



Pedro Miguel Ortiz da Silva

Master of Science

Automatic Dispenser for Kitchen Robots

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Electrical and Computer Engineering

Adviser: Prof, José António Barata de Oliveira,
NOVA University of Lisbon

Co-adviser: Francisco Marques, Research Engineer,
UNINOVA-CTS

Examination Committee



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2020

Automatic Dispenser for Kitchen Robots

Copyright © Pedro Miguel Ortiz da Silva, Faculty of Sciences and Technology, NOVA University Lisbon.

The Faculty of Sciences and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

ACKNOWLEDGEMENTS

Quero agradecer ao meu orientador Professor José António Barata e ao meu co-orientador Francisco Marques por todo o apoio e aconselhamento no desenvolvimentos desta dissertação.

Agradeço também à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa e ao Departamento de Engenharia Eletrotécnica e de Computadores por me ter proporcionado o ensino e as ferramentas que me permitiram realizar este percurso.

Agradeço aos meus pais e ao meu irmão por acreditarem sempre em mim, e por estarem sempre ao meu lado e pela força que sempre me deram. Obrigado também a toda a minha família por estarem sempre presentes e por terem feito parte deste percurso.

Agradeço à minha namorada pelo apoio incondicional e motivação que me dá, por me ajudar nos maus momentos e por fazer os bons momentos ainda melhores.

Obrigado ao meu padrinho e ao meu irmão de faculdade, Gonçalo Horta e João Miguel, por terem sido tão importantes nestes anos, por me terem ajudado e por estarem sempre presentes, não só na faculdade mas também fora dela.

Obrigado a todos os meus amigos, tanto aos que fiz na faculdade como os mais antigos, por terem feito parte deste percurso e com quem partilhei os melhores e piores momentos, com os quais sempre pude contar e que nunca me deixaram ficar mal.

ABSTRACT

In the last years we have seen technology and human-machine-interaction exponentially evolve and having great developments. With these developments and the integration of technology in every day life, a natural change in quotidian life is expected, and a place where we can see these changes is in the kitchen.

One of technology's objectives is to ease a task or do it completely on its own, with the rising pace at which the society lives it became a necessity to reduce the wasted time in every way we can. This dissertation objective was to reduce the wasted time, by being integrated in the kitchen it will reduce the time the user needs to be present and therefore use the free time as he wishes. There are already some implemented solutions, however, those solutions still have some problems that end up limiting the possibility of user absence, the ones that permit total absence don't permit any user input as to change any recipe information during its execution.

As a solution for this, an automatic dispenser was developed as the objective of this dissertation, the goal of this dispenser is to deliver the required ingredients for a given recipe, this recipe will be given by the main machine where this dispenser is to connect and be a module of. The development of this work started with looking into some existing solutions and identify their major issues, and with those in mind define software and hardware architectures, to better answer the problems at hand and get to an improved solution which the user can rely on.

Keywords: Kitchen Robot, Automatic Dispenser, Controllers.

RESUMO

Nos últimos anos a tecnologia e as interações humano-máquina têm sofrido uma evolução exponencial e com grandes desenvolvimentos. Com estes desenvolvimentos e integração dessas tecnologias no dia a dia vem uma mudança natural na vida cotidiana, uma zona onde podemos observar estas mudanças é na cozinha.

Um dos objetivos da tecnologia é o de facilitar tarefas ou fazê-las por completo, com o ritmo cada vez mais acelerado com que a sociedade vive, tornou-se numa necessidade reduzir o tempo desperdiçado nas mais diversas áreas. Esta dissertação surge com o objetivo de reduzir esse tempo desperdiçado a cozinhar, sendo esta uma tarefa que necessita de algum tempo, tempo esse que poderia ser utilizado para lazer. Apesar de existirem já algumas soluções implementadas, existem ainda alguns problemas que acabam por limitar a possibilidade de uma ausência total do utilizador, as que permitem esta ausência, não permitem qualquer alteração por parte do utilizador na receita, após iniciar o processo. De forma a solucionar estas questões, foi desenvolvido um dispensador automático nesta dissertação, o objetivo deste dispensador é o de dispensar ingredientes para uma dada receita, esta receita é dada pela máquina principal à qual este dispensador deve ser conectado, e da qual deve ser um modulo. O desenvolvimento desta dissertação começou por analisar as soluções já existentes e identificar os seus maiores problemas, e a partir destes, definir arquiteturas de *software* e *hardware* que respondem da melhor forma aos mesmos, de modo a obter uma melhor solução final em que o utilizador possa confiar.

