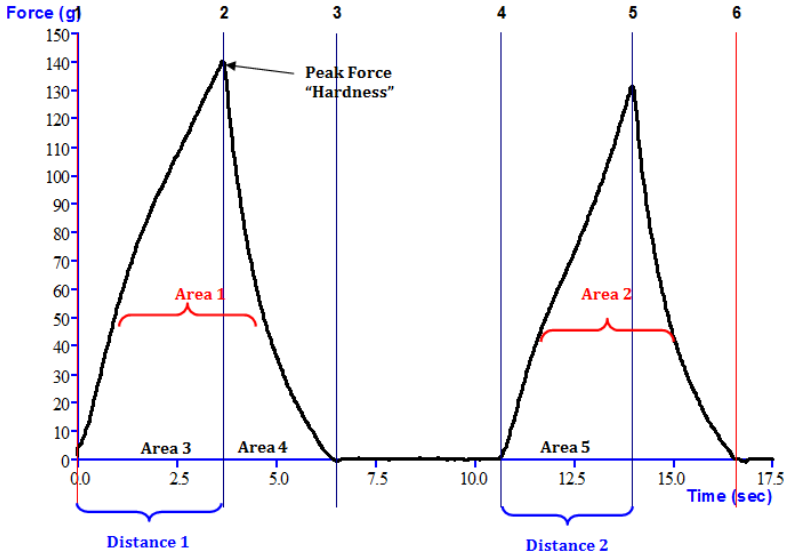


Figure 2.7: Typical force profile of two-compression TPA test showing hardness measurement (reproduced from [29]).

Procedure such as TPM and TPA are useful for classifying texture according to specific aspects such as 'mechanical' and 'geometrical' however, in order for these methods to be applied effectively and be comparable to results obtained from sensory panels, they must be used in conjunction with standard rating/scoring scales. Various scales have been developed in the past to measure specific parameters such as

hardness, brittleness and chewiness etc. one such scale is shown in Figure 2.8 for hardness, with common food items as references.

Panel rating	Product	Brand or type	Manufacturer	Sample size	Temp.
1	Cream cheese	Philadelphia	Kraft Foods	½"	45–55°F
2	Egg white	hard-cooked 5 min	.....	½" tip	room
3	Frankfurters	large, uncooked, skinless	Mogen David Kosher Meat Products Corp.	½"	50–65°F
4	Cheese	yellow, American, pasteurized process	Kraft Foods	½"	50–65°F
5	Olives	exquisite giant size, stuffed	Cresca Co.	1 olive	50–65°F
6	Peanuts	cocktail type in vacuum tin	Planters Peanuts	1 nut	room
7	Carrots	uncooked, fresh	.....	½"	room
8	Peanut brittle	candy part	Kraft Foods	.....	room
9	Rock candy	.....	Dryden & Palmer	.....	room

Figure 2.8: Standard rating scale used for scoring food 'hardness' (reproduced from [30]).

While the use of standard ratings leads to effective use of test results, issues can arise when attempting to compare and analyse results obtained using multiple or outdated standards. Such issues have been addressed by the development of techniques such as *Multidimensional Scaling* (MDS). [31] The MDS technique is a mathematical procedure that aims to 'cluster' given 'objects' (foods in this case) on a spatial representation or map according to measures of similarity or dissimilarity; foods that are assessed as being similar are clustered closer together while foods that are assessed as being dissimilar are mapped further apart. Through the use of techniques such as MDS, researchers can better analyse results obtained from various sources and standards; this is possible because information is obtained regarding *how* panelists decided to rate the various attributes of the foods they were sampling which in turn, can be used to 'scale' results obtained from various tests accordingly.

While instrumental methods are desirable for their objective results and relatively cost-effective experimental procedures, they only produce results that can be used to *estimate* the actual texture parameters as perceived by humans; this fundamental drawback is why sensory panels still remain the ideal measurement tool for food texture.

2.2 Sensory Panels

As discussed in the previous section, instrumental methods are very useful for the objective results they provide while also being cost-effective; however, sensory

panels are still considered the ideal form of texture measurement due to experiments being performed 'in-vivo' which leads to more realistic results. It can be seen that combining sensory and instrumental methods would be ideal because of the potential to obtain both realistic and consistent results; this was the basis for the design of the mastication robot used in this project. As discussed in section 1.1, the goal of this project was to enable the robot to exhibit human-like behaviour; in order to explore how this may be achieved, sensory panels and the general experimental procedures involved will be reviewed.

2.2.1 Experimental Procedures

Sensory evaluation of food is a sensitive process and conditions must be carefully controlled; according to [32] the following factors should be considered when conducting sensory evaluation experiments.

1) The sensory testing environment:
   The environment in which sensory evaluation is performed has a significant effect on the performance of a sensory panel. The environment should be free from external influences or 'sensory pollution' such as strong smells/odours or noise. In addition, the layout of the testing facility is important and should be designed such that panelists are not exposed to the preparation area before testing; this shows that humans have a natural tendency to be biased and is one of the root problems in sensory evaluation.

   The evaluation area also plays an important role in the design of experiments. It is crucial that the evaluation process is performed in a quiet and interruption free area; furthermore, panelists should not be able to communicate and thus, influence each other during the course of the experiment. This again reinforces the subjective nature of sensory evaluation and highlights another advantage in using a mastication robot (no outside influences). To mitigate inter-panelist influences, experiments are typically carried out in cubicles or booths with barriers (generally made of plywood or other suitable material) separating the panelists.

   The climate conditions within the evaluation area are also important; ventilation and odour must be controlled so as not to negatively affect the

panelists. Negative climate effects can be mitigated via effective ventilation; in addition, maintain a positive pressure within the evaluation area will reduce the effects of odours crossing over from the preparation area. Illumination also affects the panelists therefore lighting should be controlled; this shows that sensory perception is a complex combination of factors and that human perception of food cannot be isolated to specific senses.

2) Test protocol:

   The methods used during experiments should be highly standardised and repeatable. The samples served must be carefully designed and most importantly, variations between individual samples should not be detectable by the panelists. Sample temperature must also be considered and is an example of an external factor that may alter an individual's sensory perception of the food.

   Food samples presented to panelists should be randomised to avoid any order bias; this issue relates to human tendency to perceive and react to patterns, a mastication robot would not be affected by such a phenomenon. Swallowing of food samples should generally be avoided to mitigate the influence of food samples on each other. Panelists should be instructed to expectorate samples after mastication; the exception being during acceptance or similar tests where the consumer's perception of the food after swallowing is of interest.

3) Panelist considerations:

   Several factors must be considered when selecting panelists to participate in sensory evaluation. It is preferred that panelists are participating voluntarily and not paid to do so; the reason for this is that if excessive compensation is involved, panelists may not be motivated to concentrate on the evaluation process and are simply participating for the monetary reward. While this can be mitigated by the use of 'tokens' or small rewards in the form of snacks/treats, social gatherings and gifts etc. the use of a mastication robot can avoid this issue all together. This serves as an example of how instrumental and sensory methods can be combined to mitigate their weaknesses.

The above methods and techniques are generally well applied by researchers in the field; a recent study regarding the effects of black tea on Chinese steamed bread [33] serves as an example of experiments that effectively use the above guidelines to

conduct TPA. This study used TPA to measure the textural properties of the bread crust; this was done using three different regions of the crust to validate results as mentioned earlier. Following TPA, sensory evaluation was carried using nine trained panelists with the experiment being performed over the course of three days. Finally, panelists were presented with six samples each in a randomised order to prevent biasing (a sample of the score sheet used can be seen in Figure A.1). Another study regarding the effects of chemical treatments on fresh-cut papaya [34] used a 'hedonic' nine point rating scale (1 = dislike extremely, 5 = neutral, 9 = like extremely) similar to that of Figure 2.8. The sensory evaluation was carried out by thirty untrained panelists, in a room with controlled illumination; food sample temperature was maintained at 12 °C.

The environmental and human related factors affecting a given sensory evaluation experiment can be quite extensive; studies have shown that some of the major contributing factors to sensory evaluation discrepancies are: location of testing, age and attitude towards the food being consumed [35] (this is shown in Figure 2.9).

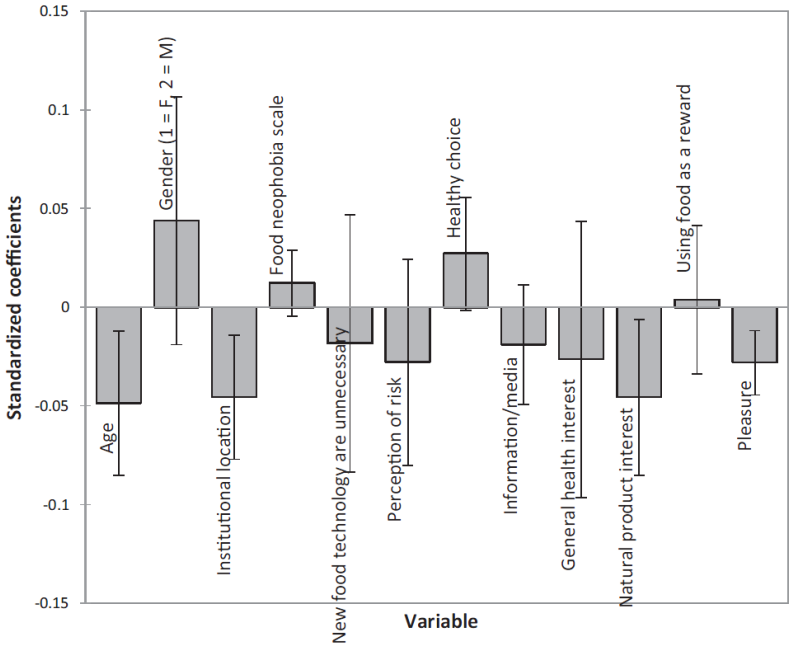


Figure 2.9: Environmental and individual factors affecting acceptance of Soda products. (reproduced from [35]).

From the above it can be seen that sensory evaluation is a very delicate process with many factors to consider; there are many variables and external influences that can affect the outcome of experiments. In addition to environmental factors, there are several issues arising due to human nature and psychology that can be somewhat

mitigated but not avoided. These issues provide a strong basis for the use of a mastication robot as a potential alternative for sensory evaluation.

2.2.2 Effects of Human Variability

According to [36], one of the fundamental measures of human sensory perception is the 'absolute threshold' (aka. 'detection threshold'); this threshold is defined as the minimum intensity of a given stimuli that will register with the consumer's consciousness (i.e. produce sensations that will be noticed). In practice, difficulties arise when attempting to measure and apply this threshold; this is mainly due to variability associated with the point at which an individual feels that a stimulus has been sensed. This variability can be quite large and even vary with the same individual, over multiple trials of the same test. In addition to the inherent variation associated with the threshold of perception, 'sensory adaptation' can also become a source of discrepancies. A common situation where sensory adaptation can become a problem is when a strong stimulus is sensed, then sensed again after sensing weaker stimuli; due to the initial exposure to the strong stimulus, a degree of adaptation may have taken place hence, when the strong stimulus is sensed in the second instance, it appears less potent. The occurrence of sensory adaptation is closely linked to the sequence in which the samples are presented; therefore, the samples presented to the consumer must be carefully controlled in order to reduce any negative side effects as discussed.

Another source of variability stems from an individual's sense of satisfaction with regards to the food being consumed. Various studies have been carried out with the aim of understanding the determining factors of sensory and food satisfaction; one such study was conducted by serving consumers two types of chicken soup with differing sensory qualities [37] and observing their reaction to the soups given certain conditions. The findings of this study showed that consumer sensory satisfaction was primarily influenced by 'taste' and 'appearance'; the 'odour' and 'texture' of the soup also influenced the consumer satisfaction to a lesser degree. These results show that human perception of food quality and acceptance is indeed affected by hedonic sensations; due to the highly subjective nature of these sensations, significant variations among individual panel members can be observed.

While it can be seen that sensory panel based methods suffer from several sources of variability, techniques have been developed to mitigate these variations. An obvious yet effective method of reducing variability is to use trained panelists during sensory evaluation experiments. Studies have been conducted that compare the effectiveness of trained and untrained panelists; one such study is documented in [38] and found statistically significant evidence supporting that trained panelists produce more consistent results. Another technique that can be used to deal with variations among panel members is 'segmentation; [10] this technique involves applying clustering algorithms to group panel members that tend to have similar responses, producing more 'homogeneous' panels that yield a better understanding of sensory 'liking'.

From the above discussion regarding human variability in sensory evaluation, it can be seen that although various techniques exist to mitigate these variations, their effects are still significant. The mastication robot used during this project was developed with the intention of addressing this problem; however, the robot lacks the ability to adapt its chewing behaviour resulting in consistent but unrealistic results.

2.3 Central Pattern Generators

In order to enable the mastication robot to emulate human masticatory behaviour, it is necessary to gain a general understanding of how human mastication is controlled by the brain. The following will discuss the *Central Pattern Generator* (CPG) which plays an important role in rhythmic control of muscles.

2.3.1 Biological Function

CPGs are neural circuits that produce patterns of neural activity without any sensory feedback; these neural patterns drive muscles in rhythmic patterns of motion. [39] Common examples of CPGs in humans are coordination of motor muscles when walking, swallowing and breathing for example. [40] Human motion is modulated by specific parts of the *Central Nervous System* (CNS) such as the *motor cortex* and *cerebellum*. However, studies have shown that CPGs are responsible for coordinating certain rhythmic motor functions in the human body. [41]

Various studies have shown that CPGs are capable of generating rhythmic patterns to drive muscles without any *external* sensory feedback. These are generally known as

'fictive motor patterns' that are produced even when the CPG is in complete sensory isolation.[40] While CPGs can operate without external sensory feedback, 'internal' signals are still required to modulate the patterns being produced. Figure 2.10 shows an example of the typical modulation and timing signals associated with CPGs.

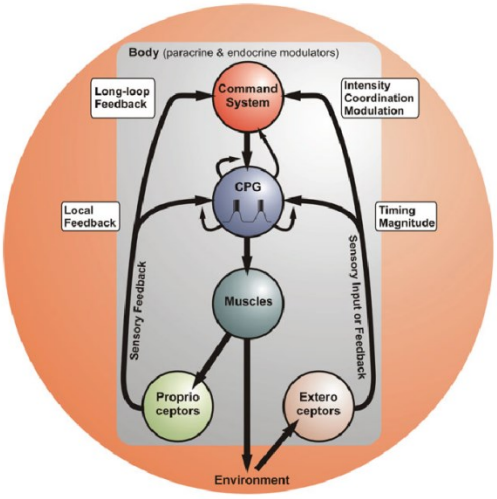


Figure 2.10: Internal CPG feedback signals (reproduced from [42]).

2.3.2 Role in Human Mastication

Human mastication is a complex process that involves highly coordinated muscle contractions to achieve specific chewing patterns. CPGs are responsible for the complex coordination of these masticatory muscles to produce the necessary rhythmic jaw movements. *Electromyography* (EMG) measurements have shown that CPG controlled muscle activity is present during mastication. [43]

2.3.3 Application to Bio-Mimetic Robots

CPGs have been applied to various bio-mimetic devices to achieve human-like neural control. One such example is the use of CPGs to design a robotic fish that can perform rhythmic swimming motions. [44]. Figure 2.11 shows a schematic of the robotic fish; it can be seen that the body of the fish is segmented and utilises an array of CPGs. Through coordinated rhythmic patterns produced by the CPGs, realistic swimming motion can be achieved.

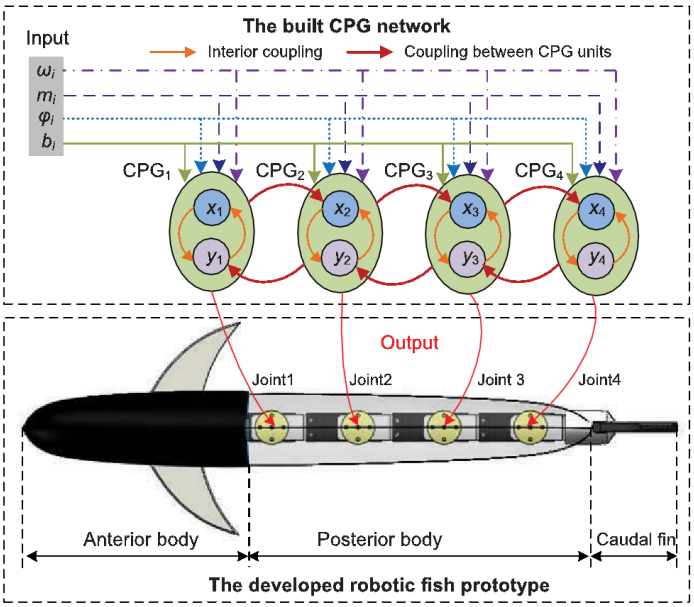


Figure 2.11: Robotic fish with CPG controlled swimming motions (reproduced from [44]).

In the case of mastication robots, CPGs can provide rhythmic chewing motions without the need for sensory feedback; [45] while this is desirable, it is more suited to multi-degree of freedom robots as they are more suited to oscillatory motion. Because the robot that was used during this project was a single degree of freedom robot, the use of an artificial CPG-based controller was not appropriate. The next section discusses knowledge-based systems which are more suitable for the current application.

2.4 Knowledge-Based Systems

Modern knowledge-based systems evolved from early computer systems known as *Expert Systems*. [46] These expert systems were designed to allow computers to exhibit artificial intelligence in the form of decision-making capabilities similar to those of human 'experts' in a particular field. To achieve this, expert systems typically utilize two sub-systems, a *Knowledge Base* (KB) and an *Inference Engine* (IE). The purpose of the KB is to store information regarding the application with which the expert system is concerned; this information is typically stored in the form of known 'facts' (regarding the application and its environment) and relationships between these facts. The IE is designed to use the information stored in the KB, to reason and deduce new knowledge that can be used to make decisions. These decisions can ultimately result in actions that will affect the environment and hence,

allow the system to exhibit seemingly intelligent behaviour in response to changes in its environment.

2.4.1 Applications

Knowledge-based systems are abstract and versatile tools that can be found in a wide variety of applications. The tourism industry relies heavily on crisis management to both, plan for and avoid unforeseen emergency situations. [20] In this particular application regarding the tourism industry, the KBS is implemented as an autonomous software system with the ability to assist with data extraction and dissemination during a crisis. The KBS uses artificial intelligence based methods to collect, sort and store data with the aim of extracting useful information; this information may in turn, provide insight into potential causes and risks associated with certain crisis situations. The KBS that was implemented in this application consisted of three main parts:

1) *Knowledge Extractor:*
   This aspect of the KBS was used to store raw data into the *knowledge base* and subsequently extract knowledge using known rules regarding the data.
2) *Knowledge Server:*
   This component facilitates the dissemination of knowledge when it is required.
3) *Knowledge Manager:*
   Knowledge management is one of the crucial aspects of the KBS; it is similar to a *Database Management System* and is responsible for the knowledge repository available to the KBS.

Another example of where a KBS may be useful is in the medical field; [19] in this application, a framework is developed to help medical practitioners with the development of KBSs to aid with medical diagnosis. Such applications are suitable candidates to be used with a KBS because patient diagnosis for example, almost exclusively relies on the medical professional's *knowledge* regarding symptoms and potential causes. If such knowledge can be encoded and transferred to a KBS, tasks such as patient diagnosis may become automated or at the least, the KBS can assist doctors and reduce the likelihood of a misdiagnosis.

Considering the applications discussed above, the application of a KBS to the mastication robot being used appears to be the logical approach. If knowledge can be

extracted from raw mastication data and stored in the KBS, a system can developed that would enable the robot to exhibit autonomous mastication behaviour and adaptive chewing patterns. However, in order for the KBS to be effective in such an application, data must be presented in a standard format that is abstracted from the data source. This is necessary because the operation of a KBS can be thought of as a modelling task aiming to describe an *abstraction* of a 'real-world' phenomenon via a conceptual model. [19] This abstraction can be achieved via *Knowledge Frameworks* as discussed in the next section.

2.5 Object-oriented Knowledge Frameworks.

As mentioned in the previous section, a knowledge framework is especially useful for abstracting from various sources of information; an example of how this is useful can be seen when considering a framework designed to retrieve knowledge stored in various hardware and software systems. [47] In this application, knowledge of interest may be: stored in various hardware/software mediums, encoded according to various standards and represented in different ways (symbolic, numerical, text based etc.) In this particular application, the knowledge framework took the form of a flowchart (please see Figure 2.12); this chart can be used to systematically asses acquired data via a standard procedure that is independent of the source.

Another effective application of a knowledge framework is in the field of service robotics. Service robots are routinely required to carry out complex tasks in challenging real-world environments; [48] tasks such as object detection based on sensory data alone can be quite challenging. To overcome these challenges, an ontology-based framework is used that organises knowledge into two basic functions: 'knowledge description' and 'knowledge association' as shown in Figure 2.13. In this framework, the knowledge description aspect aims to define robot knowledge in terms of five main classes; these range from low-level perception knowledge obtained from sensory data, to high-level knowledge regarding the physical world such as the identification of objects and spaces. The knowledge association aspect aims to find relationships between the available knowledge using various mathematical, logical and statistical methods.

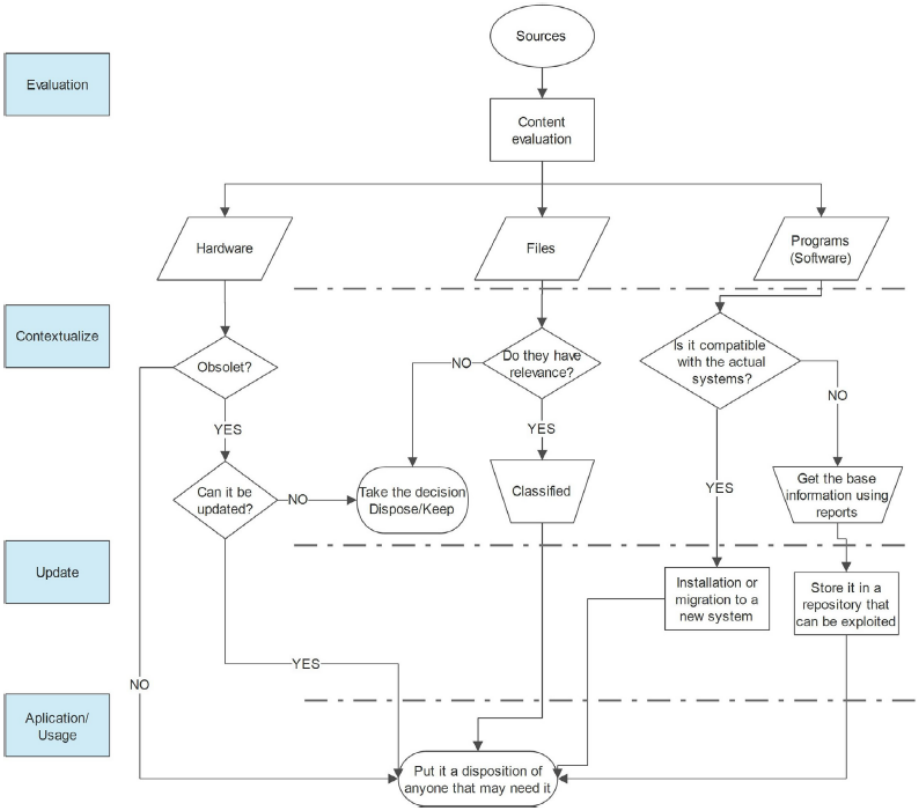


Figure 2.12: Knowledge framework for acquisition of knowledge from various software version (reproduced from [47]).

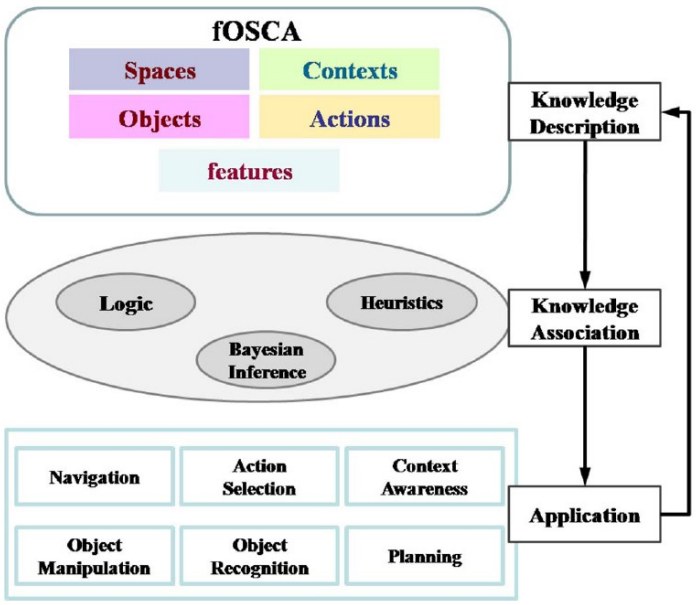


Figure 2.13: *OUR-K* knowledge framework designed for use with service robots (reproduced form [48]).

It can be seen that the framework used in Figure 2.13 fits well with the KBS as described in the previous section. The knowledge description aspect of the framework effectively abstracts from the low-level sensors that produce the data, by the use of a 'class' system. The knowledge association aspect can then freely search for knowledge within the data regardless of how it was obtained; this is possible due to the abstract interface provided by the knowledge classes.

In the case of mastication robots, knowledge frameworks that organise data in terms of 'objects' and 'attributes' (analogous to the object-oriented programming) have proved to be effective [49]. These Object-oriented knowledge frameworks (OKF) are abstract tools that can be used to decompose an application domain into a network of inter-connected 'object'-'attribute' units (as shown in Figure 2.14). Such a framework enables the KBS to abstract from low-level variations by organising stored data in terms of application specific features or 'attributes'.

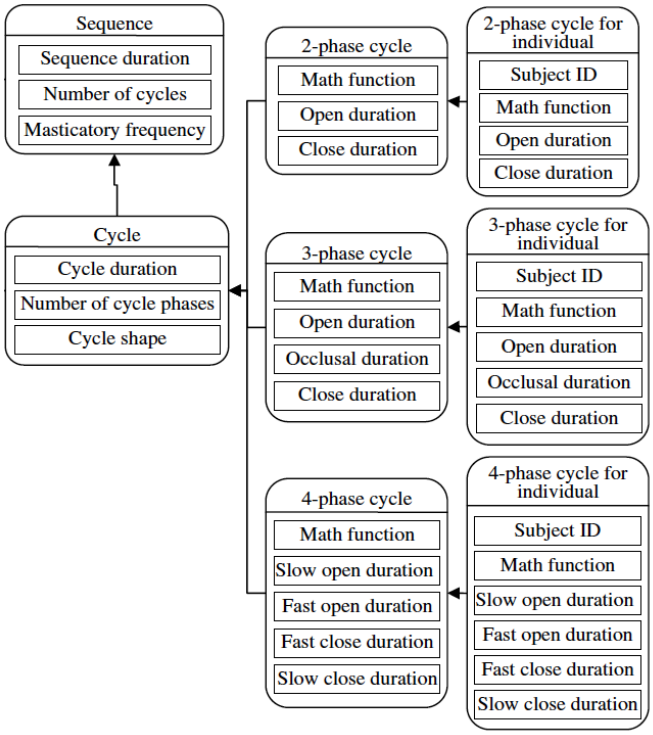


Figure 2.14: Example of an *Object-oriented Knowledge Framework* (reproduce from [49]).

Once the KBS database is populated with data stored according to an OKF, knowledge extraction may be carried out in terms of the application parameters such as the chewing parameters shown above. The knowledge extraction process is carried out with virtually no reference to the underlying data from which the knowledge is

extracted; this is made possible by the OKF serving as an ‘interface’ between low-level data and high-level, application specific parameters.

Chapter 3

Knowledge-Based System Design

3.1 Design Overview

As discussed in previous sections, the overall goal of this project was to design and implement a *Knowledge-Based System* (KBS) that would function as a 'supervisory' controller; the system would enable the robot (mentioned in section 1.2) to operate autonomously. The system that has been designed consists of the following five main components:

1) *Main Interface*:
   Main component that handles the interactions between all other system components (including the operator/user). The main interface has been implemented as a software application with a *Graphical User Interface* (GUI).
2) *Mastication Database* (MDB):
   Database containing all data extracted from various sources (e.g. experimental data, clinical data etc.) This database is queried during the knowledge extraction process.
3) *Knowledge Database* (KDB):
   Database containing all knowledge extracted from data that is stored in the MDB. The stored knowledge is retrieved during operation and is used to issue supervisory control commands to the robot.

4) *Robot Interface*:
   This is the section of the *main interface* that is responsible for communicating with the robot; its purpose is to facilitate user input to the robot and provide feedback from the robot.
5) *Robot Driver*:
   The robot driver serves as the interface between the KBS and the robot; it has been implemented as a *Dynamic-Link Library* (DLL) that contains the interface code and low-level control algorithms that are specific to the robot.

The following subsections will discuss each of the above components in more detail; the main focus will be on the user interactions and the overall system operation; for detailed implementation of each component, please see chapter 3

3.2 Main Interface

The main interface is the primary component in the system; it is the 'central hub' and is responsible for facilitating communication between all other components and the user. The main interface has been implemented as a software application with a GUI through which all user interactions occur. Figure 1.7 shows the GUI window that the user will see when the application is initially launched.

The following sub-sections will briefly describe each of the items in Figure 1.7.

3.2.1 Data Acquisition

The data acquisition tab contains two sub-tabs namely, *Pre-Processing* and *Extraction.* These tabs contain all the necessary widgets required by the user to extract data from a file, pre-process it and finally, store it into the MDB. The purpose of each tab is further explained below.

Pre-Processing

This tab is reserved for features that can be used to pre-process data before it is stored into the MDB. The aim of pre-processing is to make the data more suitable to be stored by removing/reducing any discrepancies in the data that may 'pollute' the database. During this project, pre-processing was done via the *SPSS* software package because it met the needs of the application; however, because knowledge-based

systems are generic in nature, future applications may require special pre-processing functionality which can be implemented here.

Extraction

This tab contains the widgets required for the user to browse for and open data files. Once opened, the data in the files can be extracted and stored in the MDB. The widgets in this tab are shown in Figure 3.1 and are explained below.

1) Open File Button

Clicking on this button shows the Windows *Open File* dialog box; from this dialog, the user can browse for and open any compatible file for data extraction. Compatible file formats are: *Excel Spreadsheets* and *Comma Separated Values* (CSV). Once open, the contents of the file will be displayed in the tree view widgets for processing.

2) Data File Tree View

This widget represents a tree view that displays the contents of the currently opened data file. The 'root' level of the tree shows the 'sheets' contained in the file (equivalent to the sheets in an Excel spreadsheet). Root-level items can be expanded or collapsed by double-clicking on the desired item or, via the *Expand All* and *Collapse All* buttons below the tree view. When a root-level item is expanded, the column headers in the sheet it represents are displayed; these column headers can be mapped to fields in the MDB by first, selecting a column header, then double-clicking on the desired MDB attribute field (the MDB fields are shown in a second tree view that will be discussed below).

3) Mastication Database Tree View

This widget behaves in much the same way as the *Data File Tree View* widget mentioned above however, rather than spreadsheet-like 'sheets' and 'columns', it represents database fields that are used to label and organise data stored in the MDB. The MDB fields are structured according to an *Object-oriented Knowledge Framework* (OKF) that represents the various aspects of human mastication in terms of inter-connected 'object'-'attribute' units. [49] (please see section 2.5 regarding knowledge frameworks). To illustrate the relationship between the MDB and the OKF, a portion of the OKF regarding *Mandibular movements* (object) is shown in Figure 3.2; extracted data containing measurements such as the 'position of the

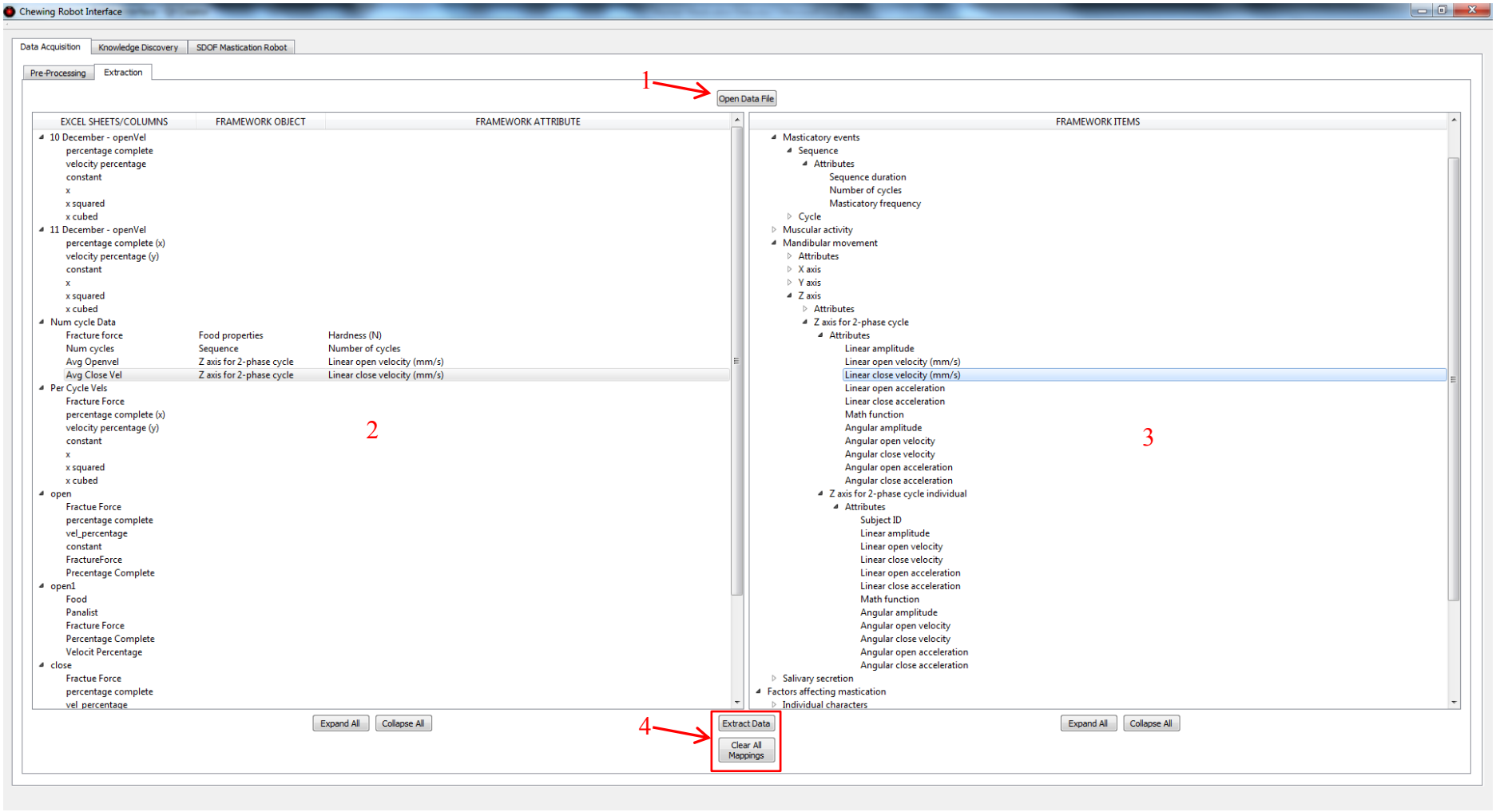


Figure 3.1: Data extraction tab.

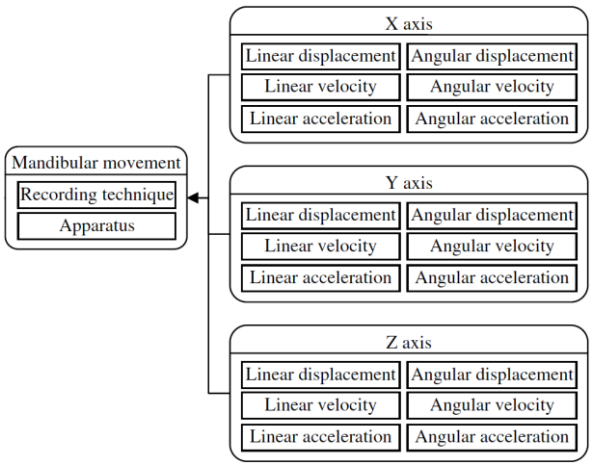


Figure 3.2: Portion of OKF regarding mandibular movements (reproduced from [49]).

mandible' or the 'chewing velocity' for example, would be stored in the database under *Linear displacement* and *Linear velocity* respectively (attributes).

4) Mapping Buttons

The tree view widgets mentioned above are used to designate mappings between data in a file to be processed, and database fields under which the data is to be stored; if the user needs to reset the mappings, the *Clear All Mappings* button can be used to remove all currently designated mappings; alternatively, individual mappings can be removed by double-clicking on the column header for which the designation is to be removed. Finally, once the required designations have been completed, the *Extract Data* button can be clicked to process the mappings; this involves extracting the data, ensuring it is correctly formatted and then storing it (along with various metadata such as the file name and date stored) into the MDB under the corresponding OKF fields.

3.2.2 Knowledge Discovery

The knowledge discovery tab (please see Figure 3.3) contains the tools required to extract knowledge from the data stored in the MDB; these tools take the form of machine learning algorithms (such as *Multiple Linear Regression* (MLR) and *Clustering*) that can reveal mathematical relationships within the data contained in the MDB. If the relationships obtained from the algorithms are deemed significant, they can be stored as knowledge in the KDB. The algorithms that have been considered for this project are MLR and clustering; the operator can use these algorithms through various widgets that are found in their respective tabs as shown below.

Regression

Regression was used during this project as it is a simple, versatile and well understood machine learning tool. Furthermore, due to its relatively simple design, the robot in use benefits primarily from knowledge of the *kinematics* of mastication; the data from which such knowledge is obtained, is well suited to MLR because it is 'continuous' and subject to analytical relationships such as *Newton's Laws of Motion*.

The regression tab contains widgets that implement MLR; the widgets are shown in Figure 3.3. The basic procedure for conducting MLR using these widgets is as follows:

1) The *Load Framework* button is clicked, populating the lists on the left and right with the MDB fields and MLR parameters respectively.
2) The dependent variable (framework attribute) is mapped to *Target Variable* then, one or more independent variables are mapped to *Explanatory Variables*.
3) The *Set Variables* button is clicked and data for each mapped variable is retrieved from the MDB. Constraints can be used to selectively retrieve data for MLR as will be shown later.
4) Once data has been successfully retrieved, the regression parameters are set and the *Perform Regression* button is clicked; this will execute the MLR algorithm which will produce the regression and correlation coefficients. The resulting data points and coefficients are plotted on the right of the GUI (this is done using the *scikit-learn* library).
5) If there is a strong relationship between the target and explanatory variables, the *Save Regression Results* button can be clicked to store the coefficients (and additional meta-data) into the KDB as knowledge.

The following list discusses the individual widgets that constitute the regression tab and shows how each widget is used to facilitate the MLR process presented above.

1) Load Framework Button

When this button is clicked, the *Mastication Database Tree View* widget is populated with the MDB fields in the same manner as discussed in section 3.2.1; in addition to this, the *Regression Tree View* widget is also populated with fields to which the target and explanatory variables regression can be assigned.

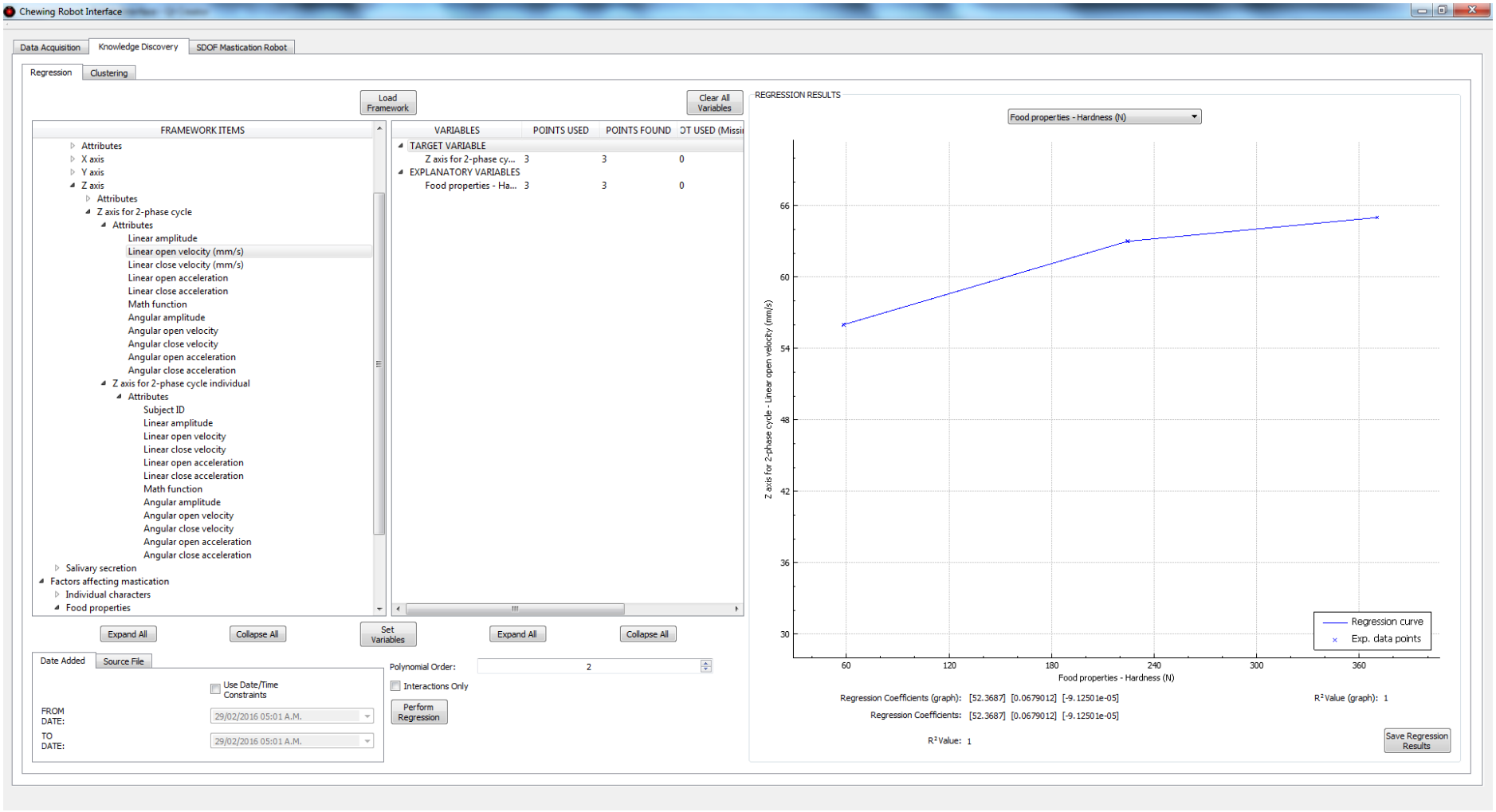


Figure 3.3: Knowledge extraction tab.

2) Mastication Database Tree View

This widget is virtually identical to its data acquisition counterpart discussed in section 3.2.1; the only difference is that here it is used to designate data to be *retrieved* from the MDB rather than stored. Before the retrieved data can be used by the MLR algorithm, the data fields in question must be mapped to target and explanatory regression variables. This mapping is performed by first selecting either *Target Variable* or *Explanatory Variables* from the *Regression Tree View* widget then, double-clicking on the desired MDB attribute field; this is repeated until *one* target variable and one or more explanatory variables have been set.

3) Regression Tree View

This widget (in conjunction with the MDB tree view widget) is used to designate, for the purposes MLR, which MDB fields are to be used as target variables and which are to be used explanatory variables. The mapping process is discussed in the above description of the MDB tree view widget.

4) Clear All Variables Button

This button can be clicked to reset the regression tree view widget and remove all MDB fields previously designated as target and/or explanatory variables. If a single MDB field is to be removed, this can be done by double-clicking on the field in question in the regression tree view widget.

5) Data Retrieval Parameters

The data retrieval parameters considered for this project are related to the 'extraction date' and 'source file' of the data stored in the MDB; by using these parameters to specify constraints on the data, it is possible to retrieve a specific subset of the MDB. The purpose of the extraction date parameter is to limit the data retrieved to only those points that were extracted (stored in the MDB) within a specified 'time window'; this is implemented via the calendar and time widgets in the *Date Added* tab (please see Figure 3.4) The source file parameter is designed to retrieve only those data points that were (or were not) extracted from a particular source file (multiple files may be specified); this is useful for processing data obtained from a particular experiment (or set of experiments) or conversely, for excluding data from said experiments. While the source file parameter is a useful feature, at the time of writing, it has only been partially implemented due to time constraints.

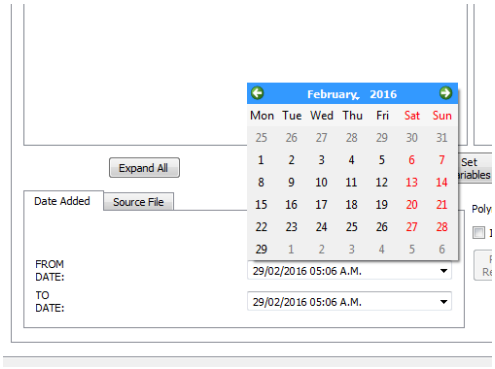


Figure 3.4: Calendar widget used to set date/time constraints.

6) Set Variables Button

As discussed above, in order to select data for use in MLR, the MDB fields containing the desired data must first be mapped to target and explanatory regression variables; the retrieval parameters can then be set to control which data points are extracted from the mapped fields (this is optional). The *Set Variables* button can now be clicked to search for and retrieve all matching data points from the MDB. Once the data retrieval process is complete, information regarding the actual number of points retrieved will be shown in the *Regression Tree View* widget; this information includes the number of points found (not yet retrieved) in the database for the given fields, the number of points discarded (due to constraints and/or discrepancies in the data)) and finally, the number of points actually retrieved.

7) Regression Parameters

Once the desired data has been retrieved from the MDB, certain regression parameters must be set before MLR can be performed. The following will explain the two widgets used to set the regression parameters:

1) *Polynomial Order:*

   This is a ‘spinbox’ widget that is used to specify the order of the curve or ‘fit’ that will be used during MLR. The user can enter any non-negative integer value into the spinbox; for example, if the user enters ‘2’, a second-order polynomial or ‘quadratic’ will be used; entering ‘3’ will result in a third-order polynomial or ‘cubic’ etc.

2) *Interactions Only:*

   This is a ‘checkbox’ widget that is used to specify whether or not to include only ‘interaction’ terms in the regression equation. Interactions are any higher-

order terms in the regression equation that that do not contain higher-order variables; for example, consider a regression equation with three variables *A, B* and *C* as shown in Equation (1);

$$y = A^2B + AB^2C - AB + ABC \quad (1)$$

all four terms are higher-order terms because each term contains more than one variable however, only the last two terms (*AB* and *ABC*) are interactions; the other two terms are not interactions because they contain higher-order *variables* (A^2 and B^2).

8) Perform Regression

This button is used to trigger the MLR algorithm to begin processing the data retrieved from the MDB; it is clicked once the retrieval and regression parameters have been set, and the user is satisfied with the number of pints retrieved for processing.