Palavras-chave: Robô de cozinha, Dispensador Automático, Controlo.

CONTENTS

List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Context and Motivation	1
1.2 Main project	2
1.3 Dissertation Objectives	3
2 Related Work	5
2.1 Current Existing Solutions	5
2.1.1 Recipe Following Solutions	5
2.1.2 Kitchen Robots	6
2.1.3 Autonomous Kitchens	8
2.2 Under Development Work	10
2.3 Unsolved Problems	13
2.4 Used Technologies	13
2.4.1 Fuzzy Control [23]	13
2.4.2 ROS [14]	20
2.4.3 Node-Red [13]	22
2.4.4 Behaviour Trees [7]	24
3 Automatic Dispenser	27
3.1 Software Architecture	27
3.2 Hardware	29
3.3 Software	33
4 Experimental results	41
4.1 Complete System	44
4.2 Velocity controller	50
5 Conclusions	53
5.1 Future Work	54

CONTENTS

Bibliography

57

LIST OF FIGURES

2.1	CookingNavi[1]	6
2.2	MimiCook	6
2.3	Kitchen Robots Examples	7
2.4	OneCook	7
2.5	Sereneti Kitchen[8]	8
2.6	Flippy[4]	8
2.7	Spyce[9]	9
2.8	Prometheus [18]	9
2.9	Moley [24]	10
2.10	Autonomous Robot in Aware Kitchen	11
2.11	Control algorithm for estimation of nitrogen and addition of urea	12
3.1	Software Architecture with all the modules and their direct connections	27
3.2	Hardware components connections diagram	29
3.3	Solid ingredients dispenser on the platform	30
3.4	Liquid ingredients dispenser on the platform	30
3.5	Arduino and DRV8825 stepper motor driver connections	31
3.6	DC motor dispenser 3D model	31
3.7	Stepper motor dispenser prototype	32
3.8	Software Architecture with all the modules, connections and services	33
3.9	Recipe sequence chart	36
3.10	Reset revolver sequence chart	37
3.11	Empty sequence chart	38
3.12	Refill sequence chart	39
3.13	Dispense sequence chart	40
4.1	Rubber tube in the rotative platform from an outside container	42
4.2	Pump connected to outside container	42
4.3	Side view of the solids dispenser, showing the exit of the ingredients and the endless screw	43
4.4	Rotative platform with dc motor dispensers	44
4.5	Components connections	45
4.6	Pump and stepper motor	45

4.7 Solid ingredients dispenser with stepper motor	45
4.8 Salt drop graph	47
4.9 Pepper drop graph	47
4.10 Velocity output with smallest of maximum defuzzifier	51
4.11 Velocity output with mean of maximum defuzzifier	51
4.12 Velocity output with center average defuzzifier	52

LIST OF TABLES

4.1	Test results for salt using dc motors	46
4.2	Test results for pepper using dc motors	46
4.3	Olive Oil test results using dc motors	46
4.4	First test results for Salt with stepper motors	48
4.5	Salt test results for stepper motors after adjustments	49
4.6	Pepper test results for stepper motors after adjustments	49
4.7	Olive Oil test results	49
4.8	Olive Oil test comparison	49
4.9	Salt test comparison	50
4.10	Pepper test comparison	50

INTRODUCTION

1.1 Context and Motivation

In the recent years, technology and Human-Machine-Interaction have seen an exponential evolution with great developments. With this in mind, it is now possible to see that every aspect in quotidian life is changing, like cooking. Cooking is also a field where technology can be implemented to make the process easier, making it more automated and helping through some steps of it, possibly doing it entirely on its own. One of the main purposes of technology is to help or simplify a user's task, automating it completely or to a certain point. Nowadays with life's increased pace, everyone wants to decrease as much waste as possible, and so, every option that accelerates a process or releases the user of it is taken into consideration as to free as much time as possible.

It is then important to look at everything that consumes this time we want to free, with cooking being one of these time consuming tasks. We must look into every part of the process to understand where to implement technological changes and provide as much support as possible to the user. We can split the cooking process into different steps and implement different solutions for each one, some of these steps might be, getting the recipe, food preparation beforehand, food processing and so on. Currently, every time we want to cook something we have to be present, to prepare the ingredients, to put them in the right order or to change plates between steps.