9) Regression Plot

This is a 'stack' widget containing a number of plot widgets that display the data points and 'fit' produced by the MLR algorithm. The stack contains plots of each explanatory variable plotted against the target variable; the desired plot can be selected via the dropdown box widget at the top of the plot.

10) Regression Results

The output of the MLR algorithm is displayed via several 'label' widgets located under the plot. The statistics reported are the regression coefficients and the R^2 value; these are reported for the overall equation (via the *Regression Coefficients* and R^2 *Value* widgets) and for the plot currently displayed (via the *Regression Coefficients (graph)* and R^2 *Value (graph)* widgets).

11) Save Regression Results Button

Once MLR has been performed, the user can examine the outputs (plots and coefficients) that have been displayed; from this, the user can determine whether or not there is a significant relationship between the target and explanatory variables used. If the results are not satisfactory, the user may remap certain MDB fields and/or change the retrieval parameters; the entire process may be restarted by clicking the *Clear All Variables* button as previously mentioned. If the results obtained are satisfactory and a potentially useful relationship has been discovered, the user may

click the *Save Regression Results* button to save the MLR output; the output is saved in the KDB under a specified format.

Clustering

The clustering tab contains various widgets that allow for *Cluster Analysis* to be performed on extracted mastication data. Due to the dynamic nature of human mastication and the complex interactions between the variables involved, it is difficult to search for any causal relationships (knowledge) that may exist within the data. Cluster analysis can help to reduce this complexity by grouping data into 'clusters' according to measures of dissimilarity between the data points; this grouping can provide insight into potential 'trends' or 'patterns' within the data that may be exploited to obtain knowledge.

At the time of writing, the clustering tools have not been fully implemented; the main reason being that MLR proved to be sufficient for obtaining knowledge regarding the kinematics of human mastication (the primary focus of the robot). In addition to this, the robot used is relatively simple therefore, complex knowledge regarding variables such as saliva and tongue-food interactions (while very useful domain knowledge) would not contribute to autonomous control of the robot. Despite its current limited use, clustering may become vital during future contributions to this project. This evident when considering that one of the main objectives of future work is to extend this KBS to more advanced robots; such robots would be capable of realistic behaviour, potentially requiring the use of complex learning algorithms such as clustering.

3.3 Mastication Database

When the user selects and opens a data file from which data is to be extracted into the MDB, the KBS first checks the file to determine whether or not it needs to be converted; this is necessary because the KBS has been designed (for simplicity) to operate on a single file format. If the selected file is a CSV file or a spreadsheet from older versions of Excel (*Excel 2003* or older) and it is being opened for the first time, it is converted to the current Excel format (file extension *.xlsx*). The converted file is stored under the *Converted Files* directory; once converted, the KBS will no longer use the original file, and all subsequent references to it will be redirected to the

converted file. Once a file has been opened (and converted if necessary), the data it contains can be extracted into the MDB.

The MDB has been implemented as a *document-oriented database* as opposed to the more commonly used *relational database*; the reason for this choice stems from the restrictions imposed on data by the predefined 'table' structures of the latter. Knowledge obtained from mastication data can take many forms; storing it in a relational database would result in redundant fields which is inefficient and wasteful of computational resources. Due to the lack of a rigid internal structure, document-oriented databases are well suited to handle the data associated with human mastication.

The MDB is stored as a file on the computer that is running the KBS software. The data stored in the MDB is organised as a collection 'elements' containing *key-value* pairs; each element is analogous to a single 'entry' or 'record' in a relational database *(*e.g. *SQL*) where 'key' is equivalent to a 'field' and 'value' is the associated data point. The elements are grouped together in a 'table' consisting of key-value pairs; the key represents the element's 'id' (a unique number assigned to each element) and the value contains the element.

Given the structure and organisation of the database as described above, it can be seen how data regarding human mastication can be mapped to the MDB fields, and organised according to an OKF (please see section 2.5 regarding the OKF). Once extracted and stored into the MDB, the data will be available to the knowledge discovery process.

3.4 Knowledge Database

The KDB has been implemented as a *document-oriented database* with structure and organisation virtually identical to that of the MDB. The main difference between the two databases is how the OKF is used; while data in the MDB is organised according to the OKF attribute it was mapped to, knowledge data in the KDB is organised according to the learning algorithm that produced it. The KDB retains information regarding the OKF mappings associated with the data from which a particular knowledge 'element' was obtained; this can be seen from the structure of the KDB elements and is required by the robot (as will be discussed in later sections).

Given the functions of the MDB and KDB as described, the OKF can be thought of as an interface between data and the knowledge discovery process; the purpose of the OKF in this regard, is to present data to the learning algorithms in a form that is abstract from the specifics of the data source. The knowledge that is stored in the KDB is used by the KBS during autonomous control of the robot.

3.5 Robot Interface

The robot interface is a tab in the GUI that contains various widgets that facilitate user interaction with the robot. This tab is not part of the *Main Interface* and is not visible by default; it is loaded dynamically at 'run-time' if the KBS detects that the *Robot Driver* (DLL file) is present in the KBS directory. The widgets in the robot interface are organised within several sub tabs as shown in Figure 3.5 and explained below.

3.5.1 Control Modes

During operation, the robot can be in any one of six states referred to as *control modes*; these control modes determine how the robot will be controlled and can be changed by the user in real-time. Of the six modes available, only three are directly applicable to the user; the remaining two modes are used internally by the robot. The three 'user modes' are briefly explained below:

1) *Manual:*
   In this mode, the robot is under 'manual' control and will not exhibit any autonomous behaviour. The user must manually issue commands to the robot via widgets in the *Manual* tab.
2) *Parametric:*
   This mode is considered 'semi-automatic' in that the user is not required to issue low-level commands to control the robot; however, the user must set the operating parameters or 'targets' of the robot (number of cycles to perform, chewing velocity etc.)
3) *Autonomous:*
   While in autonomous mode, the robot does not require any use interaction and operates exclusively via supervisory commands received from the KBS.

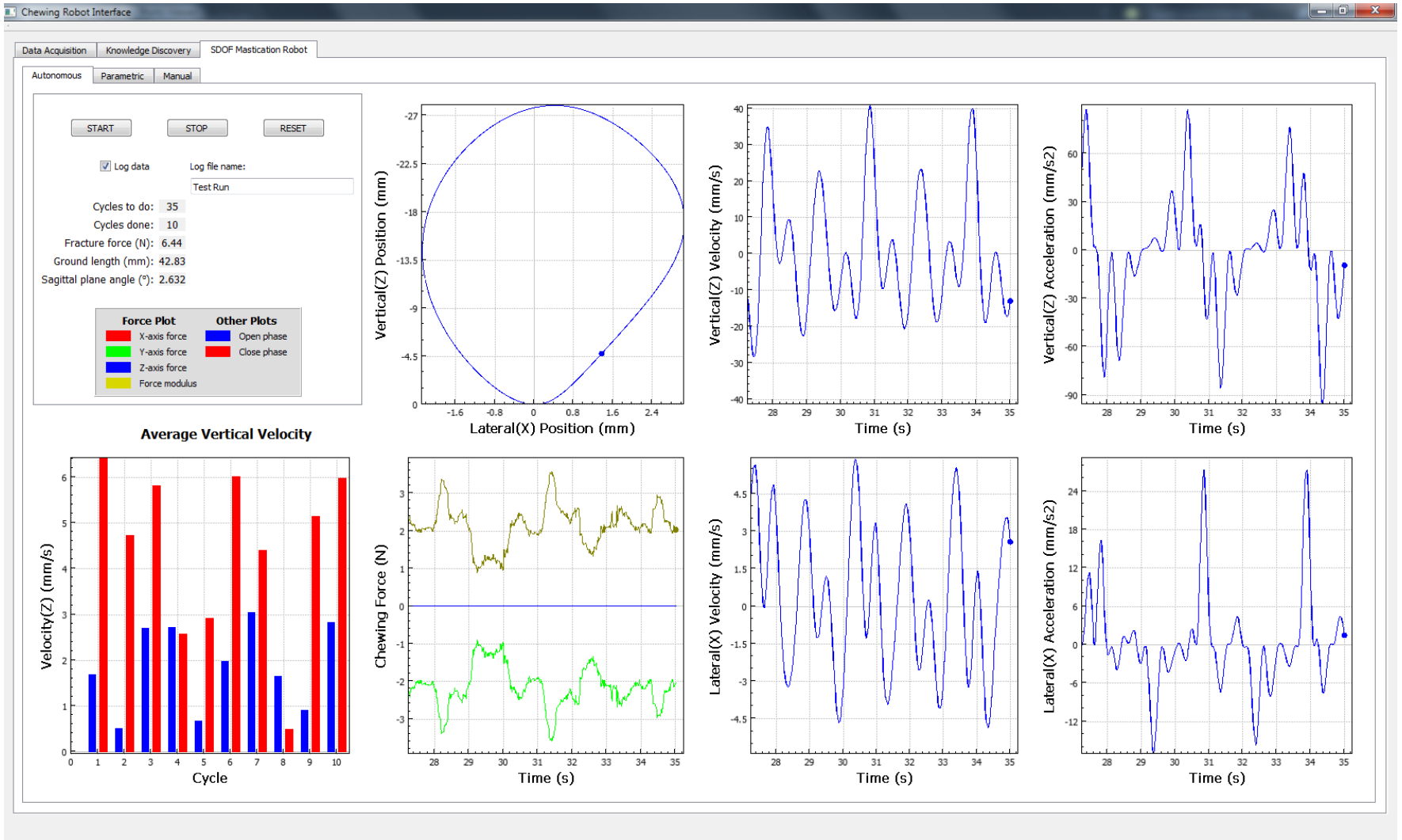


Figure 3.5: Robot Interface GUI.

The user can select the mode in which the robot will operate via the corresponding tabs on the robot interface. Each of the three tabs and their associated widgets will be explained below:

Autonomous

This tab contains widgets related to autonomous control of the robot. User interactions with the robot in this mode are limited to simple button clicks to start or stop operation; the remaining widgets provide feedback to the user as the mastication sequence proceeds. The following list explains each of the widgets in this tab as shown in Figure 3.5.

1) Control buttons

When the KBS is initially started, the robot is in *Stop* mode (an internal state); in this mode, the user is able to set the various configuration parameters of the robot as will be shown below. To change the current state of the robot and begin the mastication sequence, the following three button widgets can be used:

- *START:*
  Clicking this button switches the robot into autonomous mode and begins the mastication sequence; once started, control of the robot is transferred to the KBS.
- *STOP:*
  This button (not to be confused with the mode of the same name) terminates the mastication sequence and switches the robot into *Idle* mode (an internal state). While in idle mode, all feedback widgets remain in a 'paused' state for examination by the user; although the feedback widgets are paused, the robot itself is not considered paused hence, clicking *START* again will clear all widgets and restart the mastication sequence.
- *RESET:*
  This button switches the robot back into stop mode, clearing the feedback widgets and allowing for configuration parameters to be modified if required.

2) Data logging widgets

The purpose of the data logging widgets is to facilitate logging of data that is produced by the robot during operation; here are two widgets used for this purpose:

- *Log data checkbox:*
  This checkbox can be used to enable/disable data logging and can be toggled in real-time. While enabled (checked) data produced by the robot during operation will be stored; once the mastication sequence is complete (or otherwise terminated), the stored data will be saved to an *Excel* spreadsheet (the log file is located in the KBS directory as discussed section 3.6). An example log file is shown in Figure C.1.
- *Log file name input:*
  This is a 'line edit' widget; it allows the user to specify a string (optional) that will be appended to the file name of the Excel spreadsheet.

3) Data feedback label widgets

This collection of widgets contains labels that provide feedback to the user regarding various parameters as shown below:

- *Cycles to do:*
  This label displays the number of chewing cycles that the robot needs to complete before it will automatically stop; it is updated after the first chewing cycle and remains constant throughout the mastication sequence.
- *Cycles done:*
  This label displays the current number of chewing cycles that have been completed.
- *Fracture force (N):*
  This label displays the measured force required to fracture the food sample being chewed. The force is calculated after the first chewing cycle (first bite) is complete and remains constant throughout the mastication sequence.
- *Plot legend:*
  This is a legend for the plot widgets in the autonomous tab. The legend is colour coded to help the user interpret the information provided by the plots.

4) Average velocity plot

This is a bar graph that shows the average vertical chewing velocity (Z-Axis) per chewing cycle. The plot shows two coloured bars for each cycle, a blue bar and a red bar corresponding to the opening and closing phases respectively. The velocities are shown in millimetres per second (*mm/s*).

5) Position plot

The position plot traces a parametric curve corresponding to the trajectory followed by the robot's 'mandible molars' (end effector); the trajectory is displayed in terms of the vertical (Z-Axis) and lateral (X-Axis) displacements with the origin at *Maximum intercuspation*. The displacements are shown in millimetres (*mm*).

While the robot is in stop mode, the plot can assist the user in configuring the robot's chewing trajectory. The chewing trajectory is configured by changing the ground link length of the robot's four bar linkage (see Figure 1.5 for details); while being changed, the trajectory is displayed on the plot in real-time. The length of the ground link is only valid within a specified range (38-50 mm). If the link is adjusted outside this range, the plot curve is coloured red indicating an invalid trajectory (cannot be followed by the robot); if the link is within the range, the plot is coloured green to indicate a valid trajectory setting (this is illustrated in Figure 3.6).

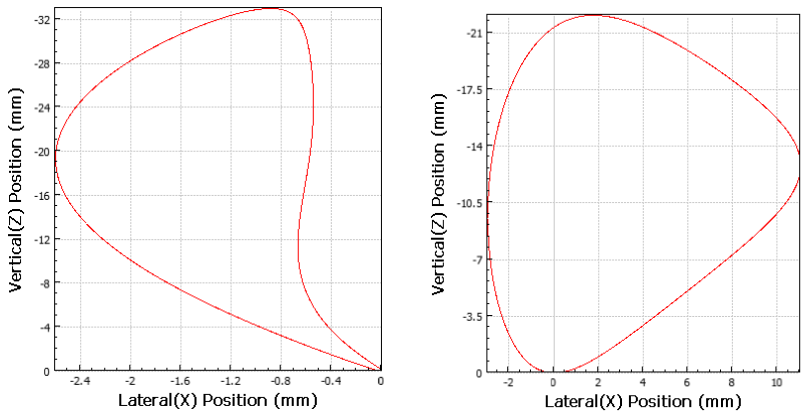


Figure 3.6: Invalid trajectories caused by (a) ground link lengths shorter than allowed and (b) ground link lengths longer than allowed.

During the mastication sequence, the plot reverts to the blue/red colour coding in order to distinguish between the opening/closing phases of the chewing cycle respectively. While in this mode, a 'dot' appears on the plot and follows the curve; the purpose of the dot is to show (in real-time) where along the trajectory the robot's end effector.

6) Force plot

The force plot shows the chewing force as measured by the robot during mastication. The force is measured using a 3-axis load cell and is displayed on the plot using four curves. Three of the curves are colour coded red, green and blue, corresponding to the

X, Y and Z axis forces respectively. The fourth curve is coloured yellow and represents the absolute magnitude of the measured force; it is calculated as the modulus of a vector whose components are the X, Y and Z axis forces. All forces on this plot are reported in Newtons (*N*).

7) Velocity plots

The velocity plots consist of two plots that show the real-time vertical and lateral chewing velocities. The plots both of the plots follow the same blue/red colour coding mentioned above in order to distinguish between open and close phase chewing velocities. The velocity values are displayed in millimetres per second (*mm/s*).

8) Acceleration plots

The acceleration plots consist of two plots that show the vertical and lateral chewing accelerations. These plots also use the blue/red colour coding mentioned above to distinguish between open/close phase accelerations. The acceleration values are displayed in millimetres per square second (mm/s^2).

Parametric

The purpose of this tab is to contain widgets related to 'parametric' control of the robot. At the time of writing, this tab has not been implemented due to time constraints; also, given that the aim of this project was *autonomous* control of the robot, the parametric features were not a priority. However, the control modes that have been discussed all serve a purpose and will be useful additions to the robot in future.

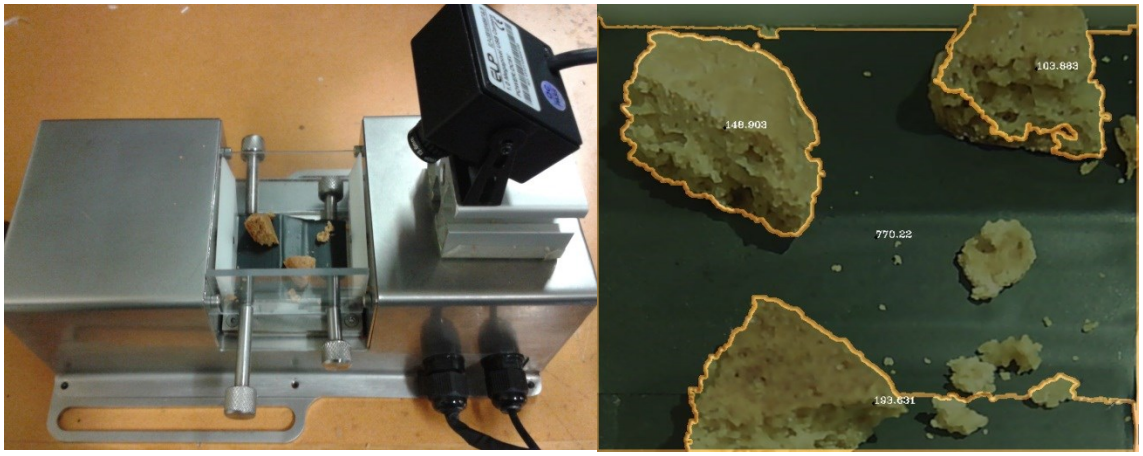
Manual

The purpose of this tab is to contain widgets that facilitate 'manual' use control of the robot. At the time of writing, this tab has not been implemented for the same reasons discussed above regarding the *Parametric* tab.

3.5.2 PSD Imaging

This tab contains widgets that display data from the robot regarding *Particle Size Distribution* (PSD). This is a new feature that has been added to the robot during this project;. Currently, the data provided by this feature consists of the following items:

- *Number of particles detected:*
  The number of food particles detected after each chewing cycle. Due to breakdown of the food sample over the course of mastication, the number of particles detected will (ideally) increase after each cycle.
- *Average particle size:*
  This data is obtained by averaging the size of all food particles detected after a chewing cycle. The average particle size is expected to reduce after each chewing cycle.
- *PSD image (with overlay):*
  This is an image provided by the robot's camera after each chewing cycle and shows the food sample currently being processed. The image has an 'overlay' that superimposes silhouettes on all detected particles; in addition, the estimated size of each particle is also shown on the image.
- *PSD image (without overlay):*
  This image is identical to the above however, the overlay is omitted for visual clarity. Sample PSD images can be seen in Figure 3.7.



(a) (b)

Figure 3.7: PSD imaging showing (a) camera sensor installed on the robot and (b) image obtained with PSD data overlaid.

The widgets required for PSD are explained as follows:

1) Enable PSD imaging checkbox

This checkbox is used to enable/disable the PSD functionality of the robot. If unchecked, all PSD functionality is disabled, no particle size information is provided and no images are produced. The checkbox can be used at any time during the operation of the robot to toggle the PSD functionality.

2) Log images checkbox

This checkbox can be used to specify whether or not the images produced by the robot are saved to the computer's hard disk. All saved images are stored in sub directories of the *PSD images* folder located in the main KBS directory tree.

3) Show image overlay checkbox

This checkbox can used to toggle the overlay of the PSD image currently displayed; this toggling can be done 'on the fly' during runtime. If the *Log images* checkbox is checked, all PSD images will be saved both with and without overlays regardless of the state of this checkbox.

4) Average particle size label

The purpose of this 'label' widget is to display the average particle size; it is updated after each chewing cycle.

5) Particle count label

This label displays the number of particles detected after each chewing cycle.

6) Cycles done label

This label displays the current number of chewing cycles that have been completed.

7) Cycles to do label

This label displays the number of chewing cycles that the robot needs to complete before it will automatically stop; it is updated after the first chewing cycle and remains constant throughout the mastication sequence.

8) PSD image display label

The purpose of this label is to display the PSD images produced by the robot; depending on the state of the *Show image overlay* checkbox (which can be toggled in real-time), images are displayed either with or without an overlay.

3.6 Robot Driver

The robot driver is a DLL file that contains the low-level control algorithms that are specific to the robot; the driver also contains code that implements the generic software interface between the robot and the KBS.

The purpose of this driver is to allow the KBS to abstract from the robot it is controlling; this is achieved through a software interface that defines a standard set of functions used by the KBS to communicate with the robot. The robot driver implements this interface by translating the function calls from the KBS into low-level commands to the robot

When the KBS software is first launched, it attempts to detect any robots that are connected to the system by searching its directory for any DLL files; the location of the DLL files in the KBS directory. If a DLL file is found, it is loaded into program memory and a tab is added to the GUI to accommodate the robot's user interface (please see section 3.5 regarding the *Robot Interface*).

In order to operate, the robot driver may need additional files such as secondary DLLs that drive the hardware components of the robot; these files are stored in subdirectory of the *Machines*. The following will discuss the structure of this directory and the files it contains:

1) Logged Data

This directory contains any spreadsheets files that the robot has saved as a result of data logging. The logging procedure has briefly been described in section 3.5.1.

2) PSD Images

This directory stores the PSD image files produced by the robot; the images are stored in subdirectories and are named according to the chewing cycle they refer to. The image files and their subdirectories are organised by the date and time which they were produced.

3) Python scripts

This directory contains the *Python* scripts required by the robot to perform various internal functions. The algorithms and code in these scripts will be discussed in detail as they are encountered in chapter 4.

4) OpenCV DLLs

OpenCV is an image processing software library; it is used by the robot to perform PSD image analysis on the food samples. To be able to access the functions in this library, the robot needs various DLL files to be present as shown here.

5) Phidget DLL

The *Phidget* DLL contains all the low-level code required to drive the hardware components of the robot; when the main robot driver DLL receives commands from the KBS software, it calls functions stored from this DLL to send hardware commands (via USB) to the robot (e.g. moving the motor, reading the sensors etc.) This DLL needs to be present at runtime in order for the robot to function.

6) Robot interface file

The robot interface file contains widgets that implement the robot specific GUI; the widgets in this file are loaded into the *Main Interface* at runtime to allow user interaction with the robot. Due to the robot interface being loaded at runtime, the functionality it provides can be customised to the robot without affecting the main KBS software.

Given the above description of the robot driver, it can be seen that the functionality of the robot is self-contained; through the use of a standard software interface that is implemented externally via a DLL, the KBS is able to communicate with any robot in isolation from its internal workings. This is a key feature of the KBS as it allows it to remain generic and not bound to the hardware it is controlling; later chapters will discuss how this is important for future work on this project.

Chapter 4

Knowledge-Based System Implementation

The overall design aspects of the *Knowledge-Based System* (KBS) and its main components have been discussed in chapter 3; the purpose of this chapter is to explain the software implementation of these design aspects and their interactions. This chapter is organised according to the major components that make up the KBS; these components are discussed in section 3.1 and are briefly listed below for convenience:

1) *Main Interface*
2) *Mastication Database* (MDB)
3) *Knowledge Database* (KDB)
4) *Robot Interface*
5) *Robot Driver*

The following sections will address the above items by first discussing the overall system design and main component interactions; subsequent sections will discuss each main component in detail and explain how it has been implemented in software.

4.1 System Overview

Throughout this section and many of the following sections, the *Unified Modelling Language* (UML) will be used to explain how the various system components interact

with each other; the following are the UML constructs used to describe various aspects of the system (please refer to the UML website[1] for more information):

- *Component Diagrams:*
  These diagrams will be used to show overall system structure, main software components involved and the interactions and/or dependencies between components.
- *Class Diagrams:*
  Class diagrams will be used to show the structure of the *Object-Oriented Programming* (OOP) classes used in the implementation.

The overall system including main software components is shown in Figure 4.1; from this figure, the user, robot and KBS interaction can be seen. The role of each component in terms of system implementation will be briefly explained:

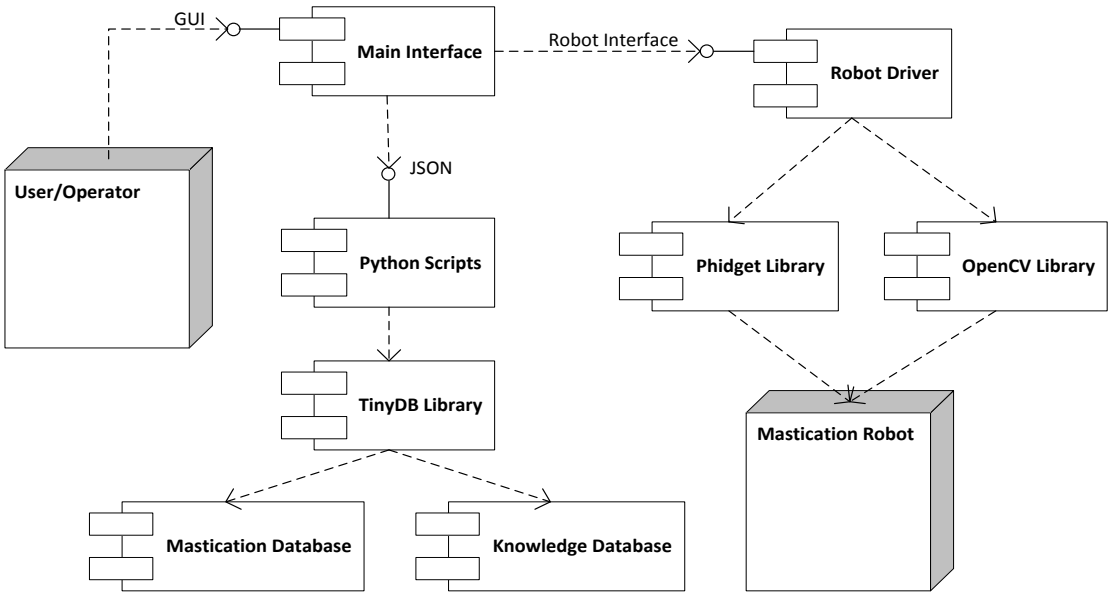


Figure 4.1: Overall software design of KBS.

- *User/Operator:*
  This node represents the 'entry point' of the user into the system. All interactions between the user and the rest of the system are performed via the *Graphical User Interface* (GUI).

[1] http://www.uml.org/

- *Main Interface:*
  This component has been written in the *C++* programming language and implements the GUI through which the user interacts with the system; it also implements the user tools required for working with the MDB and KDB. Finally, the main interface also facilitates user interaction with the robot through the *Robot Driver*.
- *Python Scripts:*
  These are scripts written in the *Python* programming language; they contain functionality required by the KBS to perform tasks such as data extraction and knowledge discovery. Interaction between Python and the main interface is performed through *Java Script Object Notation* (JSON[2]) and will be elaborated upon in later sections.
- *TinyDB Library:*
  TinyDB is a *document-oriented database* management library written in Python; it has been used in this project to implement and communicate with the MDB and KDB databases. For details regarding the use of and functionality provided by TinyDB, please refer to the official website[3].
- *MDB and KDB Databases:*
  The MDB and KDB have been implemented as document-oriented databases and are managed by the TinyDB library. The databases are stored on the host computer as JSON encoded text files.
- *Robot Driver:*
  The robot driver is a *Dynamic-Link Library* (DLL) that is written in C++; it implements the *abstract class* interface required by the main interface to communicate with the robot. The robot driver is also responsible for low-level control of the robot via the *OpenCV*[4] and *Phidget*[5] libraries.

From the components shown and described above, it can be seen that the following three are the defining components of the system in terms of specific software implementation:

[2] http://www.json.org/
[3] http://tinydb.readthedocs.org/en/latest/
[4] http://opencv.org/
[5] http://www.phidgets.com/

1) *Main Interface*
2) *Python Scripts*
3) *Robot Driver*

The remainder of the components are primarily supporting components such as external libraries that are used to supplement the core functionality. The following sections will focus on the functionality of the three components above and will discuss the specifics of their implementations.

4.2 Main Interface

This section will discuss the software implementation of the main interface; an overview of the component is given below, followed by a discussion of the relevant classes and functions.

4.2.1 Component Overview

The *Main Interface* as previously described, is GUI-based software application that the implements tools and features necessary for user interaction and database work; Figure 4.2 shows the relevant software sub components as described:

1) *GUI:*
   The GUI sub component contains classes that implement the *Main Interface* functionality; this functionality is logically divided into three categories:
   a) *Data Extraction*
   b) *Regression*
   c) *Robot Interface*
2) *Qt Framework:*
   This is a software framework written in C++ that can be used to build general purpose, cross-platform applications that require a GUI; documentation regarding Qt can be found on the official website[6].

[6] http://doc.qt.io/

3) *QCustomPlot Library:*

This is a library that extends Qt to provide plotting functionality; it is used here to implement the *Regression Plot* widget mentioned in section 3.2.2. The library is documented on the official website[7].

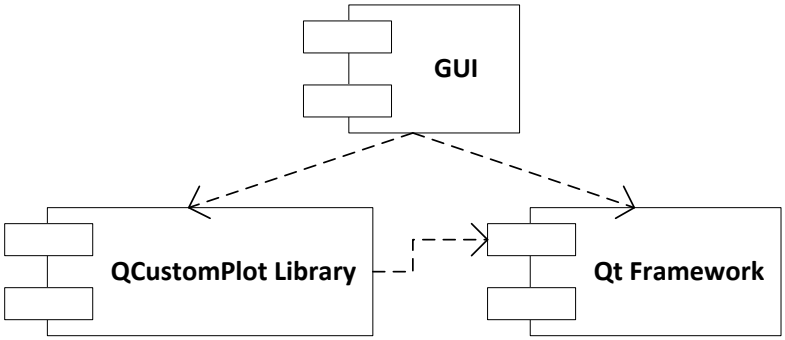


Figure 4.2: Sub components of *Main Interface*.

The following will discuss the classes used to implement the *GUI* sub component and will be organised according to the three categories shown above. References to various Python functions will be made when required; these functions will not be discussed in detail here, please see section 4.3 regarding Python functions.

4.2.2 Data Extraction

The data extraction category implements the functionality regarding extraction and storage of mastication data into the MDB; the relevant class diagram is shown in Figure 4.3. All widgets referred to are located in the *Extraction* tab of the GUI unless stated otherwise.

The *MainWindow* class handles all functionality involving the GUI widgets; it contains methods and members that implement the features discussed in section 3.2. The following will explain the methods of this class that pertain to the data extraction process.

[7] http://www.qcustomplot.com/index.php/support/documentation

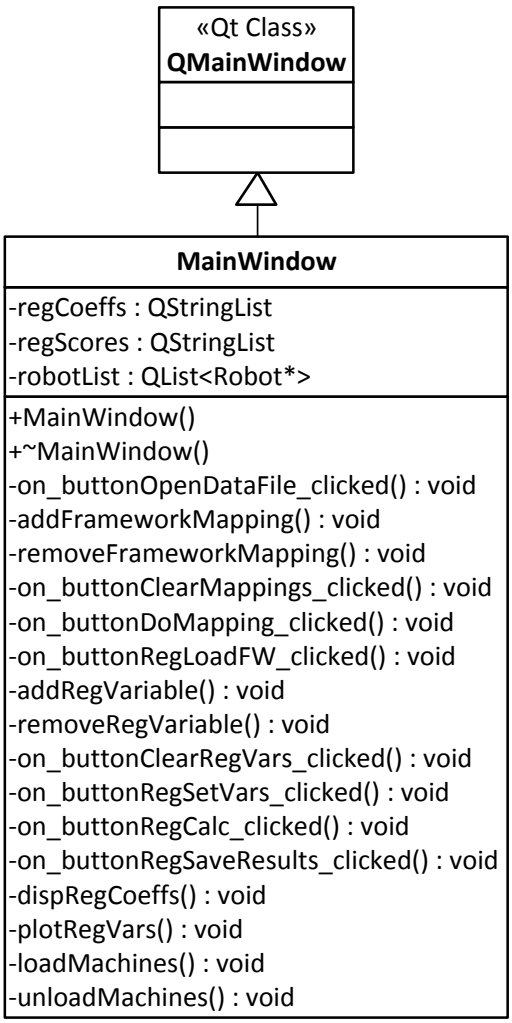


Figure 4.3: *Main Interface* implementation - *MainWindow* class.

1) on_buttonOpenDataFile_clicked

This method is called when the *Open File* button is clicked; the procedure is as follows:

1) The Windows *Open File* Dialog box is displayed for the user to select a data file.
2) The path and name of the file is encoded into a JSON document (*Path_Excel.json*) and saved in the KBS directory.
3) A Python function *excelSheetsCols* is called; it operates on Path_Excel.json and produces two other JSON files containing the sheet/column names (*Data_Excel.json*) and the *Object-oriented Framework* (OKF) fields (*Framework_DB.json*) respectively. The function returns.
4) The *Data File Tree View* and the *Mastication Database Tree* View widgets are populated according to the information from the two JSON files. The classes

responsible for *displaying* the data onto these widgets will not be discussed because their purpose is simply to implement the Qt *Model-View* framework; detailing their implementation here would detract from the core functionality of the system being described. If further information regarding the Qt Model-View Framework and its implementation is required, it can be found within the Qt documentation[8].

2) addFrameworkMapping

This method is called when an MDB field in the tree view widget is double-clicked, mapping it to the currently selected field from the data file tree view widget. The *object* and *attribute* of the selected MDB field are stored along with the selected sheet name and column header; this effectively 'tags' the data in that column for extraction.

3) removeFrameworkMapping

This method is called when a column header that has been previously mapped is double-clicked; the mapping is removed and the data in that column is no longer marked for extraction.

4) on_buttonClearMappings_clicked

This method is called when the *Clear All Mappings* button is clicked; it removes all mappings from the column headers in the data file tree view widget.

5) on_buttonDoMapping_clicked

This method is called when the *Extract Data* button is clicked and proceeds as follows:

1) All column headers in the data file tree view that have been mapped to MDB fields are iterated through; at the end of the iterations, a JSON file (*Framework_Mappings.json*) is produced containing each mapped column, the sheet it belongs to and the MDB field it has been mapped to.
2) A Python function *extractData* is called; it uses the information in Framework_Mappings.json to extract and store the data within all mapped columns into the MDB. The function returns.

[8] http://doc.qt.io/qt-5/model-view-programming.html

4.2.3 Regression

This category contains functionality for the *Multiple Linear Regression* (MLR) algorithm that is available to the user via the *Regression* tab in the GUI. The class diagram is shown in Figure 4.3 with relevant methods discussed below.

1) on_buttonRegLoadFW_clicked

This method is called when the *Load Framework* button is clicked; it populates the MDB tree view and *Regression tree view* widgets as has been previously discussed.

2) addRegVariable

This method is called when a field from the MDB is double-clicked; it designates the field as a *Target* or *Explanatory* variable for use during the MLR process. The *object* and *attribute* of the selected MDB field are stored, along with either *Target* or *Explanatory* variable depending on which one was selected at the time this function was invoked.

3) removeRegVariable

This method is called when an MDB field that has been previously mapped as a regression variable is double-clicked; the mapping is removed and the data corresponding to that field is not used during MLR.

4) on_buttonClearRegVars_clicked

This method is called when the *Clear All Variables* button is clicked; it removes all mappings from the regression tree view widget.

5) on_buttonRegSetVars_clicked

This method is called when the *Set Variables* button is clicked; its purpose is to process all mapped MDB fields for data retrieval as follows:

1) All MDB fields that have been mapped as either target or explanatory variables are iterated through; at the end of the iterations, a JSON file (*Regression_Variables.json*) is produced containing the object and attribute of each mapped field, along with any retrieval parameters/constraints that have been set.
2) A Python function *getRegVals* is called; it processes Regression_Variables.json and attempts to retrieve any compliant (with

regards to retrieval parameters) mastication data stored in the MDB. This function then produces a second JSON file (*Regression_Values.json*) that contains any data that was successfully retrieved, along with statistics regarding the number of points found, discarded and retrieved. The function returns.

3) The statistics produced by the Python function are displayed in the regression tree view widget, next to their corresponding variables.

6) on_buttonRegCalc_clicked

This method is called when the *Perform Regression* button is clicked; it proceeds as follows:

1) The regression parameters such as *polynomial order* and *interactions* are recorded and appended to *Regression_Values.json*.
2) A Python function *regression* is called; it uses the retrieved data and regression parameters stored in Regression_Values.json to perform MLR. The results of the MLR algorithm are stored in a JSON file (*Regression_Coefficients.json*). The function returns.
3) The MLR results in Regression_Coefficients.json are read; the regression coefficients and correlation coefficients are stored in the class attribute members: *regCoeffs* and *regScores* respectively.
4) The functions: *plotRegVars* and *dispRegCoeffs* are called to plot the regression fit and display the data stored in regCoeffs and regScores respectively.

7) on_buttonRegSaveResults_clicked

This method is called when the *Save Regression Results* button is clicked and calls a Python function *regSaveResults*; this function reads the MLR results in Regression_Coefficients.json and stores them into the KDB.

8) dispRegCoeffs

This function is called by *on_buttonRegSaveResults_clicked;* its purpose is to display the regression and correlation coefficients in the *Regression Results* widgets discussed in section 3.2.2.

9) plotRegVars

This function is called by *on_buttonRegSaveResults_clicked;* its purpose is plot the 'fit' or curve calculated by the MLR algorithm along with the actual data points used, in the *Regression Plot* widgets mention in section 3.2.2.

4.2.4 Robot Interface

The robot interface is an *abstract class* that defines a standard set of functions that the KBS requires to communicate autonomously with the robot; any robot connected to the KBS must implement this class in its *Robot Driver* DLL as will be shown in section 4.4. Through the use of such an interface, the functionality of the robot is encapsulated, 'decoupling' it form the KBS; this allows the use of virtually any robot, provided it can implement the required interface functionality given its hardware. Another related concept is the use of an OKF which can be thought of as an interface between the KBS and the application (i.e. Human Mastication); through the use of different robot/OKF combinations, the KBS can be used in virtually any application that requires a robot (or otherwise intelligent device) to exhibit knowledge-based autonomous behaviour.

The class diagram for the interface class *Interface* is shown in Figure 4.4; it is used by the *Main Interface* via the *Robot* class as shown in the class diagram of Figure 4.5. The procedure by which the KBS detects and communicates with an attached robot is as follows:

N.B.: For the remainder of this section, the term *robot* will be used to refer to the *Robot* class (or an object thereof) while the term *machine* will refer to the actual robotic device being controlled (and its driver/DLL).

1) The *Machines* directory is searched for any DLL files (machine drivers).
2) An object of class Robot is instantiated for each DLL found.
3) The robot object loads the machine DLL and stores it in its *module* member; a tab on the GUI is also created to accommodate the user interface specific to the loaded machine.
4) Upon being loaded by the robot object, the DLL is initialised and its implementation of the Interface class is ready for use.

5) The robot object queries the *machineData* member of the machine for details regarding the OKF it is compatible with and the OKF fields it requires for control (discussed further below).
6) The robot object attempts to search for and load any knowledge from the KDB regarding the OKF fields specified by the machine.
7) The robot object 'activates' the machine, allowing the user to interact with it and (if knowledge was loaded successfully) begin autonomous control.

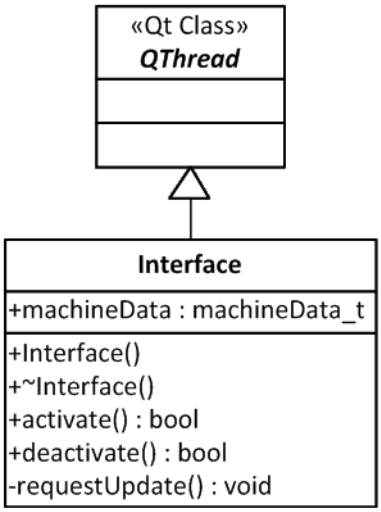


Figure 4.4: *Main Interface* implementation - *Interface* class.

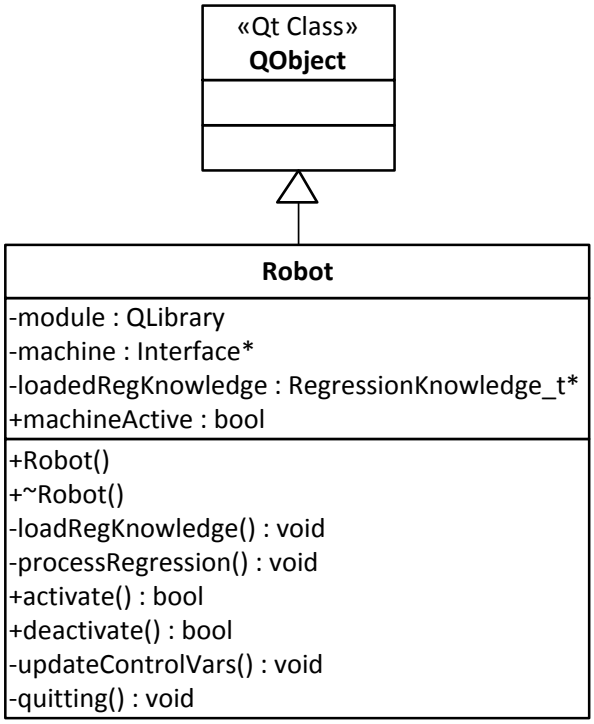


Figure 4.5: *Main Interface* implementation - *Robot* class.

To support understanding of the above procedure, explanations of the *Interface* and *Robot* classes is given below:

Interface Class

The following describes the main members and structures of the Interface class and shows how they are used during autonomous control of the machine.

1) machineData

The machineData member is a C++ structure that contains variables used by the KBS to communicate with the robot and is shown in Figure 4.6 along with two sub structures it references. The order in which these structures are used is as follows:

1) The *framework* member contains the name of the OKF that the machine is compatible with (in this case, *Human Mastication*) and is read by the robot object.
2) The *frameworkValid* Boolean variable is set by the robot to notify the machine whether or not its framework is available in the KBS. If this variable is *false*, autonomous control is not possible.
3) Two lists: *controlVars* and *feedbackVars* contain the *controlVars_t* and *feedbackVars_t* sub structures respectively; these structures contain the OKF fields that the machine requires for autonomous control. The *object* and *attribute* members of these sub structures refer to their OKF counterparts thus, identifying OKF entries as either variables for control of the machine or variables for feedback from the machine. The *data* members contain the value of the as set by the machine (in case of feedbackVars) and the robot (in case of controlVars).

To give an example of the above, consider a mastication robot with the following machineData structure:

- *framework*: Human Mastication
- *controlVar:*
    - *Object:* Mastication Sequence
    - *Attribute:* Number of Cycles
- *feedbackVar:*
    - *Object:* Food Properties
    - *Attribute:* Food Hardness

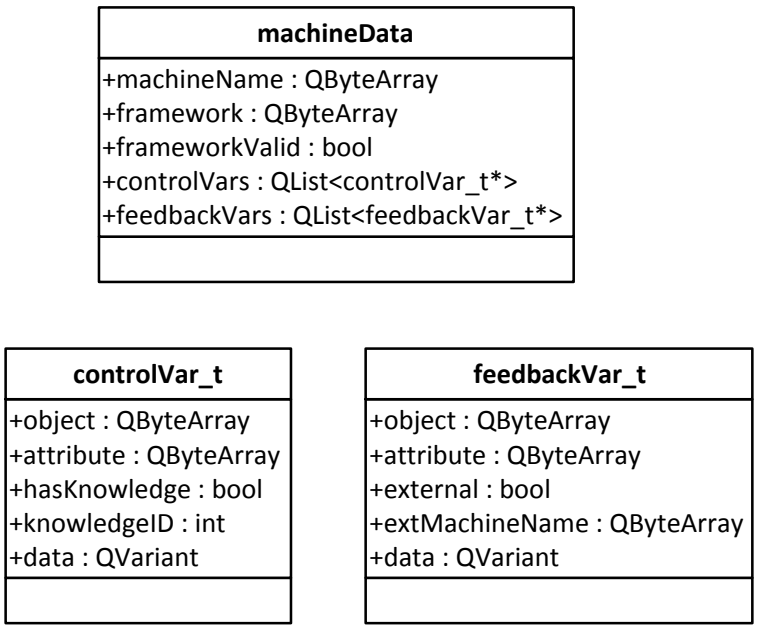


Figure 4.6: *Main Interface* implementation - *machineData* and associated structures.

Once this structure is read by the robot object, it will search the KDB for any knowledge previously obtained regarding the above variables. In the case of MLR, it will search the KDB for regression results where the target variable is *Number of Cycles* and the explanatory variable is *Food Hardness*.

If knowledge is found, it can be used to autonomously control the machine as follows:

1) Autonomous control is initiated by the user via the machine's GUI.
2) The machine performs one chewing cycle (corresponding to the first bite in humans) and measures the maximum force encountered before food fracture (hardness).
3) The magnitude of the measured force is stored in the *data* member of the feedbackVar and a request is sent to the robot for an update of the controlVar.
4) The robot applies the regression equation obtained from the KDB to the hardness value and stores the result (number of chewing cycles to be performed) in the *data* member of the controlVar.
5) The machine continues the mastication sequence and *automatically* stops once the prescribed number of chewing cycles have been completed.

Note that while this particular example only involves a pair of control and feedback variables that are only updated once, the process is applicable to any number and combination of variables that may be updated once, or continuously/periodically.

2) activate/deactivate

These methods are called when the robot attempts to activate/deactivate the machine; they are implemented by the machine DLL and perform the shutdown and start-up operations specific to the machine in question.

3) requestUpdate

This method is a Qt *Signal* and is emitted to notify the robot object of a new update request. Details regarding Qt's *Signals & Slots* mechanism can be found in the documentation[9].

Robot Class

The KBS communicates with attached machines via an object of the Robot class; a separate object is instantiated for every machine detected by the KBS. The following is description of the class implementation.

1) Robot

This is the *constructor* of the class which is called when an object is first instantiated. The robot object is created by the *loadMachines* function (please see *MainWindow* class in Figure 4.3) which is responsible for finding the DLL files. The parameters passed to this constructor include the DLL path and the tab within which the machine GUI will be accommodated; using these parameters, the DLL is loaded and initialised.

2) ~Robot

This is the *destructor* of the class; its purpose is to 'deactivate' the robot (preventing any further interactions by the user) and unload the machine DLL, freeing its memory.

3) loadRegKnowledge

This method is called once the robot has acquired the OKF details from the machine's *machineData* structure; the function searches for and loads any available knowledge from the KDB. The loaded knowledge is stored in the *loadedRegKnowledge* member.

[9] http://doc.qt.io/qt-5/signalsandslots.html

4) processRegression

This method is called by *updateControlVars*; it applies the regression coefficients to the machines feedback variables via a 'mapping' specified by the *loadRegKnowledge* Python function.

5) activate/deactivate

These methods call their *Interface* class counterparts; the current state of the machine is stored in the *machineActive* member.

6) updateControlVars

This method is a Qt *Slot* and is called after an update request is received from the machine via the *requestUpdate* signal; it forwards data from the feedback variables to other methods such as *processRegression* for processing. The *data* fields of the control variables are update by this function once all feedback variables have been processed.

7) quitting

This method is a slot that is invoked when the KBS software is about to be shutdown/terminated; it waits for the machine to complete any sensitive tasks (such as data logging), safely deactivates the machine, and allows the KBS software to terminate.