Each of this tasks require most of our attention, we need to check the time of every action, put the right quantity of ingredients and try not to miss any detail to get the desired final result. For that we may use some help from the technological solutions available for each step, for example, to find and follow a recipe there are already some implemented technologies that may help in this process, by audiovisual means they make it easier to know what and when to do every action with precision during the whole process. It is

also possible to get most of the process automated, either by getting a kitchen robot or a full autonomous kitchen. The first option contains and provides a variety of tools and programs, to do some specific processes simplifying the ingredients preparation and food confection. In the last one the kitchen is closed during the process to prevent any harm to the human and there is no need for human inputs, making it possible not only to be totally absent but it also releases the user of the need to know anything about the recipe.

1.2 Main project

As explained in the previous section, technology is already implemented into food preparation, as such, kitchen robots are increasingly used appliances, either at home or at work, to ease the meal preparation process. With this in mind, the project in which this dissertation is inserted, aims to research and develop new advanced technological solutions. The solutions are to be applied in kitchen robots in order to develop an equipment which is the main hub of the kitchen, and to reach that goal features linked to the following solutions will be implemented:

1. Interface and connectivity;
2. Nutritional intelligence;
3. Food replenishment and traceability;
4. Sensing and automation;
5. Analytical intelligence for equipment management.

This project aims to support the research and technological development associated to the innovations foreseen in the short term, and contribute decisively to the future of kitchen robots, in order to anticipate technological trends and the demands of new consumers / users, and start to explore the potential that can be implemented in the medium and long term. Future kitchen robots generations will be sustained in a new and completely different approach, in a way that they can be used and understood by the consumer, may it be a professional or an amateur.

There are already some predicted functionalities for the future of this robots. The development of an app to make it possible to operate the machine from afar, provide information about the cost, the time taken to prepare the meal, the nutritional information and more. Making it a connected machine is also a possible solution, this will ease the after sale customer experience, with over the air updates made autonomously that reduce maintenance/repair cost and time.

It's intention of this project that the solution developed contribute to simplify the meal preparation process, making the robot able to operate more independently and reducing the time that the user needs to spend during the process. To achieve that, the machine will have an automatic dispenser module that can be added to the main frame of the

machine.

This module pretends to release the user of the tedious task of monitoring and controlling every phase of the process while providing at the same time, the option to change any desired aspect of the meal, this will let the user decide whether he wants to abstract himself of all the process or to give some personal modifications.

1.3 Dissertation Objectives

As mentioned before, technology is impacting every aspect in life, one of the fields where this is happening is in the kitchen, we already took a look into the project in which this dissertation is inserted and its proposed solutions and problems, with this in mind the objectives of this dissertation were defined. This dissertation consists in designing and building an automatic dispenser for a kitchen robot.

This automatic dispenser will have more quantity and variety of ingredients, with particular attention to the seasoning, it will be flexible, in a way that lets the user manage and modify the recipe during the confection, making use of the available sensors and user preferences. It will be modular, which means it can be added to the machine later and only if the user finds it useful, and will be controlled making use of scales , motors and controllers that to measure and process data in order to develop the better answer to what is requested.

RELATED WORK

In this section we will look into the current existing solutions, as to understand what is the current status of the problem at hand, as well as under development projects, this way it will be possible to understand how to better develop our final solution. The technologies that will be used will also be explained in this section, since they are a part of what is already implemented, even if not in these particular applications.

2.1 Current Existing Solutions

We will now look into different kinds of solutions that are already implemented, these solutions will be split into three different groups based on their autonomy level, we look into their differences and problems, and understand where does the solution proposed in this dissertation fit.

2.1.1 Recipe Following Solutions

Since the main purpose of this project is to ease the cooking process and all its tasks, it's wise to consider every type of help that's possible to provide, for example, providing help during the recipe following. During this process the user has to be capable of giving attention to every detail in the recipe as well as execute every step with precision, there are already some implemented solutions with this in mind:

1. **Cooking Navi**[1]: Cooking Navi it's an application developed as a cooking navigation system, the main objective of this application is to guide the user during the cooking process. To guide him through all the steps of the recipe, it makes use of not only text recipes, but also audio and video guides with the purpose of releasing the user of the struggle from trying to read an heavy text while focusing on cooking.

It also offers the possibility of cooking various recipes in parallel based on a flow chart system.