4.3 Python Scripts

The following will describe the various Python functions used in the implementation of the KBS. Figure 4.7 shows the overall system and associated libraries/packages used. The purpose of each library is briefly explained as follows:

1) *scikit-learn* Library:
   This library contains various mathematical and machine learning algorithms implemented in Python using the *NumPy* and *SciPy* packages. scikit-learn was used in this project to implement MLR.[50]
2) *OpenPyXL* Library:
   This library is designed to work with *Excel* files (mainly the new *.xlsx* file format). It is used in this project to handle data files selected by the user.

Information regarding features and functionality of the library can be obtained through the official website[10].

3) *xlrd* Library:

This Library is similar to OpenPyXL with the exception that it is used for the older Excel files such as the *.xls* format.

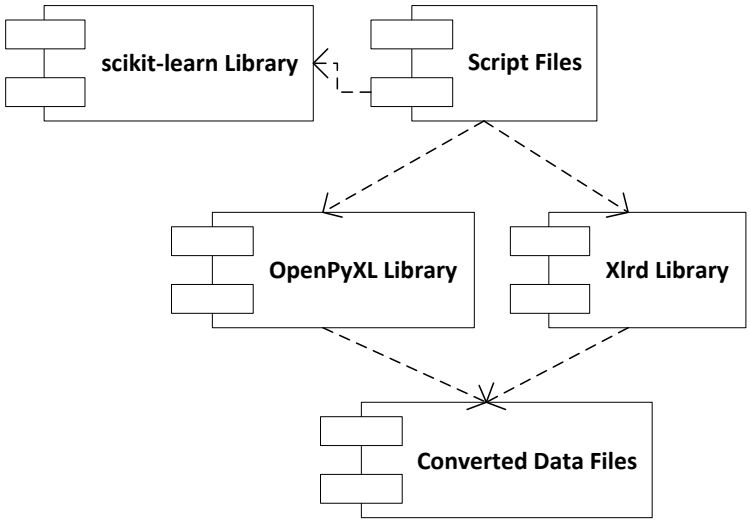


Figure 4.7: System overview of the Python functionality used by the KBS.

4.3.1 Data Extraction

The python scripts used during data extraction operations provide the following functionality:

- Opening and loading *Excel* spreadsheets that contain data to be extracted.
- Conversion for *Comma Separated Values* (CSV) and older Excel files.
- Automatic loading and creation of the OKF.

The following will list and explain the scripts used to implement this functionality.

1) ExcelSheetsCols.py

This script implements the *excelSheetsCols* function which is responsible for opening the file selected by the user, converting if necessary and then obtaining the sheet names and column headers for display on the GUI. The specific procedure implemented by this function is as follows:

[10] https://openpyxl.readthedocs.org/en/2.3.3/

1) The JSON file *Path_Excel.json* is loaded and the path of the data file is read.
2) The extension of the file is checked and if it is either *.csv* or *.xls*, the appropriate conversion function is called.
3) The function iterates across all columns in the first row of every sheet and records any cells that are not empty. Theses represent the sheet names and column headers.
4) The sheet names and column headers are encoded into a JSON file *Data_Excel.json*; this file is then read by the *Main Interface* and displayed on the appropriate tree widget.
5) Finally, if the *Framework_DB.json* file is not present in the KBS directory, it created and the OKF is automatically loaded onto the GUI.

The code for this function is shown below:

```
1.  #=============================================================================
2.  import os
3.  import time
4.  import json
5.  import openpyxl as xl
6.  #=============================================================================
7.  from convertcsv import convertCSV_F
8.  from convertedxls import convertXLS_F
9.  from framework import createFW
10. import definitions as defs
11. #=============================================================================
12.
13. #=============================================================================
14. #Function for opening data files and
15. def excelSheetsCols_F():
16.     with open(defs.jsonExcelFile, 'r') as jPathFile:
17.         path = json.load(jPathFile)['file']
18.
19.     fileExt = os.path.splitext(path)[1]
20.
21.     if(fileExt == '.csv'):
22.         dataSource = convertCSV_F(path)
23.         with open(defs.jsonExcelFile, 'w') as jPathFile:
24.             json.dump(dataSource, jPathFile, indent = True)
25.
26.     elif(fileExt == '.xls'):
27.         dataSource = convertXLS_F(path)
28.         with open(defs.jsonExcelFile, 'w') as jPathFile:
29.             json.dump(dataSource, jPathFile, indent = True)
30.
31.     elif(fileExt == '.xlsx'):
32.         srcDate = list(time.localtime(os.path.getctime(path))[0:5])
33.         dataSource = {'sourcePath': path, 'sourceDate': srcDate,
34.                       'convertedPath': path}
35.         with open(defs.jsonExcelFile, 'w') as jPathFile:
36.             json.dump(dataSource, jPathFile, indent = True)
37.
38.     dataFile = xl.load_workbook(dataSource['convertedPath'], data_only = Tru
    e)
39.
```

```
40.     sheetNames = {}
41.     columnHeaders = []
42.
43.     for sheet in dataFile.sheetnames:
44.         workingSheet = dataFile.get_sheet_by_name(sheet)
45.
46.         columnHeaders.append(sheet)
47.
48.         for col in range(1, workingSheet.max_column + 1):
49.             currentCell = workingSheet.cell(row = 1, column = col)
50.
51.             if currentCell.value != None:
52.                 columnHeaders.append(currentCell.value)
53.
54.         sheetNames[dataFile.get_index(workingSheet)] = columnHeaders[:]
55.         del columnHeaders[:]
56.
57.     with open(defs.jsonDataFile, "w") as jDataFile:
58.         json.dump(sheetNames, jDataFile, indent = True)
59. #=============================================================================
60.     if not(os.access(defs.frameworkDBFile, os.F_OK)):
61.         createFW()
62. #=============================================================================
```

2) ConvertXLS.py

This script contains the *convertXLS* function which implements the outdated *Excel* (*.xls*) file conversion. It is implemented as follows:

1) Metadata regarding the source file is generated; this includes:
    - Source file creation date
    - Source file path
    - Conversion date
    - Path of converted file within KBS directory
2) *xlrd* is used to iterate through all of the cells in the source file and copy them into a *.xlsx* file using *OpenPyXL*.
3) The converted file is saved within the KBS directory

The code for this function is shown below:

```
1.  #=============================================================================
2.  import os
3.  import xlrd
4.  import openpyxl as xl
5.  from openpyxl.styles import Alignment, Font
6.  import time
7.  from tinydb import TinyDB, where
8.  #=============================================================================
9.  import definitions as defs
10. #=============================================================================
11.
12. #=============================================================================
13. def convertXLS_F(xlsPath):
14.     srcFile = os.path.split(xlsPath)[1]
```

```
    srcDate = list(time.localtime(os.path.getctime(xlsPath))[0:5])

    convDir = '../Converted Data Files/XLS/'
    convFile = os.path.splitext(srcFile)[0] + '.xlsx'
    convPath = convDir + convFile

    convSourceDB = TinyDB(defs.convSourceDBFile)
    database = convSourceDB.table("Converted Data Files")

    sourceID = database.contains((where('sourceFile') == srcFile) &
                                 (where('sourceDate') == srcDate))

    if sourceID:
        database.close()
        return {'sourcePath': xlsPath, 'sourceDate': srcDate,
                'convertedPath': convPath}

    xlWb = xlrd.open_workbook(xlsPath)
    xlSheetNames = xlWb.sheet_names()

    xlxWb = xl.Workbook()
    xlxWs = xlxWb.active
    xlxWs.title = xlSheetNames[0]

    for sheet in xlSheetNames[1:]:
        xlxWb.create_sheet(title = sheet)

    for sheet in xlSheetNames:
        xlSheet = xlWb.sheet_by_name(sheet)
        xlxSheet = xlxWb.get_sheet_by_name(sheet)

        headerFont = Font(b = True)
        alignment = Alignment('center', 'center')

        for colNum in range(xlSheet.ncols):
            headerCell = xlxSheet.cell(row = 1, column = colNum + 1)
            headerCell.font = headerFont
            headerCell.alignment = alignment
            for rowNum in range(xlSheet.nrows):
                xlValue = xlSheet.cell_value(rowNum, colNum)
                xlxCell = xlxSheet.cell(row = rowNum + 1, column = colNum +
1)
                xlxCell.alignment = alignment
                xlxCell.value = xlValue

    xlxWb.save(convPath)
    convDate = list(time.localtime(os.path.getctime(convPath))[0:5])

    database.insert({'sourceFile': srcFile, 'sourceDate': srcDate,
                     'convertedFile': convFile, 'convertedDate': convDate})


    database.close()

    return {'sourcePath': xlsPath, 'sourceDate': srcDate,
            'convertedPath': convPath}
#==============================================================================
```

3) ConvertCSV.py

This script file contains the *convertCSV* function that is used to convert *.csv* data files to the *.xlsx* format. The code for this function is shown below:

```
#=============================================================================
import os
import time
import csv
import openpyxl as xl
from openpyxl.styles import Alignment, Font
from tinydb import TinyDB, where
#=============================================================================
import definitions as defs
#=============================================================================

#=============================================================================
def convertCSV_F(csvPath):
    srcFile = os.path.basename(csvPath)
    srcName = os.path.splitext(srcFile)[0]
    srcDate = list(time.localtime(os.path.getctime(csvPath))[0:5])

    convFile = srcName + '.xlsx'
    convPath = '../Converted Data Files/CSV/' + convFile

    convSourceDB = TinyDB(defs.convSourceDBFile)
    database = convSourceDB.table("Converted Data Files")

    sourceID = database.contains((where('sourceFile') == srcFile) &
                                 (where('sourceDate') == srcDate))

    if sourceID:
        database.close()
        return {'sourcePath': csvPath, 'sourceDate': srcDate,
                'convertedPath': convPath}

    with open(csvPath) as csvFile:
        csvData = csv.reader(csvFile)

        if(len(csvData.next()) > 1):
            delimitChar = ','
        else:
            delimitChar = ';'

    with open(csvPath) as csvFile:
        csvData = csv.DictReader(csvFile, delimiter = delimitChar)

        wb = xl.Workbook()
        ws = wb.active
        ws.title = srcName

        headerFont = Font(b = True)
        alignment = Alignment('center', 'center')

        for col, header in enumerate(csvData.fieldnames):
            headerCell = ws.cell(row = 1, column = col+1)
            headerCell.alignment = alignment
            headerCell.font = headerFont
            headerCell.value = header

        for i, entry in enumerate(csvData):
            for j, header in enumerate(csvData.fieldnames):
                xlxCell = ws.cell(row = i+2, column = j+1)
                xlxCell.alignment = alignment
                if((entry[header] == '.') or (entry[header] == ' ')):
                    xlxCell.value = None
                else:
                    try:
                        xlxCell.value = float(entry[header])
                    except ValueError:
                        xlxCell.value = entry[header]
```

```
67.
68.         wb.save(convPath)
69.
70.     convDate = time.localtime(os.path.getctime(convPath))[0:5]
71.
72.     database.insert({'sourceFile': srcFile, 'sourceDate': srcDate,
73.                      'convertedFile': convFile, 'convertedDate': convDate})
74.
75.     database.close()
76.
77.     return {'sourcePath': csvPath, 'sourceDate': srcDate,
78.             'convertedPath': convPath}
79. #=============================================================================
```

4) ExtractData.py

This script implements the *extractData* function; the purpose of this function is to read data from the source file and store it into the MDB. The implementation is shown below:

```
1.  #=============================================================================
2.  import os
3.  import time
4.  import json
5.  from tinydb import TinyDB, where
6.  import openpyxl as xl
7.  #=============================================================================
8.  import definitions as defs
9.  #=============================================================================
10.
11. #=============================================================================
12. def extractData_F():
13.     with open(defs.jsonExcelFile, "r") as jExcelFile:
14.         dataSource = json.load(jExcelFile)
15.
16.     convertedPath = dataSource['convertedPath']
17.     dataFile = xl.load_workbook(convertedPath, data_only = True)
18.
19.     with open(defs.jsonMappingsFile) as jMappingsFile:
20.         itemsToMap = json.load(jMappingsFile)
21.
22.     extractedDB = TinyDB(defs.databaseFile)
23.     database = extractedDB.table("Extracted Data")
24.
25.     sourcePath = dataSource['sourcePath']
26.     srcFile = os.path.basename(sourcePath)
27.
28.     srcDate = dataSource['sourceDate']
29.
30.     dataElements = []
31.     for item in itemsToMap:
32.         sheet = itemsToMap[item]['sheet']
33.         column = itemsToMap[item]['column']
34.         itemObj = itemsToMap[item]['object']
35.         itemAttrib = itemsToMap[item]['attribute']
36.
37.         source = database.get((where('sourceFile') == srcFile) &
38.                               (where('sourceDate') == srcDate) &
39.                               (where('sheet') == sheet))
40.
41.         if source:
```

```
42.             sourceID = source.eid
43.
44.             currentItem = database.contains((where('sourceID') == sourceID)
    &
45.                                             (where('object') == itemObj) &
46.                                             (where('attribute') == itemAttri
    b))
47.
48.             if currentItem:
49.                 continue
50.
51.         else:
52.             sourceID = database.insert({'sourceFile': srcFile, 'sourceDate':
     srcDate,
53.                                         'sheet': sheet})
54.
55.         workingSheet = dataFile.get_sheet_by_name(sheet)
56.
57.         colIndex = 0
58.         data = []
59.         for col in range(1, workingSheet.max_column+1):
60.             currentCell = workingSheet.cell(row = 1, column = col)
61.
62.             if currentCell.value == column:
63.                 colIndex = col
64.                 break
65.
66.         for rows in range(2, workingSheet.max_row + 1):
67.             currentCell = workingSheet.cell(row = rows, column = colIndex)

68.             data.append(currentCell.value)
69.
70.         dateAdded = list(time.localtime()[0:5])
71.
72.         dataElements.append({'data': data, 'object': itemObj, 'attribute': i
    temAttrib,
73.                              'sheet': sheet, 'column': column, 'added': date
    Added,
74.                              'sourceID': sourceID})
75.
76.     database.insert_multiple(dataElements)
77.
78.     extractedDB.close()
79. #============================================================================
```

4.3.2 Regression

The Python scripts used during regression have four main functions and are typically used in the order given:

1) Retrieve data to be tested from MDB.
2) Perform regression on retrieved data.
3) Save results obtained from the regression.
4) Load regression knowledge at run-time when a compatible robot is connected.

These functions and their implementations will be discussed as follows:

1) GetRegVals.py

This script implements the *getRegVals* function according to the following code:

```
#================================================================================
import json
#================================================================================
import definitions as defs
#================================================================================

#================================================================================
def getRegvals_F():
    with open(defs.jsonRegVarsFile) as jRegVarsFile:
        regVars = json.load(jRegVarsFile)

    targetVar = regVars['targetVar']
    expVars = regVars['expVars']
    constraints = regVars['constraints']

    variables = [targetVar, expVars]

    defs.getDataVals(constraints, variables)

    with open(defs.jsonRegValsFile, 'w') as jRegValsFile:
        json.dump({'variables': variables}, jRegValsFile, indent = True)
#================================================================================
```

The function getRegVals is merely a 'wrapper' function that handles JSON encoding of retrieved data; the core function that retrieves data from the MDB is *getDataVals* and is implemented as follows:

```
#================================================================================
def getDataVals(constraints, variables):
    #Open databse table
    extractedDB = TinyDB(databaseFile)
    database = extractedDB.table("Extracted Data")

    #Create list of all variables
    varList = []
    for varType in variables:
        for var in varType:
            varList.append(var)

    #Number of variables present
    numVars = len(varList)

    #Create List to hold all source IDs used (not unique)
    sourceIDList = []
    #Iterate over each variable and find all matching elements from the FW

    for var in varList:
        dataElements = database.search((where('object') == var['object']) &

                                       (where('attribute') == var['attribute']))
        #Create list of all elements which meet constraints (if any)
        elementList = []
        pointsFound = 0
        dateAddedConst = 0
        #Iterate over each element and check it against constraints

        for element in dataElements:
```

```
28.             elementData = len(element['data']) -
    element['data'].count(None)
29.             pointsFound += elementData
30.             discard = False
31.             if constraints is not None:
32.                 #CONSTRAINT: Date element added to DB
33.                 #Iterate over each date/time item and check for out of bound

34.                 dateAdded = time.mktime(tuple(element['added'] + [0, 0, 0, 0
    ]))
35.                 fromDate = time.mktime(tuple(constraints['fromdate'] + [0, 0
    , 0, 0]))
36.                 toDate = time.mktime(tuple(constraints['todate'] + [0, 0, 0,
    0]))
37.
38.                 if((dateAdded < fromDate) or (dateAdded > toDate)):

39.                         discard = True
40.                         dateAddedConst += elementData
41.
42.             #If element passes, append it to the list along with its source
    ID
43.             if not discard:
44.                 elementList.append(element)
45.                 sourceIDList.append(element['sourceID'])
46.         #Assign all elements belonging to the current variable
47.         var['data'] = elementList
48.         var['pointsFound'] = pointsFound
49.         var['dateAddedConst'] = dateAddedConst
50.
51.     noneIndexList = []
52.     elementCntList = []
53.     sourceList = []
54.     for i, source in enumerate(set(sourceIDList)):
55.         elementList = []
56.         elementCount = 0
57.         noneIndexList.append([])
58.         for var in varList:
59.             found = False
60.             for element in var['data']:
61.                 if(element['sourceID'] == source):
62.                     for j, val in enumerate(element['data']):
63.                         if(val == None):
64.                             noneIndexList[i].append(j)
65.                     elementList.append(element['data'])
66.                     elementCount += 1
67.                     found = True
68.                     break
69.
70.             if not found:
71.                 elementList.append([])
72.
73.         elementCntList.append(elementCount)
74.         sourceList.append(elementList)
75.
76.     for var in varList:
77.         var['data'] = []
78.         var['truncPoints'] = 0
79.         var['incomPoints'] = 0
80.
81.     for i, source in enumerate(sourceList):
82.         if(elementCntList[i] == numVars):
83.             for j, element in enumerate(source):
84.                 newElement = []
85.                 for k in range(len(element)):
86.                     if(noneIndexList[i].count(k) > 0):
```

```
                        if(element[k] != None):
                            varList[j]['truncPoints'] += 1

                    else:
                        newElement.append(element[k])
                varList[j]['data'] += newElement
        else:
            for j, var in enumerate(varList):
                var['incomPoints'] += len(source[j]) -
    source[j].count(None)

    for var in varList:
        if len(var['data']) > 0:
            var['minVal'] = min(var['data'])
            var['maxVal'] = max(var['data'])
           else:
               var['minVal'] = None
               var['maxVal'] = None

        database.close()
   #============================================================================
```

2) Regression.py

This script contains the *regression* function which is used to conduct MLR on retrieved data. The implementation of this function I shown below:

```
#============================================================================
import json
import numpy as np
from numpy import array as npa
from sklearn.preprocessing import PolynomialFeatures as pf
from sklearn.linear_model import LinearRegression as lr
from sklearn.pipeline import Pipeline as pl
#============================================================================
import definitions as defs
#============================================================================

#============================================================================
def regression_F():
    with open(defs.jsonRegValsFile) as jRegValsFile:
        regVals = json.load(jRegValsFile)

    variables = regVals['variables']
    polyOrder = regVals['polyOrder']
    interOnly = regVals['interOnly']

    dataList = []
    allVars = []
    for varType in variables:
        for var in varType:
            dataList.append(var['data'])
            allVars.append(var)

    regData = npa(dataList)

    for i, data in enumerate(regData[1:], 1):
        model = pl([('poly', pf(polyOrder, interOnly)), ('linear', lr(False)
    )])
        model = model.fit(data[:, np.newaxis], regData[0])
        coeffs = model.named_steps['linear'].coef_.tolist()
        rSquared = model.score(data[:, np.newaxis], regData[0])
        yVals = model.predict(data[:, np.newaxis])
```

```
36.         allVars[i]['coeffs'] = coeffs
37.         allVars[i]['score'] = rSquared
38.         allVars[i]['yVals'] = yVals.tolist()
39.
40.     model = pl([('poly', pf(polyOrder, interOnly)), ('linear', lr(False))])

41.     model = model.fit(np.transpose(regData[1:]), regData[0])
42.     coeffs = model.named_steps['linear'].coef_.tolist()
43.     rSquared = model.score(np.transpose(regData[1:]), regData[0])
44.
45.     regCoeffs =  {'variables': variables, 'coeffs': coeffs, 'score': rSquare
    d,
46.                   'polyOrder': polyOrder, 'interOnly': interOnly}
47.
48.     with open(defs.jsonRegCoeffFile, 'w') as jRegCoeffFile:
49.         json.dump(regCoeffs, jRegCoeffFile, indent = True)
50. #============================================================================
```

3) RegSaveResults.py

This script file contains the *regSaveResults* function which is used to save results obtained from regression into the KDB. The function implementation is shown below:

```
1.  #============================================================================
2.  import time
3.  import json
4.  from tinydb import TinyDB, where
5.  #============================================================================
6.  import definitions as defs
7.  #============================================================================
8.
9.  #============================================================================
10. def regSaveResults_F():
11.     knowledgeDB = TinyDB(defs.knowledgeDBFile)
12.     database = knowledgeDB.table("Human Mastication")
13.
14.     with open(defs.jsonRegCoeffFile, "r") as jRegCoeffFile:
15.         knowledgeData = json.load(jRegCoeffFile)
16.
17.     targetVar = knowledgeData['variables'][0][0]
18.     expVars = knowledgeData['variables'][1]
19.
20.     target = {'object': targetVar['object'],
21.               'attribute': targetVar['attribute']}
22.
23.     expVarList = []
24.     expVarDataCnt = 0
25.     for var in expVars:
26.         expVarDataCnt += len(var['data'])
27.         expVarList.append({'object': var['object'],
28.                            'attribute': var['attribute']})
29.
30.     algorithm = 'regression'
31.     dataCount = len(targetVar['data']) + expVarDataCnt
32.     coeffs = knowledgeData['coeffs']
33.     score = knowledgeData['score']
34.     polyOrder = knowledgeData['polyOrder']
35.     interOnly = knowledgeData['interOnly']
36.
37.     dateAdded = list(time.localtime()[0:5])
38.
39.     existingItem = database.get((where('targetVar') == target) &
40.                                 (where('expVars') == expVarList) &
```

```
                                (where('algorithm') == algorithm))

    if existingItem:
        database.update({'dataCount': dataCount, 'coeffs': coeffs,
                         'dateAdded': dateAdded, 'score': score,
                         'polyOrder': polyOrder, 'interOnly': interOnly},
                          eids = [existingItem.eid])
        database.close()
        return

    knowledgeItem = {'targetVar': target, 'expVars': expVarList,
                     'algorithm': algorithm, 'dataCount': dataCount,
                     'coeffs': coeffs, 'score': score, 'dateAdded': dateAdde
d,
                     'polyOrder': polyOrder, 'interOnly': interOnly}

    database.insert(knowledgeItem)
    database.close()
#==============================================================================
```

4) *LoadRegKnowledge.py*

This script contains functions that are used on start up to retrieve regression knowledge and 'map' it to the *feedback variables* being used. The following is the implementation of the *loadRegKnowledge* function:

```
def loadRegKnowledge_F():
    with open(defs.jsonRegKnowledge, "r") as jRegKnowledge:
        machineRegKnowledge = json.load(jRegKnowledge)
        machineFW = machineRegKnowledge["framework"]
        machineCVars = machineRegKnowledge["controlVars"]
        machineFVars = machineRegKnowledge["feedbackVars"]

    knowledgeDB = TinyDB(defs.knowledgeDBFile)

    if not machineFW in knowledgeDB.tables():
        machineRegKnowledge["validFW"] = False

        with open(defs.jsonRegKnowledge, "w") as jRegKnowledge:
            json.dump(machineRegKnowledge, jRegKnowledge, indent = True)
            return

    machineRegKnowledge["validFW"] = True

    database = knowledgeDB.table(machineFW)
    fwRegKnowledge = database.search(where("algorithm") == "regression")

    numRegLoaded = 0
    for mCVar in machineCVars:
        mCVar["hasKnowledge"] = False
        coeffMapping = []
        numMapFVars = []
        for knowledge in fwRegKnowledge:
            fwCVar = knowledge["targetVar"]
            if not((mCVar["object"] == fwCVar["object"]) and
                   (mCVar["attribute"] == fwCVar["attribute"])):
                       continue

            fwFVars = knowledge["expVars"]
            fVarsFound = True
            fVarList = []
            for fwFVar in fwFVars:
```

```
                try:
                    fVarList.append(machineFVars.index(fwFVar))
                except ValueError:
                    fVarsFound = False
                    break

            if not fVarsFound:
                continue

            numRegLoaded += 1
            mCVar["hasKnowledge"] = True
            mCVar["polyOrder"] = knowledge["polyOrder"]
            mCVar["interOnly"] = knowledge["interOnly"]
            mCVar["regCoeffs"] = knowledge["coeffs"]
            mCVar["numRegCoeffs"] = len(knowledge["coeffs"])
            mCVar["score"] = knowledge["score"]
            mCVar["numDataPoints"] = knowledge["dataCount"]

            mapCoeffs(knowledge["polyOrder"], knowledge["interOnly"],
                      fVarList, coeffMapping, numMapFVars)
            mCVar["mapping"] = coeffMapping
            mCVar["numMapFVars"] = numMapFVars

            break

    machineRegKnowledge["numRegLoaded"] = numRegLoaded

    with open(defs.jsonRegKnowledge, "w") as jRegKnowledge:
        json.dump(machineRegKnowledge, jRegKnowledge, indent = True)

    database.close()
```

The regression coefficient mapping function *mapCoeffs*:

```
def mapCoeffs(degree, interactionOnly, varList, mapping, numMapFVarList):
    numFVars = len(varList)
    appendList = []
    appendList.append([])

    for power in range(1, degree + 1):
        pointers = range(power)
        appendList.append([])

        while pointers[-1] < numFVars:
            for pointer in pointers:
                appendList[power].append(varList[pointer])

            pointers[-1] += 1

        if len(pointers) > 1:
            pointers[-1] = pointers[-2] + 1
        else:
            continue

        pointers = [pointer + 1 for pointer in pointers]
        while pointers[-1] < numFVars:
            while pointers[-1] < numFVars:
                for pointer in pointers:
                    appendList[power].append(varList[pointer])

                pointers[-1] += 1

            pointers[-1] = pointers[-2] + 1
            pointers = [pointer + 1 for pointer in pointers]
```

```
31.
32.         if interactionOnly:
33.             continue
34.
35.         for var in range(numFVars):
36.             for pointer in pointers:
37.                 appendList[power].append(varList[var])
38.
39.     mapping.append([])
40.     numMapFVarList.append(0)
41.     for power, ls in enumerate(appendList[1:], 1):
42.         start = 0
43.         end = start + power
44.         while end <= len(ls):
45.             tmp = ls[start:end]
46.             mapping.append(tmp)
47.             numMapFVarList.append(len(tmp))
48.             start += power
49.             end += power
```

4.4 Robot Driver

This section will discuss the implementation of the robot driver DLL file; Figure 4.8 shows the interactions between the robot driver DLL and the various third party libraries used.

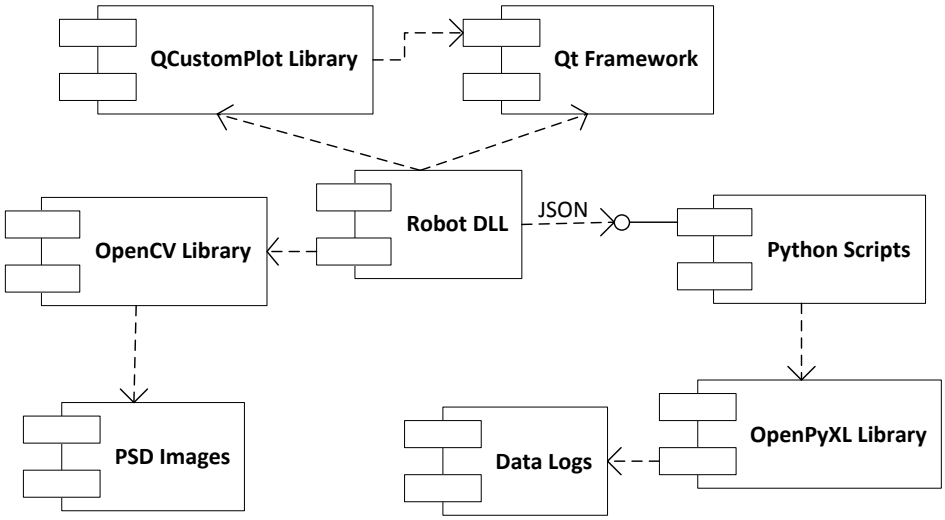


Figure 4.8: Robot Driver DLL showing interactions between external libraries.

1) Qt Framework

The Qt framework is required to implement the robot interface GUI (please see Figure 3.5); it contains the necessary widgets and tools that can be used to provide the user with a graphical interface to the mastication robot being used.

2) QCustomPlot Library

The QCustomPlot library is used to implement the various plot widgets used by the robot interface.

3) *OpenCV* Library

The OpenCV library contains various machine vision algorithms that are used to implement the PSD features of the robot.

4) Python Scripts

The use of Python in the implementation of the KBS has been discussed in the previous section; Python is also used (in conjunction with *OpenPyXL*) by the robot driver to implement its spreadsheet data logging features.

5) Robot DLL

The robot DLL contains the specific low-level code that controls the robot. For example, during autonomous control, the robot measures the 'hardness' of the food and relays the measurement to the KBS with a request for its chewing speed to be updated. The KBS applies regression coefficients to the measured value to obtain a desired chewing speed (based on human masticatory knowledge); when this speed is relayed back to the robot, the robot DLL must issue the necessary low level commands to drive the robot's hardware in such a way that this 'chewing velocity' is achieved (this typically involves PID control of motors etc.

Chapter 5

System Validation

This chapter will discuss the results obtained from the use of the *Knowledge-Based System* (KBS) to autonomously control the mastication robot; the chapter will be organised according to the following sections:

1) *Data Extraction*:
   Mastication data obtained from various sources will be discussed; pre-processing of the data for use by the KBS will also be discussed. Finally, extraction of the data from the (processed) source files and into the *Mastication Database* (MDB) will be shown.
2) *Knowledge Discovery*:
   The knowledge obtained from data in the MDB will be shown including how it is stored in the *Knowledge Database* (KDB).
3) *Autonomous Control*:
   The final section in this chapter will discuss whether or not the KBS was able to successfully control the mastication robot.

5.1 Data Extraction

As previously mentioned, many researchers in various fields such as the food and health sciences are interested in human mastication; as a consequence, data regarding human masticatory behaviour is readily available from the various experiments that have been conducted.

During this project, several researchers and scientists contributed by supplying data regarding human masticatory parameters and varying food properties. The data obtained was produced mainly form experiments involving human panelists, fitted with electronic recording equipment. The main dataset used during this project was obtained from an experiment regarding hardness of model food gels [51]. Although this experiment was conducted using both, *Agar* and *Carrageenan* gels, only Agar will be discussed due to difficulties in obtaining fracture forces for Carrageenan. The following sections will discuss how the data was pre-processed, followed by its extraction into the MDB.

5.2 Pre-Processing

Data obtained from the experiment was first pre-processed using the *SPSS*[11] statistical software package; the aim of the pre-processing was to condition the data for extraction into the MDB. The following lists the variables in the data that were used to control the robot:

- *Control Variables:*
    1) *Number of chewing cycles*:
       This is the average number of chewing cycles performed by the panelists for each food sample; it serves as the stopping condition for autonomous control of the robot.
    2) *Average open-phase chewing velocity:*
       This is the average open-phase chewing velocity recorded from the panelists during the mastication sequence.
    3) *Per cycle open velocity:*
       This is calculated as the percentage of average open-phase chewing velocity as it varies with each chewing cycle; it is used to specify the opening velocity of the robot during each cycle.
    4) *Average close-phase chewing velocity:*
       Same as (2) but for close-phase velocity.
    5) *Per cycle close velocity:*
       Same as (3) but for close-phase velocity.

[11] http://www-01.ibm.com/software/analytics/spss/

- *Feedback Variables:*
    1) *Food hardness:*

       This is the 'hardness' of the food sample, as calculated form mechanical tests that were performed on the samples during the experiment; it is measured by the robot as the force required to fracture the food during the first bite/cycle.
    2) *Percentage of cycles complete:*

       This is calculated by the robot as the ratio of *cycles done* to *total cycles to do* and is used during calculation of the per cycle velocities above.

The raw data in the file was recorded using fourteen panelists, each with three trials per sample; therefore, in order to obtain representative values for the variables above, the raw data was pre-processed for outlier removal and averaging; the results of the pre-processing are shown in Figure 5.1 to Figure 5.5. From these figures, it can be seen that there are strong polynomial (quadratic) relationships between the variables in question meaning, it is possible for *Multiple Linear Regression* (MLR) to acquire knowledge from the pre-processed data.

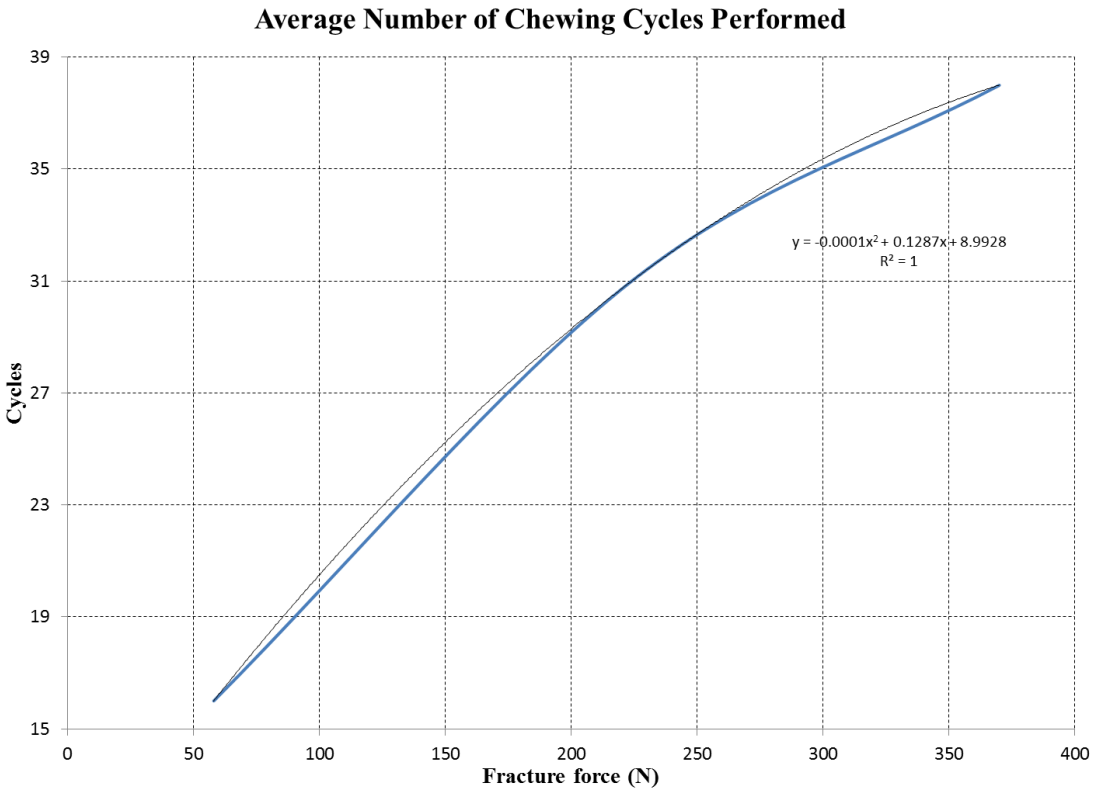


Figure 5.1: Relationship between the *average number of chewing cycles performed* and *fracture force* during human mastication of Agar gels.

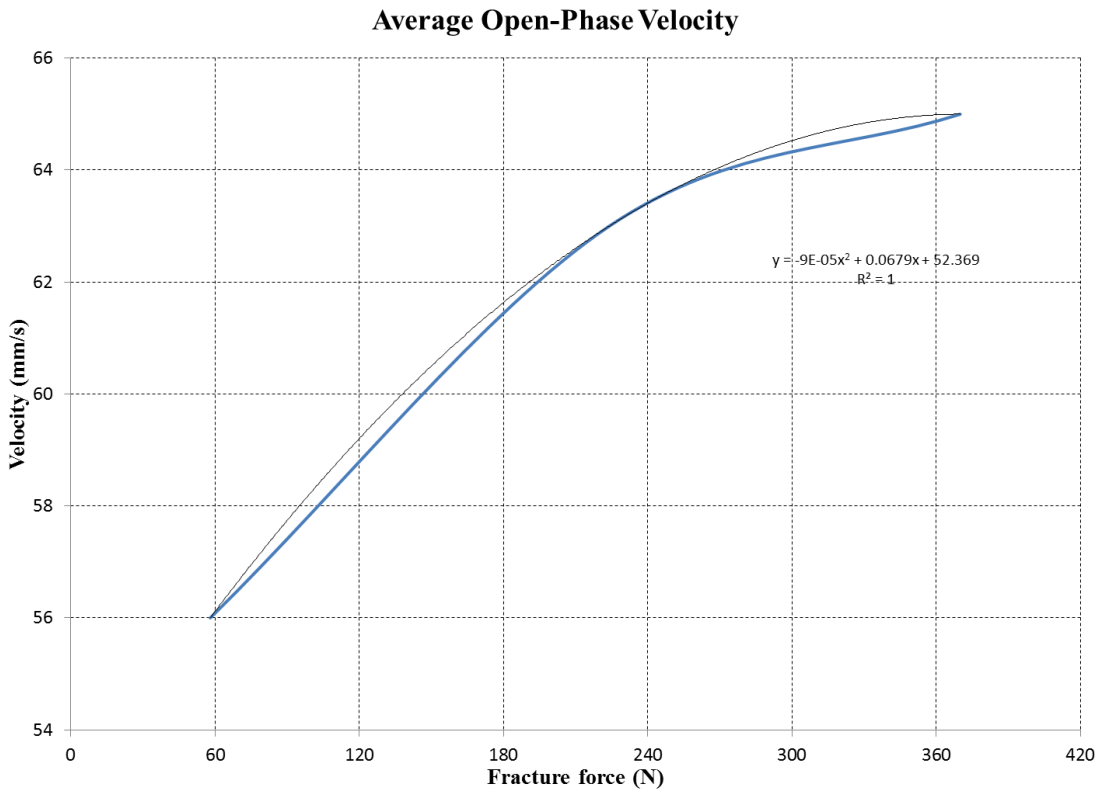


Figure 5.2: Relationship between the *average open-phase velocity* and *fracture force* during human mastication of Agar gels.

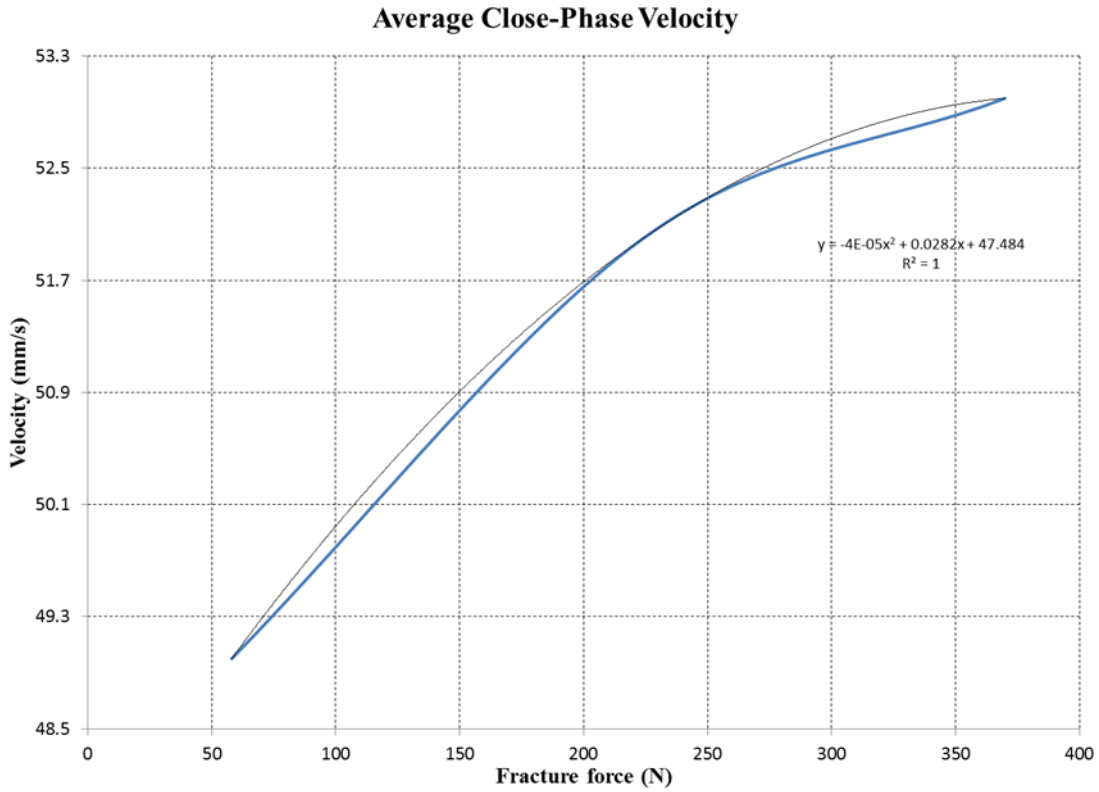


Figure 5.3: Relationship between the *average close-phase velocity* and *fracture force* during human mastication of Agar gels.

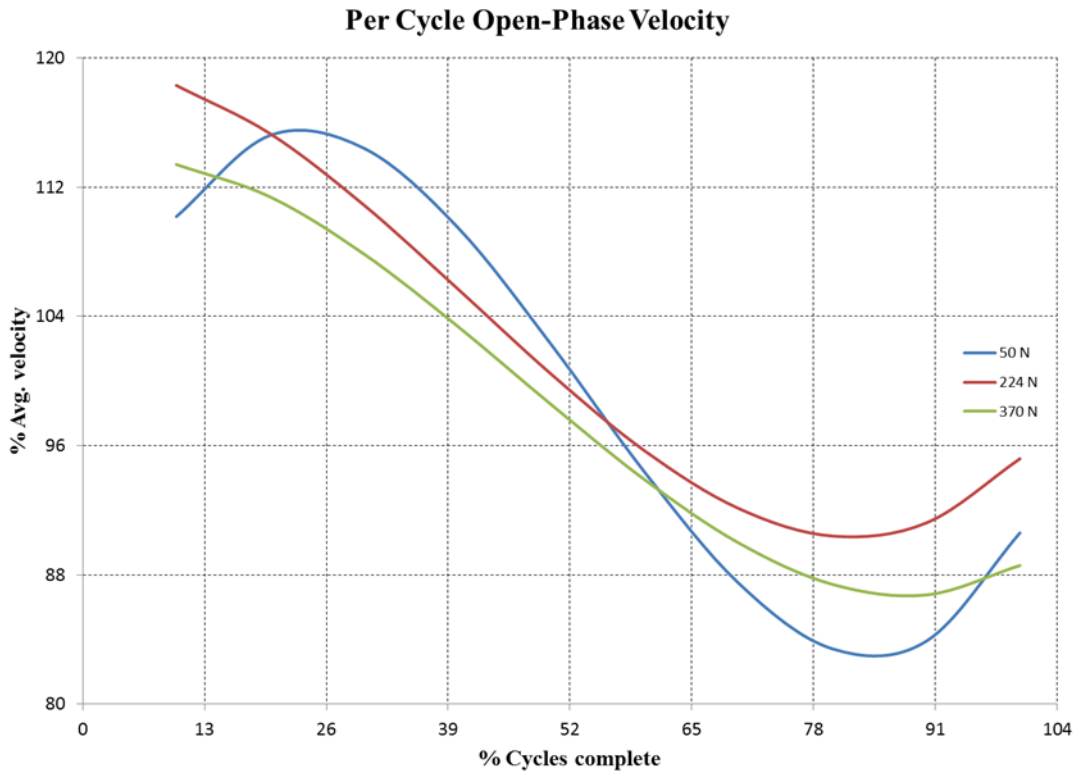


Figure 5.4: Relationship between the *per cycle open-phase velocity* and *percentage of cycles complete* during human mastication of Agar gels.

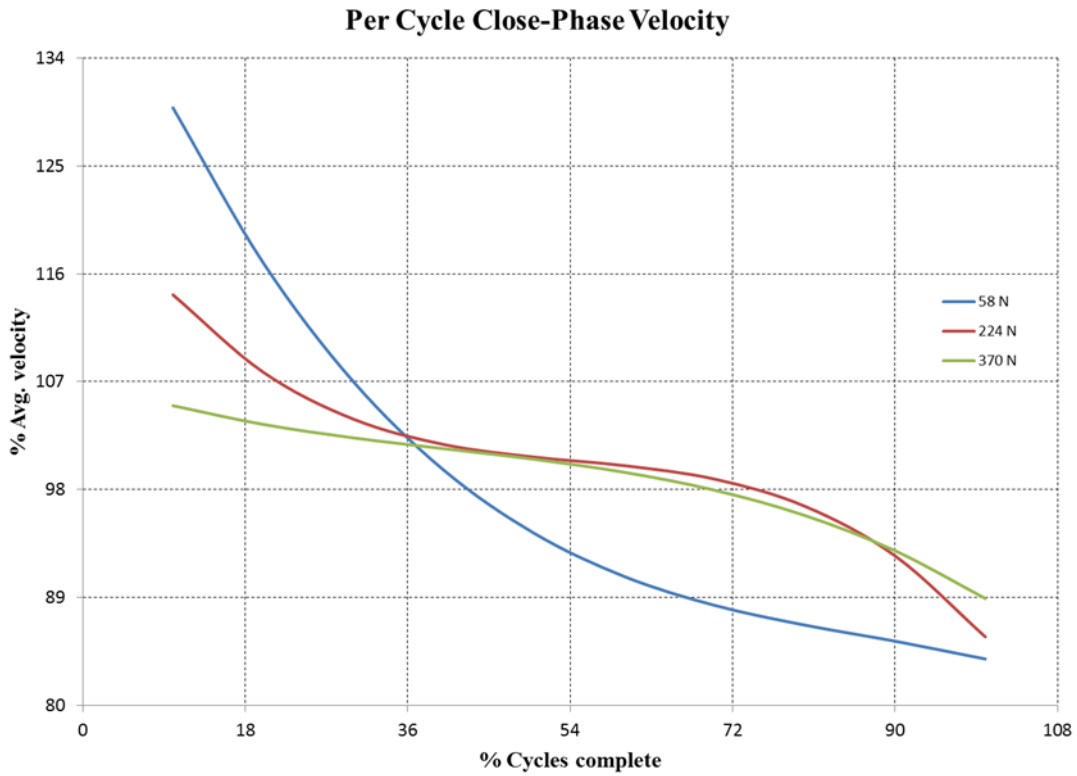


Figure 5.5: Relationship between the *per cycle close-phase velocity* and *percentage of cycles complete* during human mastication of Agar gels.