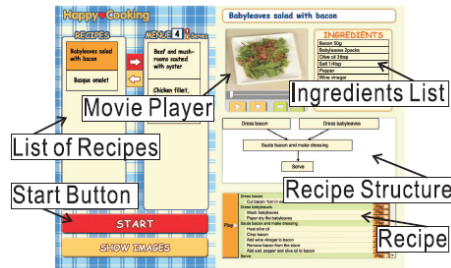


Figure 2.1: CookingNavi[1]

2. **MimiCook**[20]: This system, although with a different method, has the same goal as the previous one, to help the user through the cooking process, guiding him through all the steps of the recipe. It provides an Augmented Reality environment making use of a computer, a depth camera, a projector and a scaling device. By using this setup recipes are embodied in the cooking area, using the depth camera to check if the user is making everything correctly, like using the right objects. With the projector it is possible to draw guidelines and signal the next object or step, it is also possible to show a scale projected onto the table using the information from the scaling device that shows the user when the desired weight is reached, making it easier to follow the recipe without errors.



Figure 2.2: MimiCook

2.1.2 Kitchen Robots

The main reason for the use of kitchen robots is to automate the process of cooking and release the user from the stress of preparing the meal or knowing how to make it. While the objective of this project is to improve the kitchen robots even further, making it possible for the user to be absent while the robot cooks, there are already some solutions proposed that still need the interference of the user, some of the most relevant are:

1. **Yämmi** [25], **Bimby**[3], **KenWood Cooking Chef** [5], **LadyMaxx** [17], **Chef Express**[6]: The Five of them are kitchen robots designed with some of the main functions needed to assist a cooker. They make use of a collection of tools, options and accessories provided by the machines, there is also the possibility of buying additional accessories to get even more options. Some of them have a recipe database adapted to the specific use of that robot, in which, each recipe consists in guiding the user through the use of the robot, changing between buttons and tools used in each step and the ingredients needed to add. The user has to be part of the process and needs to be present for most of the time.



Figure 2.3: Kitchen Robots Examples

2. **OneCook**[15]: OneCook it's also a kitchen robot, but instead of having a recipe database that guides the user through the use of the machine and the ingredients to use, it cooks the meal by itself. This robot only needs ingredients prepared previously and stored in specified containers, it also allows you to buy food packages with all the needed ingredients ready to put into the machine. This makes this preparation the only moment in which the user needs to be present.



Figure 2.4: OneCook

3. **Sereneti Kitchen**[21]: Like the previous one, this robot cooks the meal by itself and doesn't need any external inputs during the cook process. It only needs previously prepared ingredients that are stored by order of use in containers inside the machine, after storing these ingredients the user can use an app to schedule the preparation of the meal without being there to start it manually, however this robot offers a limited set of recipes as the only tool it has is a rotative robotic arm.

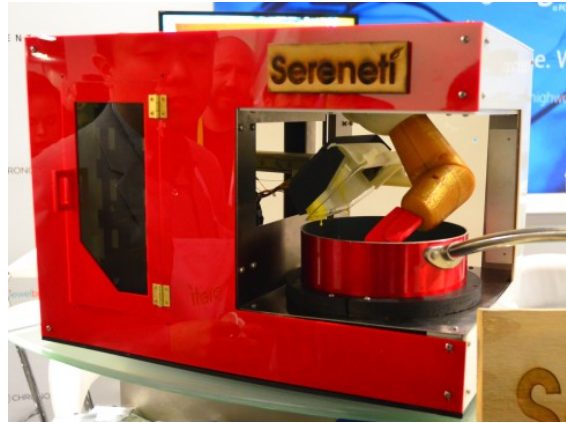


Figure 2.5: Sereneti Kitchen[8]

4. **Flippy**[10]: Flippy is a different way of looking at the kitchen robots, instead of being designed to help the user cook a lot of dishes, it is designed to do a specific function, in this case being flip burgers. It needs constant monitoring of a human, as an helper in this specific task, it's a system composed by a control panel and a robotic arm with a flipping tool made of different materials.



Figure 2.6: Flippy[4]

2.1.3 Autonomous Kitchens

Apart from kitchen robots or recipe following helpers, there are already some solutions regarding autonomous kitchens. In those solutions the goal is to make the cooking process

completely automated, without the need of human inputs, after getting the instructions for the desired meal these kitchens cook by their own. Some examples are the follow:

1. **Spyce**[22]: Spyce is the name of a restaurant in Boston in which all the meals are cooked by an autonomous kitchen. It was developed by MIT students with the goal of lowering the meal price. This system works by having a conveyor that gets the ingredients out of the warehouse in the right amounts, putting them in bowls that rotate while cooking. These bowls are placed in a way that lets the customers see their meal being prepared, without the need of external inputs, except in the final steps, like plate presentation.