5.3 MDB Storage

Data extracted from the pre-processed file was stored in the MDB according to its associated *Object-oriented Knowledge Framework* (OKF) fields. A portion of the MDB database file is shown in Figure 5.6.


```
{
 "_default": {},
 "Extracted Data": {
  "1": {
   "sourceDate": [
    2016,
    2,
    29,
    2,
    6
   ],
   "sourceFile": "Human Masticaton Data.xlsx",
   "sheet": "Num cycle Data"
  },
  "3": {
   "added": [
    2016,
    2,
    29,
    4,
    54
   ],
   "sheet": "Num cycle Data",
   "column": "Fracture force",
   "attribute": "Hardness (N)",
   "object": "Food properties",
   "sourceID": 1,
   "data": [
    58,
    224,
    370
   ]
  },
  "2": {
   "added": [
    2016,
    2,
    29,
    4,
    54
   ],
```


Figure 5.6: Portion of MDB with extracted data.

5.4 Knowledge Discovery

Once the raw data regarding human mastication had been pre-processed and extracted into the MDB, knowledge was obtained via MLR. A sample of the knowledge as it is stored in the *Knowledge Database* (KDB) is shown in Figure 5.7.

From examination of Figure 5.7, it can be seen that the data stored in the MDB was successfully retrieved via the OKF fields (as shown by the *object* and *attribute* fields in the KDB).

```
{
 "Human Mastication": {
  "1": {
   "expVars": [
    {
     "attribute": "Hardness (N)",
     "object": "Food properties"
    }
   ],
   "dateAdded": [
    2016,
    2,
    29,
    5,
    3
   ],
   "dataCount": 6,
   "algorithm": "regression",
   "polyOrder": 2,
   "interOnly": false,
   "targetVar": {
    "attribute": "Linear open velocity (mm/s)",
    "object": "Z axis for 2-phase cycle"
   },
   "score": 1.0,
   "coeffs": [
    52.36869501187143,
    0.06790121497581325,
    -9.125014282630566e-05
   ]
  }
 },
 "_default": {}
}
```


Figure 5.7: The knowledge database showing MLR knowledge between *open velocity* and *food hardness*.

5.5 Autonomous Control

Once the KDB was populated with knowledge, the robot was tested in autonomous mode using various food samples to produce a range of fracture forces (~25 N, ~125 N and 300 N). The purpose of the test was to determine whether or not the robot would be able to conduct an entire mastication sequence without user intervention.

The robot behaved as expected according to the following:

1) The fracture force of the food was calculated after the first cycle as the maximum value measured by the load cell.
2) The *Hardness* feedback variable was populated with the fracture force and a request for update was sent to the KBS.

3) The KBS was able to successfully apply the knowledge obtained from the mastication; the control variables: *number of chewing cycles* and *vertical chewing velocity* were successfully populated and sent to the robot.
4) The robot successfully stopped chewing after the prescribed number of cycles were complete for all fracture forces. The velocity modulation however, was not as accurate as desired; this was due to inadequately tuned PID control parameters and not an issue with the KBS (because PID is a low-level functionality specific to the robot).

Overall, the autonomous control tests were considered a success; due to time constraints, PID parameters of the robot were not adequately tuned resulting in some undesirable performance. This is a low-level issue and does not compromise the KBS in any way; the PID may be tuned in future work to rectify the problem as necessary.

Chapter 6

Conclusion

This project was part of a research effort to improve the current methods used in *Food Texture Analysis* (FTA). Initial contributions to this research focussed on implementing an 'instrumental' alternative to the traditional sensory panels; the purpose of this instrument was to address the problem of inherent variability associated with human sensory perception. The results of these prior contributions lead to the development of a mastication robot that is capable of accurately emulating human masticatory behaviour, while being simple to control. The robot however, suffered from a significant flaw in that it was unable to adapt its behaviour to food properties as they varied during mastication.

The goal of this project was to address the aforementioned shortcomings of the mastication robot; this was achieved by the design and development of a *Knowledge-Based System* (KBS). The purpose of the KBS was to enable the robot to autonomously adapt its chewing patterns, through knowledge of human masticatory behaviour. The general procedure by which this was achieved required the KBS to 'extract' human mastication data from various source files, for storage into its *Mastication Database* (MDB). Data stored in the MDB would then be converted to knowledge (via regression) and used to control the robot.

The KBS that was implemented allows the user to interact with the system and robot, through a *Graphical User Interface* (GUI). This GUI provides various tools for the user to perform the necessary functions such data extraction and knowledge discovery. Communications between the robot and KBS were conducted through an *abstract class*, allowing low-level robot functionality to be isolated from the KBS.

In addition to the implementation of a KBS, an additional feature was added to the robot; an embedded camera for machine vision applications was installed. This camera enables the robot to acquire images of the food being chewed between cycles; these images are then processed to obtain *Particle Size Distribution* (PSD) information. PSD is a valuable tool and will be point of focus for future work in this area.

Finally, the robot was tested under KBS control to evaluate its ability to operate without human intervention. The tests showed positive results with the robot and KBS communicating via the abstract interface. Control of the robot was achieved through generic (can be modified for use in any application) *control* and *feedback* variables based on an *Object-oriented Knowledge Framework* (OKF) The robot, under KBS control was able to exhibit adaptive behaviour by varying its velocity over the course of mastication; it was also able to automatically stop when the required number of cycles were complete. The only negative outcome was that the actual velocity profile of the robot's end effector, did not match what was predicted by the data very well; this was due to inadequate PID tuning (due to time constraints) and had no bearing on the KBS system. Future work can remedy the PID issues as required.

In summary, this project was considered successful in that it achieved the overall goal of enabling autonomous control of the mastication robot. The system was developed using industry recognised software that is *open-source* and *cross-platform*; this allows the KBS software to be used effectively in a research and/or commercial setting. Furthermore, through the use of an OKF and abstract robot interface class, the KBS system can be extended to work with virtually any intelligent machine needing to exhibit adaptive/autonomous behaviour. In closing, it is important to note that while this project is 'a step in the right direction', it is still very much a work-in-progress; significant development is still required before such a system can be considered a viable alternative to traditional sensory methods.

Chapter 7

Future Work

In addition to what has been implemented during this project, several additions can be made to improve the overall system in future; suggestions regarding improvements to the *Knowledge-Based System* (KBS) and robot are as follows:

7.1 KBS Improvements

The KBS software can benefit from several improvements to its core features:

Data Acquisition

- Pre-processing features can be added to reduce dependency on external software packages such as *SPSS*. Although this may initially appear as a 're-invention of the wheel', it can be beneficial in future when dealing with highly specialised data and/or algorithms for which general software packages are not suitable.
- The *Data Extraction tab* can be upgraded to include tools for source file manipulation; it would be useful for the user to have a preview of the data that is about be stored in the database for example.

Knowledge Discovery

- This feature can benefit from the addition of more machine learning algorithms. While *regression* is very powerful, versatile and used extensively throughout this project, other algorithms such *clustering* and *Artificial Neural Networks* (ANN) are a welcome addition to any machine learning arsenal given their success in *unsupervised learning*.

Robot Interface

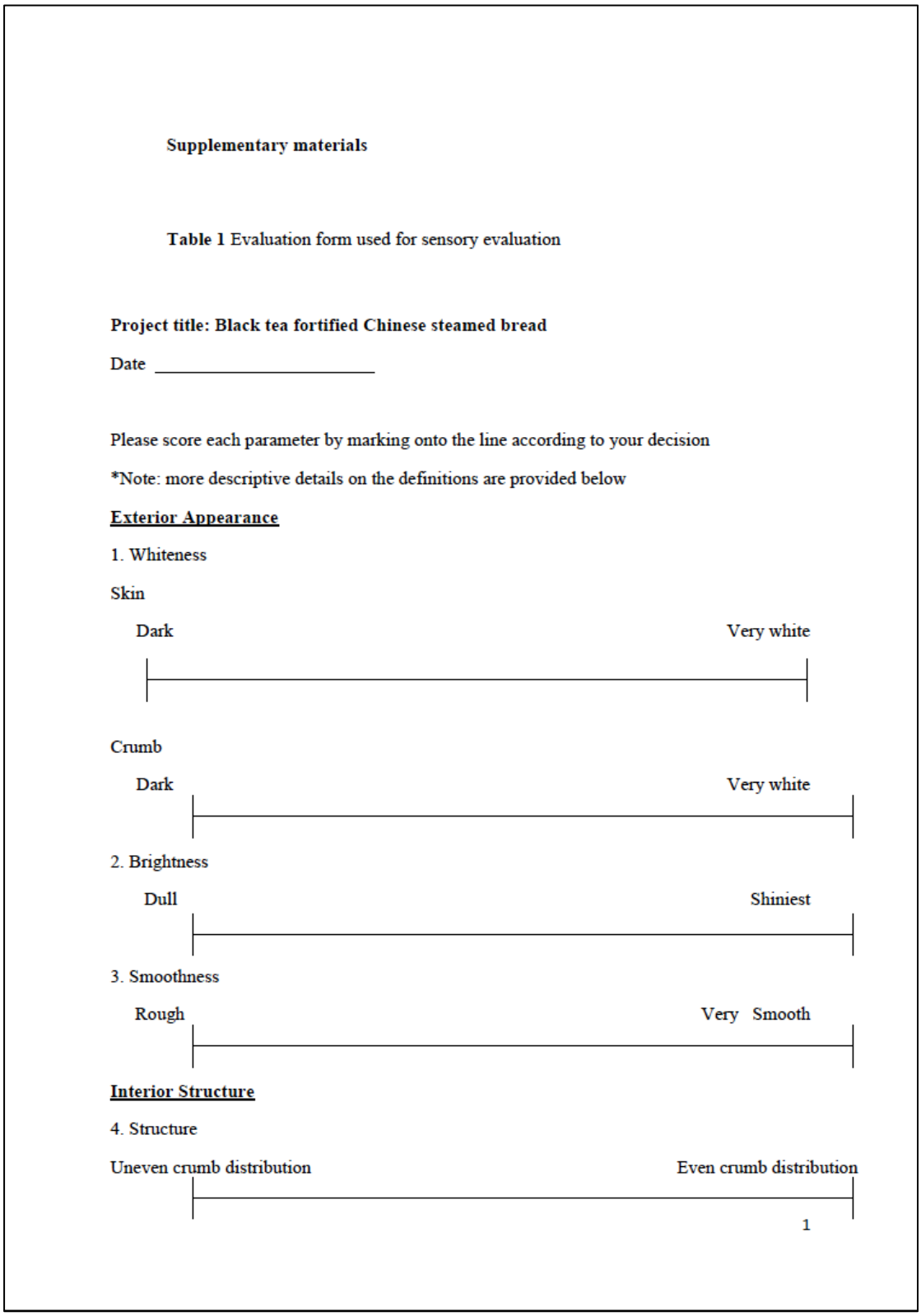
A possible improvement in this area would be to add capability for a *simulated* robot to be connected to the KBS. This can be achieved through software packages such as *SIMULINK* which are able to produce *DLL* files that implement simulated mechanical models. This would enable features such as the ability to experiment with algorithms on robot *models* without requiring a physical device connected. An example of a SIMULINK model for the robot used during this project can be seen in Figure B.1.

7.2 Robot Improvements

In addition to the *Particle Size Distribution* (PSD) feature added to the robot, the following improvements would also improve the performance of the robot:

- Saliva secretion mechanism for bolus formation and more realistic food sample deformation during mastication.
- Improved 'tongue' capability that can interact with the food rather than merely reposition it (as in the current system using servo motors)

Appendix A



Supplementary materials

Table 1 Evaluation form used for sensory evaluation

Project title: Black tea fortified Chinese steamed bread

Date _____

Please score each parameter by marking onto the line according to your decision

*Note: more descriptive details on the definitions are provided below

Exterior Appearance

1. Whiteness

Skin

Dark — Very white

Crumb

Dark — Very white

2. Brightness

Dull — Shiniest

3. Smoothness

Rough — Very Smooth

Interior Structure

4. Structure

Uneven crumb distribution — Even crumb distribution

1

Figure A.1: Sample scoring sheet for sensory evaluation (reproduced from [33]).

Appendix B

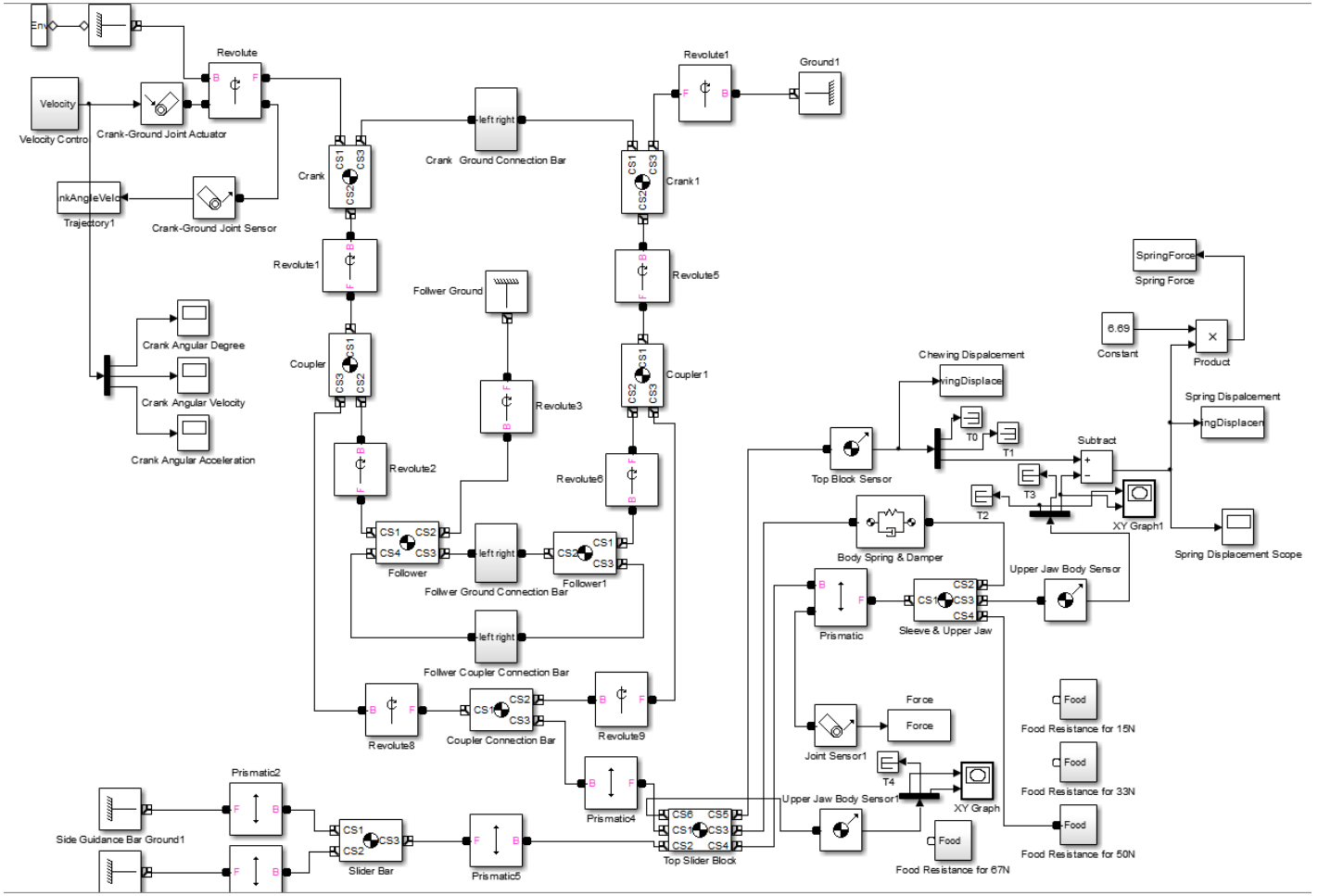


Figure B.1: SIMULINK model of SDOF mastication robot (reproduced from [52]).

Appendix C

Microsoft Excel - [Wed Feb 24 17-01-43 2016].xlsx

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Time (s)	Chewing Phase	Cycles To Do	Cycle	Fracture Force (N)	Ground Length (mm)	Sagittal Plane Angle (deg)	Lateral(X) Position (mm)	Sagittal(Y) Position (mm)	Vertical(Z) Position (mm)	Lateral(X) Velocity (mm/s)	Sagittal(Y) Velocity (mm/s)	Vertical(Z) Velocity (mm/s)	Lateral(X) Acceleration (mm/s2)	Sagittal(Y) Acceleration (mm/s2)	Vertical(Z) Acceleration (mm/s2)	Average Opening(Z) Velocity (mm/s)	Average Closing(Z) Velocity (mm/s)	X-Axis Chewing Force (N)	Y-Axis Chewing Force (N)	Z-Axis Chewing Force (N)	Average Chewing Force (N)	PSD Average Particle Size	Nu Pa
1028	11.773	Open	35	3	0	42.85994	1.5378	2.1332183	-0.65957388	-24.5686857	-6.56110653	-0.68368211	-25.466701	-97.2026964	-7.15727065	-266.603542	14.179943	30.0381964	0	0	0	0	1172	
1029	11.781	Open	35	3	0	42.85994	1.5378	2.08180168	-0.66484176	-24.7649107	-7.39355576	-0.74477535	-27.7423833	-119.836879	-8.46256368	-315.224833	14.179943	30.0381964	0	0	0	0	1172	
1030	11.789	Open	35	3	0	42.85994	1.5378	2.04570411	-0.66843537	-24.89877	-5.17881068	-0.50950495	-18.9787185	319.339373	33.1406086	1234.46549	14.179943	30.0381964	0	0	0	0	1172	
1031	11.797	Open	35	3	0	42.85994	1.5378	2.04570411	-0.66843537	-24.89877	-5.17881068	-0.50950495	-18.9787185	319.339373	33.1406086	1234.46549	14.179943	30.0381964	0	0	0	0	1172	
1032	11.805	Open	35	3	0	42.85994	1.5378	2.01952527	-0.67098902	-24.9938921	-0.7293208	-0.07053678	-2.6274478	273.378334	26.4735533	986.122144	14.179943	30.0381964	0	0	0	0	1172	
1033	11.813	Open	35	3	0	42.85994	1.5378	2.00930337	-0.67197435	-25.0305949	-1.46189382	-0.14044796	-5.23159261	-104.882351	-9.94230074	-370.344049	14.179943	30.0381964	0	0	0	0	1172	
1034	11.821	Open	35	3	0	42.85994	1.5378	1.98879206	-0.67393179	-25.1035082	-2.93652692	-0.27836609	-10.3689507	-211.546502	-19.5192622	-727.079453	14.179943	30.0381964	0	0	0	0	1172	
1035	11.829	Open	35	3	0	42.85994	1.5378	1.98879206	-0.67393179	-25.1035082	-2.93652692	-0.27836609	-10.3689507	-211.546502	-19.5192622	-727.079453	14.179943	30.0381964	0	0	0	0	1172	
1036	11.837	Open	35	3	0	42.85994	1.5378	1.94750804	-0.6777929	-25.2473318	-5.92192526	-0.54639108	-20.352702	-429.783546	-37.5293362	-1397.94266	14.179943	30.0381964	0	0	0	0	1172	
1037	11.845	Open	35	3	0	42.85994	1.5378	1.83835472	-0.68750871	-25.6092392	-7.54667183	-0.64751107	-24.1193539	-63.1025356	-2.10548734	-78.4279944	14.179943	30.0381964	0	0	0	0	1172	
1038	11.853	Open	35	3	0	42.85994	1.5378	1.79898969	-0.69084141	-25.7333801	-5.64076933	-0.47114996	-17.5500208	274.105944	24.7249453	920.987664	14.179943	30.0381964	0	0	0	0	1172	
1039	11.861	Open	35	3	0	42.85994	1.5378	1.79898969	-0.69084141	-25.7333801	-5.64076933	-0.47114996	-17.5500208	274.105944	24.7249453	920.987664	14.179943	30.0381964	0	0	0	0	1172	
1040	11.869	Open	35	3	0	42.85994	1.5378	1.77769429	-0.6926071	-25.7991511	-2.66614343	-0.21943381	-8.17376242	373.030634	31.108866	1158.78444	14.179943	30.0381964	0	0	0	0	1172	
1041	11.877	Open	35	3	0	42.85994	1.5378	1.76091516	-0.69398004	-25.850292	-1.52722897	-0.12423176	-4.62754997	-109.422479	-8.76777482	-326.593744	14.179943	30.0381964	0	0	0	0	1172	
1042	11.885	Open	35	3	0	42.85994	1.5378	1.76091516	-0.69398004	-25.850292	-1.52722897	-0.12423176	-4.62754997	-109.422479	-8.76777482	-326.593744	14.179943	30.0381964	0	0	0	0	1172	
1043	11.893	Open	35	3	0	42.85994	1.5378	1.7379694	-0.69583154	-25.9192592	-2.87284295	-0.22993391	-8.56488436	-168.728262	-13.0350287	-485.546094	14.179943	30.0381964	0	0	0	0	1172	
1044	11.901	Open	35	3	0	42.85994	1.5378	1.69340132	-0.69934225	-26.0500306	-6.38580809	-0.49495432	-18.436717	-504.706288	-36.8105847	-1371.16964	14.179943	30.0381964	0	0	0	0	1172	
1045	11.909	Open	35	3	0	42.85994	1.5378	1.58008353	-0.70776256	-26.3636818	-7.81401007	-0.55563982	-20.6972114	-62.7610136	-1.01441522	-37.7862882	14.179943	30.0381964	0	0	0	0	1172	
1046	11.917	Open	35	3	0	42.85994	1.5378	1.53935781	-0.71061163	-26.4698077	-5.83108136	-0.40121323	-14.9449243	284.64116	21.5092749	801.206094	14.179943	30.0381964	0	0	0	0	1172	
1047	11.925	Open	35	3	0	42.85994	1.5378	1.53935781	-0.71061163	-26.4698077	-5.83108136	-0.40121323	-14.9449243	284.64116	21.5092749	801.206094	14.179943	30.0381964	0	0	0	0	1172	
1048	11.933	Open	35	3	0	42.85994	1.5378	1.51735292	-0.71211199	-26.5256951	-3.1472794	-0.21262911	-7.92029203	384.323055	26.526232	988.08439	14.179943	30.0381964	0	0	0	0	1172	
1049	11.941	Close	35	3	0	42.85994	1.5378	1.49687336	-0.71348369	-26.5767899	-1.37992242	-0.09162369	-3.41292108	-24.8411746	-1.54125853	-57.4108486	14.179943	30.0381964	0	0	0	0	1172	
1050	11.949	Close	35	3	0	42.85994	1.5378	1.47476916	-0.71493747	-26.6309424	-3.16136027	-0.20593423	-7.67091238	-255.066739	-16.046449	-597.719488	14.179943	30.0381964	0	0	0	0	1172	
1051	11.957	Close	35	3	0	42.85994	1.5378	1.47476916	-0.71493747	-26.6309424	-3.16136027	-0.20593423	-7.67091238	-255.066739	-16.046449	-597.719488	14.179943	30.0381964	0	0	0	0	1172	
1052	11.965	Close	35	3	0	42.85994	1.5378	1.36829707	-0.72154913	-26.8772223	-8.90009025	-0.52545626	-19.5728939	-480.31309	-23.783254	-885.910294	14.179943	30.0381964	0	0	0	0	1172	
1053	11.973	Close	35	3	0	42.85994	1.5378	1.31063106	-0.72485739	-27.0004526	-8.26042704	-0.46005384	-17.1366975	91.9572445	9.09962645	338.954996	14.179943	30.0381964	0	0	0	0	1172	
1054	11.981	Close	35	3	0	42.85994	1.5378	1.26556326	-0.72730802	-27.0917368	-6.45171532	-0.34230037	-12.7504596	259.495457	16.2140166	603.961268	14.179943	30.0381964	0	0	0	0	1172	
1055	11.989	Close	35	3	0	42.85994	1.5378	1.26556326	-0.72730802	-27.0917368	-6.45171532	-0.34230037	-12.7504596	259.495457	16.2140166	603.961268	14.179943	30.0381964	0	0	0	0	1172	
1056	11.997	Close	35	3	0	42.85994	1.5378	1.22678388	-0.72932119	-27.1667262	-5.54977833	-0.28175406	-10.4951501	129.317972	8.38835652	312.460667	14.179943	30.0381964	0	0	0	0	1172	
1057	12.005	Close	35	3	0	42.85994	1.5378	1.1324956	-0.73384341	-27.3351757	-7.69684769	-0.34742143	-12.9412156	-376.081044	-13.4067999	-499.394323	14.179943	30.0381964	0	0	0	0	1172	
1058	12.013	Close	35	3	0	42.85994	1.5378	1.06206501	-0.73687245	-27.4480056	-8.83227183	-0.36095495	-13.4453303	-142.878218	-1.07896305	-40.1906517	14.179943	30.0381964	0	0	0	0	1172	
1059	12.021	Close	35	3	0	42.85994	1.5378	1.06206501	-0.73687245	-27.4480056	-8.83227183	-0.36095495	-13.4453303	-142.878218	-1.07896305	-40.1906517	14.179943	30.0381964	0	0	0	0	1172	
1060	12.029	Close	35	3	0	42.85994	1.5378	0.88979736	-0.74299829	-27.676189	-10.8548568	-0.32784787	-12.2121138	182.502101	12.9041951	480.672633	14.179943	30.0381964	0	0	0	0	1172	
1061	12.037	Close	35	3	0	42.85994	1.5378	0.81100785	-0.74518254	-27.757551	-11.2992	-0.28512453	-10.6206978	-64.1426592	6.47253297	241.097521	14.179943	30.0381964	0	0	0	0	1172	
1062	12.045	Close	35	3	0	42.85994	1.5378	0.75024134	-0.74659866	-27.8103003	-7.61902268	-0.16281946	-6.06491602	462.761882	13.5920422	506.294476	14.179943	30.0381964	0	0	0	0	1172	
1063	12.053	Close	35	3	0	42.85994	1.5378	0.75024134	-0.74659866	-27.8103003	-7.61902268	-0.16281946	-6.06491602	462.761882	13.5920422	506.294476	14.179943	30.0381964	0	0	0	0	1172	
1064	12.061	Close	35	3	0	42.85994	1.5378	0.68569115	-0.74784495	-27.8567239	-2.98257636	-0.05142189	-1.91543116	356.684841	6.7201106	250.319622	14.179943	30.0381964	0	0	0	0	1172	
1065	12.069	Close	35	3	0	42.85994	1.5378	0.65838324	-0.74829183	-27.8733697	-3.90667824	-0.06050448	-2.25375129	-132.396889	-1.06964991	-39.8437434	14.179943	30.0381964	0	0	0	0	1172	
1066	12.077	Close	35	3	0	42.85994	1.5378	0.61212355	-0.74893944	-27.897493	-6.62466594	-0.08287652	-3.08709507	-390.210898	-2.053579	-76.4944435	14.179943	30.0381964	0	0	0	0	1172	
1067	12.085	Close	35	3	0	42.85994	1.5378	0.61212355	-0.74893944	-27.897493	-6.62466594	-0.08287652	-3.08709507	-390.210898	-2.053579	-76.4944435	14.179943	30.0381964	0	0	0	0	1172	
1068	12.093	Close	35	3	0	42.85994	1.5378	0.45799637	-0.75010068	-27.9407484	-11.1331082	-0.02840525	-1.05807656	-220.238646	7.45868337	277.830964	14.179943	30.0381964	0	0	0	0	1172	
1069	12.101	Close	35	3	0	42.85994	1.5378	0.34906775	-0.74999474	-27.936802	-15.655787	0.07037898	2.62157007	-654.097351	18.7844679	699.708859	14.179943	30.0381964	0	0	0	0	1172	
1070	12.109	Close	35	3	0	42.85994	1.5378	0.21909594	-0.74886516	-27.8947259	-18.7025368	0.2408929	8.97309916	-441.468546	28.195773	1050.27367	14.179943	30.0381964	0	0	0	0	1172	
1071	12.117	Close	35	3	0	42.85994	1.5378	0.21909594	-0.74886516	-27.8947259	-18.7025368	0.2408929	8.97309916	-441.468546	28.195773	1050.27367	14.179943	30.0381964	0	0	0	0	1172	
1072	12.125	Close	35	3	0	42.85994	1.5378	-0.11134486	-0.74111483	-27.6060315	-22.0547763	0.74921157	27.9076296	48.0456233	29.1726173	1086.66047	14.179943	30.0381964	0	0	0	0	1172	

Data

Figure C.1: Example log file generated by the KBS.

References

[1] R. D. Brown. (2010). *Food Texture Analysis* [Online]. Available: http://www.americanlaboratory.com/914-Application-Notes/485-Food-Texture-Analysis/

[2] F. Hayakawa, Y. Kazami, K. Nishinari, K. Ioku, S. Akuzawa, Y. Yamano*, et al.*, "Classification of Japanese Texture Terms," *Journal of Texture Studies,* vol. 44, pp. 140-159, 2013.

[3] A. M. Paula and A. C. Conti-Silva, "Texture profile and correlation between sensory and instrumental analyses on extruded snacks," *Journal of Food Engineering,* vol. 121, pp. 9-14, 2014.

[4] W. Xu and J. E. Bronlund, "Understanding Food Texture Using Masticatory Robots," in *Mastication Robots: Biological Inspiration to Implementation*, ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 207-236.

[5] G. Ares and A. Giménez, "New Trends in Sensory Characterization of Food Products," in *Advances in Food Science and Nutrition*, ed: John Wiley & Sons, Inc., 2013, pp. 321-359.

[6] H. T. Lawless and H. Heymann, "Introduction," in *Sensory Evaluation of Food: Principles and Practices*, ed New York, NY: Springer New York, 2010, pp. 1-18.

[7] H. S. G. Tan, A. R. H. Fischer, H. C. M. van Trijp, and M. Stieger, "Tasty but nasty? Exploring the role of sensory-liking and food appropriateness in the willingness to eat unusual novel foods like insects," *Food Quality and Preference,* vol. 48, Part A, pp. 293-302, 2016.

[8] B. Rousseau, "Sensory discrimination testing and consumer relevance," *Food Quality and Preference,* vol. 43, pp. 122-125, 2015.

[9] T. Næs, P. B. Brockhoff, and O. Tomic, "Quality Control of Sensory Profile Data," in *Statistics for Sensory and Consumer Science*, ed: John Wiley & Sons, Ltd, 2010, pp. 11-38.

[10] E. Vigneau, E. M. Qannari, B. Navez, and V. Cottet, "Segmentation of consumers in preference studies while setting aside atypical or irrelevant consumers," *Food Quality and Preference,* vol. 47, Part A, pp. 54 - 63, 2016.

[11] P. Talsma, "Assessing sensory panel performance using generalizability theory," *Food Quality and Preference,* vol. 47, Part A, pp. 3 - 9, 2016.

[12] TextureTechnologies. *TA.HD Plus Texture Analyzer* [Online]. Available: http://www.texturetechnologies.com/texture-analyzers/TA-HDPlus-texture-analyzer.php

[13] W. Xu and J. E. Bronlund, "Introduction," in *Mastication Robots: Biological Inspiration to Implementation*, ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1-33.

[14] A. Okino, T. Inoue, H. Takanobu, A. Takanishi, K. Ohtsuki, M. Ohnishi*, et al.*, "A clinical jaw movement training robot for lateral movement training," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, 2003, pp. 244-249 vol.1.

[15] C. Salles, A. Tarrega, P. Mielle, J. Maratray, P. Gorria, J. Liaboeuf*, et al.*, "Development of a chewing simulator for food breakdown and the analysis of in vitro flavor compound release in a mouth environment," *Journal of Food Engineering,* vol. 82, pp. 189-198, 9// 2007.

[16] W. L. Xu, D. Lewis, J. E. Bronlund, and M. P. Morgenstern, "Mechanism, design and motion control of a linkage chewing device for food evaluation," *Mechanism and Machine Theory,* vol. 43, pp. 376-389, 2008.

[17] C. Sun, J. E. Bronlund, L. Huang, and W. L. Xu, "Adaptive fuzzy control on a chewing machine," in *Cybernetics and Intelligent Systems (CIS), 2011 IEEE 5th International Conference on*, 2011, pp. 202-207.

[18] C. Sun, W. L. Xu, J. E. Bronlund, and M. Morgenstern, "Dynamics and Compliance Control of a Linkage Robot for Food Chewing," *Industrial Electronics, IEEE Transactions on,* vol. 61, pp. 377-386, 2014.

[19] G. Lanzola and M. Stefanelli, "A Specialized Framework for Medical Diagnostic Knowledge-Based Systems," *Computers and Biomedical Research,* vol. 25, pp. 351-365, 1992.

[20] Z. Jia, Y. Shi, Y. Jia, and D. Li, "A Framework of Knowledge Management Systems for Tourism Crisis Management," *Procedia Engineering,* vol. 29, pp. 138-143, 2012.

[21] M. Bourne, "Relation Between Texture and Mastication," *Journal of Texture Studies,* vol. 35, pp. 125-143, 2004.

[22] L. Canetti, E. Bachar, and E. M. Berry, "Food and emotion," *Behavioural Processes,* vol. 60, pp. 157-164, 2002.

[23] A. Kramer, "Food Texture — Definition, Measurement and Relation to Other Food Quality Attributes," in *Texture Measurements of Foods: Psychophysical Fundamentals: Sensory, Mechanical, and Chemical Procedures, and their Interrelationships*, A. Kramer and A. S. Szczesniak, Eds., ed Dordrecht: Springer Netherlands, 1973, pp. 1-9.

[24] A. S. Szczesniak, "Texture is a sensory property," *Food Quality and Preference,* vol. 13, pp. 215-225, 2002.

[25] *Sensory analysis — Vocabulary*. ISO Standard 5492. 2008

[26] D. Kilcast, "1 - Measurement of the sensory quality of food: an introduction," in *Instrumental Assessment of Food Sensory Quality*, ed: Woodhead Publishing, 2013, pp. 1-26.

[27] AMETEK. *What is Food Texture and How is it Measured* [Online]. Available: http://www.ametektest.com/learningzone/library/articles/what-is-food-texture-and-how-is-it-measured

[28] M. A. Brandt, E. Z. Skinner, and J. A. Coleman, "Texture Profile Method," *Journal of Food Science,* vol. 28, pp. 404-409, 1963.

[29] TextureTechnologies. *An Overview of Texture Profile Analysis (TPA)* [Online]. Available: http://texturetechnologies.com/texture-profile-analysis/texture-profile-analysis.php#section-02

[30] A. S. Szczesniak, M. A. Brandt, and H. H. Friedman, "Development of Standard Rating Scales for Mechanical Parameters of Texture and Correlation Between the Objective and the Sensory Methods of Texture Evaluation," *Journal of Food Science,* vol. 28, pp. 397-403, 1963.

[31] M. A. Chauvin, C. Parks, C. Ross, and B. G. Swanson, "Multidimensional representation of the standard scales of food texture," *Journal of Sensory Studies,* vol. 24, pp. 93-110, 2009.

[32] T. H. Lawless and H. Heymann, "Principles of Good Practice," in *Sensory Evaluation of Food: Principles and Practices*, ed New York, NY: Springer New York, 2010, pp. 57-77.

[33] F. Zhu, R. Sakulnak, and S. Wang, "Effect of black tea on antioxidant, textural, and sensory properties of Chinese steamed bread," *Food Chemistry,* vol. 194, pp. 1217-1223, 2016.

[34] S. Albertini, A. E. Lai Reyes, J. M. Trigo, G. A. Sarriés, and M. H. F. Spoto, "Effects of chemical treatments on fresh-cut papaya," *Food Chemistry,* vol. 190, pp. 1182-1189, 2016.

[35] J. S. Mouta, N. C. de Sá, E. Menezes, and L. Melo, "Effect of institutional sensory test location and consumer attitudes on acceptance of foods and beverages having different levels of processing," *Food Quality and Preference,* vol. 48, Part A, pp. 262-267, 2016.

[36] T. H. Lawless and H. Heymann, "Measurement of Sensory Thresholds," in *Sensory Evaluation of Food: Principles and Practices*, ed New York, NY: Springer New York, 2010, pp. 125-147.

[37] B. Vad Andersen and G. Hyldig, "Food satisfaction: Integrating feelings before, during and after food intake," *Food Quality and Preference,* vol. 43, pp. 126-134, 7// 2015.

[38] V. A. Solah, X. Meng, S. Wood, R. J. Gahler, D. A. Kerr, A. P. James*, et al.*, "Effect of Training on the Reliability of Satiety Evaluation and Use of Trained Panellists to Determine the Satiety Effect of Dietary Fibre: A Randomised Controlled Trial," *PLoS ONE,* vol. 10, p. e0126202, 2015.

[39] P. S. Katz, "Evolution of central pattern generators and rhythmic behaviours," *Philosophical Transactions of the Royal Society B: Biological Sciences,* vol. 371, 2016.

[40] E. Marder and D. Bucher, "Central pattern generators and the control of rhythmic movements," *Current Biology,* vol. 11, pp. R986-R996, 2001.

[41] R. S. Razavian, N. Mehrabi, and J. McPhee, "A Neuronal Model of Central Pattern Generator to Account for Natural Motion Variation," *Journal of Computational and Nonlinear Dynamics,* vol. 11, 2016.

[42] W. Stein, "Sensory Input to Central Pattern Generators," in *Encyclopedia of Computational Neuroscience*, D. Jaeger and R. Jung, Eds., ed New York, NY: Springer New York, 2013, pp. 1-11.

[43] W. L. Xu, F. Clara Fang, J. Bronlund, and J. Potgieter, "Generation of rhythmic and voluntary patterns of mastication using Matsuoka oscillator for a humanoid chewing robot," *Mechatronics,* vol. 19, pp. 205-217, 2009.

[44] J. Yu, Z. Wu, M. Wang, and M. Tan, "CPG Network Optimization for a Biomimetic Robotic Fish via PSO," *IEEE Transactions on Neural Networks and Learning Systems,* 2015.

[45] W. Xu and J. E. Bronlund, "Neural Control of a Mastication Robot," in *Mastication Robots: Biological Inspiration to Implementation*, ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 237-270.

[46] P. Jackson, *Introduction to Expert Systems*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[47] M. Barcelo-Valenzuela, P. S. Carrillo-Villafaña, A. Perez-Soltero, and G. Sanchez-Schmitz, "A framework to acquire explicit knowledge stored on different versions of software," *Information and Software Technology,* vol. 70, pp. 40-48, 2016.

[48] G. H. Lim, I. H. Suh, and H. Suh, "Ontology-Based Unified Robot Knowledge for Service Robots in Indoor Environments," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* vol. 41, pp. 492-509, 2011.

[49] D. Xie, W. L. Xu, K. D. Foster, and J. Bronlund, "Object-oriented knowledge framework for modelling human mastication of foods," *Expert Systems with Applications,* vol. 36, pp. 4810 - 4821, 2009.

[50] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel*, et al.*, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108-122.

[51] H. Koç, E. Çakir, C. J. Vinyard, G. Essick, C. R. Daubert, M. A. Drake*, et al.*, "Adaptation of Oral Processing to the Fracture Properties of Soft Solids," *Journal of Texture Studies,* vol. 45, pp. 47-61, 2014.

[52] C. Sun, "Modelling and Compliance Control of a Linkage Chewing Robot and Its Application in Food Evaluation," PhD Thesis, School of Engineering, Massey University (Albany, NZ), 2012.

Copyright Permissions

Reference [13]:

SPRINGER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Springer ("Springer") consists of your license details and the terms and conditions provided by Springer and Copyright Clearance Center.

License Number	3886971442237
License date	Jun 13, 2016
Licensed Content Publisher	Springer
Licensed Content Publication	Springer eBook
Licensed Content Title	Introduction
Licensed Content Author	Weiliang Xu
Licensed Content Date	Jan 1, 2010
Type of Use	Thesis/Dissertation
Portion	Figures/tables/illustrations
Number of figures/tables/illustrations	1
Author of this Springer article	No
Order reference number	
Original figure numbers	Fig. 1.14b
Title of your thesis / dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [15]:

ELSEVIER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	3886980386474
License date	Jun 13, 2016
Licensed Content Publisher	Elsevier
Licensed Content Publication	Journal of Food Engineering
Licensed Content Title	Development of a chewing simulator for food breakdown and the analysis of in vitro flavor compound release in a mouth environment
Licensed Content Author	C. Salles,A. Tarrega,P. Mielle,J. Maratray,P. Gorria,J. Liaboeuf,J.-J. Liodenot
Licensed Content Date	September 2007
Licensed Content Volume Number	82
Licensed Content Issue Number	2
Licensed Content Pages	10
Start Page	189
End Page	198
Type of Use	reuse in a thesis/dissertation
Intended publisher of new work	other
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Order reference number	
Original figure numbers	Fig. 3
Title of your thesis/dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [16]:

ELSEVIER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	3886980818310
License date	Jun 13, 2016
Licensed Content Publisher	Elsevier
Licensed Content Publication	Mechanism and Machine Theory
Licensed Content Title	Mechanism, design and motion control of a linkage chewing device for food evaluation
Licensed Content Author	W.L. Xu,D. Lewis,J.E. Bronlund,M.P. Morgenstern
Licensed Content Date	March 2008
Licensed Content Volume Number	43
Licensed Content Issue Number	3
Licensed Content Pages	14
Start Page	376
End Page	389
Type of Use	reuse in a thesis/dissertation
Intended publisher of new work	other
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Order reference number	
Original figure numbers	Fig. 7
Title of your thesis/dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [23]:

SPRINGER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Springer ("Springer") consists of your license details and the terms and conditions provided by Springer and Copyright Clearance Center.

License Number	3886981281110
License date	Jun 13, 2016
Licensed Content Publisher	Springer
Licensed Content Publication	Springer eBook
Licensed Content Title	Food Texture — Definition, Measurement and Relation to Other Food Quality Attributes
Licensed Content Author	Amihud Kramer
Licensed Content Date	Jan 1, 1973
Type of Use	Thesis/Dissertation
Portion	Figures/tables/illustrations
Number of figures/tables/illustrations	1
Author of this Springer article	No
Order reference number	
Original figure numbers	Fig. 2
Title of your thesis / dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [28]:

JOHN WILEY AND SONS LICENSE TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and John Wiley and Sons ("John Wiley and Sons") consists of your license details and the terms and conditions provided by John Wiley and Sons and Copyright Clearance Center.

License Number	3886990328304
License date	Jun 13, 2016
Licensed Content Publisher	John Wiley and Sons
Licensed Content Publication	Journal of Food Science
Licensed Content Title	Texture Profile Method
Licensed Content Author	MARGARET A. BRANDT,ELAINE Z. SKINNER,JOHN A. COLEMAN
Licensed Content Date	Aug 25, 2006
Licensed Content Pages	6
Type of use	Dissertation/Thesis
Requestor type	University/Academic
Format	Print and electronic
Portion	Figure/table
Number of figures/tables	1
Original Wiley figure/table number(s)	Table 2
Will you be translating?	No
Title of your thesis / dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [30]:

JOHN WILEY AND SONS LICENSE TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and John Wiley and Sons ("John Wiley and Sons") consists of your license details and the terms and conditions provided by John Wiley and Sons and Copyright Clearance Center.

License Number	3886990774814
License date	Jun 13, 2016
Licensed Content Publisher	John Wiley and Sons
Licensed Content Publication	Journal of Food Science
Licensed Content Title	Development of Standard Rating Scales for Mechanical Parameters of Texture and Correlation Between the Objective and the Sensory Methods of Texture Evaluation
Licensed Content Author	ALINA SURMACKA SZCZESNIAK,MARGARET A. BRANDT,HERMAN H. FRIEDMAN
Licensed Content Date	Aug 25, 2006
Licensed Content Pages	7
Type of use	Dissertation/Thesis
Requestor type	University/Academic
Format	Print and electronic
Portion	Figure/table
Number of figures/tables	1
Original Wiley figure/table number(s)	Table 1
Will you be translating?	No
Title of your thesis / dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [33]:

ELSEVIER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	3886991397173
License date	Jun 13, 2016
Licensed Content Publisher	Elsevier
Licensed Content Publication	Food Chemistry
Licensed Content Title	Effect of black tea on antioxidant, textural, and sensory properties of Chinese steamed bread
Licensed Content Author	Fan Zhu,Ratchaneekorn Sakulnak,Sunan Wang
Licensed Content Date	1 March 2016
Licensed Content Volume Number	194
Licensed Content Issue Number	n/a
Licensed Content Pages	7
Start Page	1217
End Page	1223
Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Order reference number	
Original figure numbers	Table 1
Title of your thesis/dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [35]:

ELSEVIER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	3887000088000
License date	Jun 13, 2016
Licensed Content Publisher	Elsevier
Licensed Content Publication	Food Quality and Preference
Licensed Content Title	Effect of institutional sensory test location and consumer attitudes on acceptance of foods and beverages having different levels of processing
Licensed Content Author	Juliana S. Mouta,Nathaly C. de Sá,Ellen Menezes,Lauro Melo
Licensed Content Date	March 2016
Licensed Content Volume Number	48
Licensed Content Issue Number	n/a
Licensed Content Pages	6
Start Page	262
End Page	267
Type of Use	reuse in a thesis/dissertation
Intended publisher of new work	other
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Order reference number	
Original figure numbers	Fig. 2
Title of your thesis/dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [42]:

SPRINGER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Springer ("Springer") consists of your license details and the terms and conditions provided by Springer and Copyright Clearance Center.

License Number	3887000365461
License date	Jun 13, 2016
Licensed Content Publisher	Springer
Licensed Content Publication	Springer eBook
Licensed Content Title	Sensory Input to Central Pattern Generators
Licensed Content Author	Wolfgang Stein
Licensed Content Date	Jan 1, 2015
Type of Use	Thesis/Dissertation
Portion	Figures/tables/illustrations
Number of figures/tables/illustrations	1
Author of this Springer article	No
Order reference number	
Original figure numbers	Fig. 1
Title of your thesis / dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [47]:

ELSEVIER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	3887000827472
License date	Jun 13, 2016
Licensed Content Publisher	Elsevier
Licensed Content Publication	Information and Software Technology
Licensed Content Title	A framework to acquire explicit knowledge stored on different versions of software
Licensed Content Author	Mario Barcelo-Valenzuela,Patricia Shihemy Carrillo-Villafaña,Alonso Perez-Soltero,Gerardo Sanchez-Schmitz
Licensed Content Date	February 2016
Licensed Content Volume Number	70
Licensed Content Issue Number	n/a
Licensed Content Pages	9
Start Page	40
End Page	48
Type of Use	reuse in a thesis/dissertation
Intended publisher of new work	other
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Order reference number	
Original figure numbers	Fig. 2
Title of your thesis/dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots

Reference [49]:

ELSEVIER LICENSE
TERMS AND CONDITIONS

Jun 13, 2016

This Agreement between Ramin Odisho ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	3887001286157
License date	Jun 13, 2016
Licensed Content Publisher	Elsevier
Licensed Content Publication	Expert Systems with Applications
Licensed Content Title	Object-oriented knowledge framework for modelling human mastication of foods
Licensed Content Author	D. Xie,W.L. Xu,K.D. Foster,J. Bronlund
Licensed Content Date	April 2009
Licensed Content Volume Number	36
Licensed Content Issue Number	3
Licensed Content Pages	12
Start Page	4810
End Page	4821
Type of Use	reuse in a thesis/dissertation
Intended publisher of new work	other
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	2
Format	both print and electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Order reference number	
Original figure numbers	Fig. 3, Fig. 5
Title of your thesis/dissertation	Knowledge-Based System for Autonomous Control of Intelligent Mastication Robots