Figure 2.7: Spyce[9]

2. **Prometheus**[16]: Prometheus is an autonomous kitchen, but unlike the previous one, it's not built to serve in a restaurant, instead, it works in an area like a conventional kitchen. Having the recipes and actions programmed into it, it can cook the entire meal all by itself, it has a robotic arm ready to use all the necessary tools adapted to it, and another one which acts like a support for recipients. This way, with the recipes prepared to be followed by this kitchen alone, it can prepare full meals without any intervention by humans.



Figure 2.8: Prometheus [18]

3. **Moley**[11]: Moley is an autonomous kitchen, and like the previous one, it works in a closed environment where the human cannot interfere or customize anything. This kitchen also works with robotic arms, but in this case, the arms are not specifically used to hold or be tools as they resemble human arms. The recipes are made for this kitchen using sensors to record human actions, like wrist movement sensors. This makes it possible for this kitchen to use its arms to redo every single step, and prepare the meal in the exact same way as the person used to record the process.

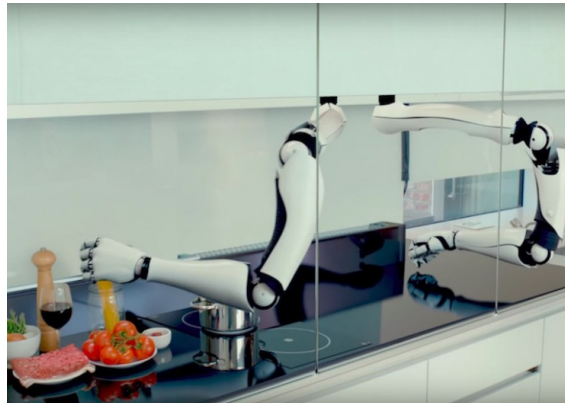


Figure 2.9: Moley [24]

2.2 Under Development Work

In this section we will look into under development projects that are in some way linked to kitchen automation, being it by autonomous kitchens, kitchen robots, control algorithms or architectures and explain how they work.

1. **Intelligent Kitchen: Cooking Support by LCD and Mobile Robot with IC-Labeled Objects**[12]: This kitchen's objective is to help the user through the cooking process by trying to predict his next move. By making use of a set of sensors, this system is capable of observe actions and record them, treating them as $a_0, a_1, a_2...$ (Action 0, Action1, Action 2...), and records a series of actions as **action pattern** $p_0=a_0, a_4, a_3$. The IC Labels on the objects provide information such as the object in which it is attached and where it is, with this it is possible to know if, for example, the human took a cup out of the cupboard. It then uses the PrefixSpan algorithm, which usage is to discover sequential patterns in sequence databases, to extract this recorded patterns and then foresee the next most probable move, based on the number of occurrences of a given action, after the previous one, on the whole set of recorded action patterns.

It's also possible to treat some actions as noise in a pattern, this mechanism is implemented to deal with the redundancy of human behaviour. If the probability of this

action is greater than the established threshold the system points out this action for the user to follow.

2. **Autonomous Robot in Aware Kitchen**[19]: This project aims to develop intelligent service robots that operate in standard human environments. Its goal is to enable a robot to do, like humans, all sensing, deliberation and action selection on board. Making use of the concept of ubiquitous robotics, which is the combination of ubiquitous computing with intelligent perception and control. Computing is ubiquitous when computing devices are distributed and embedded invisibly into the objects of everyday life.

By equipping a normal kitchen with sensors and RFID tags, it's transformed into an AwareKitchen. The kitchen observes and learns, through the sensors, how to do a process by recording information from a human doing it, to then create an activity model. The robot learns from these models and use them as resources to learn high-performance action routines. Making it possible to record human routines and apply that knowledge to the robot to get the desired tasks done.

This robot can perceive what's inside a cupboard the same way it perceives what is in its hand, by reading the information gathered by the sensors. During the process, the robot knows he has to take that specific cup, it then receives the information from the sensors to know where the cup is and to get it. This gives the robot the autonomy to take on a task without human intervention.

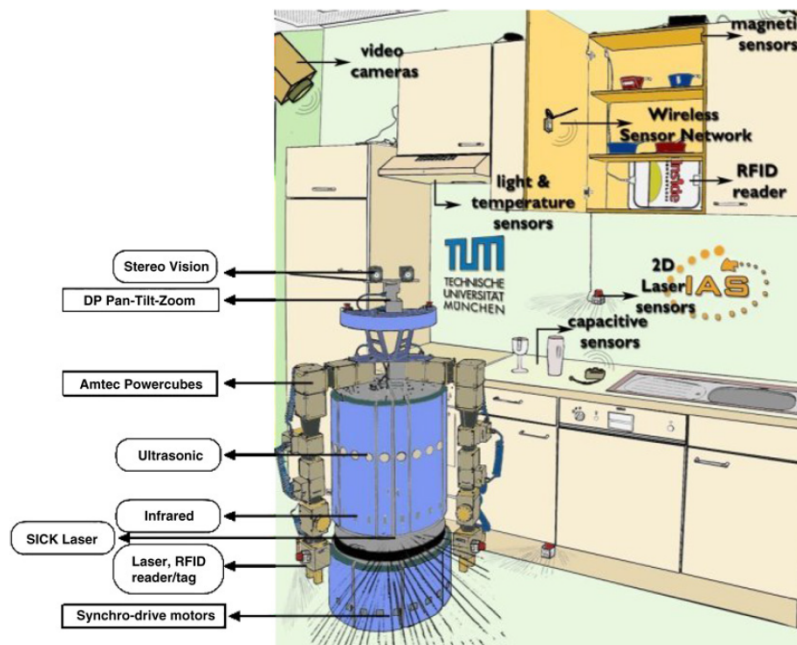


Figure 2.10: Autonomous Robot in Aware Kitchen

3. **Automatic fertilizer dispenser**[2]: This research proposes a solution to maintain soil fertility with nitrogen, phosphorous and potassium, by measuring the amount of nutrients present and provide them. The presence and quantity of nutrients is measured through sensors and chemical processes, and used to choose a suitable fertilizer.

The main system can be splitted into three main parts: sensor system, microcontroller and dispensary system. It is then divided into two subsystems: Sensor system, to estimate the presence of nutrients in the soil, and an intelligent system for estimation and control of the required amount of fertilizers.

Since this project is not related to kitchen areas, our main focus will be in the dispensary system, this system controls the flow of fertilizers to the soil. To measure the quantity of nutrients needed, a table is used to set the set the amount of nutrients depending on the soil fertility rate, setting levels as low, medium and high. The dispensary system consists of containers to store the fertilizers, a mixer compartment to collect them, valves and relays. Solenoid valves operated by relays are used to control the flow of fertilizers from the storage compartment to the mixer and from the mixer to the soil.

The algorithm for estimation of the three fertilizers is similar and can be shown by the diagram below. The addition of the fertilizer works by means of the ON time of a valve, where ' t_0 ' is the opening time in seconds to add maintenance dosage of the fertilizer, i.e. when the nutrient levels are within the limits. When there is high nutrient levels the used time is ' $t_0 - t_1$ ' for a reduced valve opening time. If the nutrient level is low, the time used is ' $t_0 + t_2$ ' for an increased valve opening time.

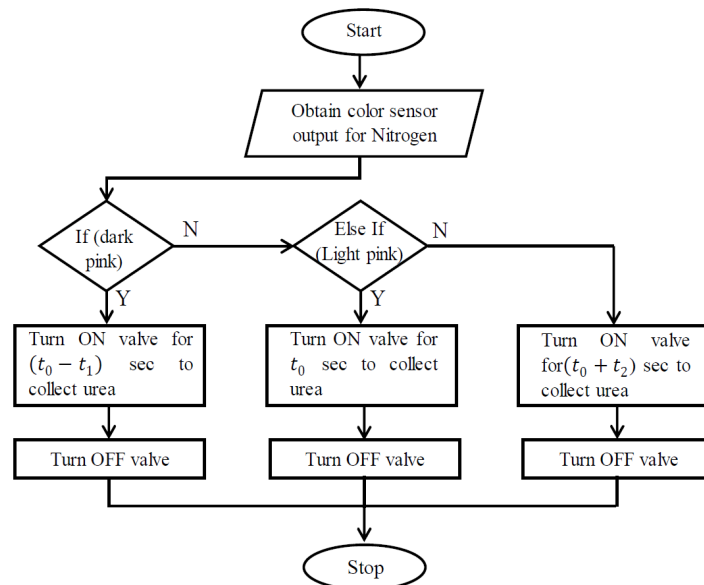


Figure 2.11: Control algorithm for estimation of nitrogen and addition of urea

2.3 Unsolved Problems

If we take a closer look into every solution explained in the previous section, it's possible to spot some problems yet to be solved, some steps where these solutions do not apply or pose a problem to the implementation of any change desired by the user. In terms of autonomy and time release from the user, the least optimal solutions would be the recipe following helpers. While they help the user during this task, and contribute to the good quality of the meal, thanks to the help provided by these applications to do every action as described in the recipe, their only function is to guide the user. This means that the user has to be present, and not to interfere punctually but to be the main actuator in the process.

On the other hand, the autonomous kitchens are the complete opposite, these solutions offer the most time saving ability by working in a closed environment. In this environment the process is entirely automated and user input free, as a down side, they also make it harder or impossible to adjust some step in the recipe or to change it completely to match the like of the user. Between these two extremes we have the kitchen robots, depending on the chosen machine it's possible to have more or less tools at our disposition, and with this choice the time saving ability changes as well.

However, independently of which is the chosen machine, user inputs are always necessary at some time, may them be by preparing the food beforehand, by needing to time a process or by putting the ingredients in the machine at the right time. All of these actions while permitting some adjustment in the recipe for the user to get the exact meal he wants, also require the user to spend time at some point of the process, this makes it impossible to be totally absent during the process which is the ultimate goal.

2.4 Used Technologies

This section will present the technologies used in the development of this dissertation, the control algorithm for the velocity controller, the model used for decision making and the tools for software implementation and hardware integration. We will take a look into these technologies and we will also understand where they were chosen to implement this solution.

2.4.1 Fuzzy Control [23]

As the main goal of this solution is to dispense the given ingredients, a way to improve its efficiency is to implement a velocity controller, the motor velocity should be optimised depending on the weight that's required, the one that's left to drop as well as the one already dispensed, if we look into the evolution of the dropped weight, it is not linear, as the ingredient can be clustered, or have empty spaces in the container. To overcome this drawbacks, a nonlinear controller is required as to better answer to the situation at hand, for this dissertation, the chosen algorithm is the fuzzy control algorithm.

Fuzzy control is non linear control algorithm, based on a set of rules and actions depending on them. A rule works as an IF-THEN statement and the words used to qualify the inputs and outputs are characterised by membership functions. We can take the next sentence as an example, "IF speed is low, THEN apply more force to the accelerator", in this sentence the words "low" and "more" are characterised by the membership functions. The starting point for the construction of a fuzzy system is to obtain a collection of fuzzy IF-THEN rules, and bring them together in a single system, either from human experts or based on domain knowledge. There are three types of fuzzy systems that are commonly used, pure fuzzy systems, Takagi-Sugeno-Kang (TSK) and fuzzy systems with fuzzifier and defuzzifier.

1. **Pure fuzzy systems:** In this system the basic configuration is purely done by IF-THEN rules. The fuzzy inference engine then combines these fuzzy rules a mapping from input fuzzy sets to output fuzzy sets, based on fuzzy logic principles (these sets are words in natural language). The problem of this method is that the rules are purely made from natural language whereas in engineering systems the inputs and outputs are valued variables;
2. **Takagi-Sugeno-Kang (TSK):** To solve the previous method problem, this one uses if-then rules in a different way, by introducing real values. Instead of using only natural language, this method gives actual values to the inputs or output, for example, "IF something is high THEN output is x", where x is an actual value instead of a natural word like "high".
The problems in this system are related to the use of actual values. When one of the parts of the rule is a mathematical formula it may not provide a natural framework to represent human knowledge. And not only that, but thanks to this, there is not much freedom left to apply different principles in fuzzy logic, and so the versatility of fuzzy systems is not well represented in this framework;
3. **Fuzzy systems with fuzzifier and defuzzifier:** In order to solve the problems posed to the previous systems, there is a third method which consists in adding a fuzzifier and a defuzzifier to the system. The fuzzifier transforms a real valued variable into a fuzzy set to the input, and the defuzzifier transforms a fuzzy set into a real valued variable for the output. This makes it possible to maintain the freedom level of the system while obtaining the desired behaviour for the system.

As we can see, the last system is built in order to solve the problems posed by the previous ones, as such, the one we will be using is the fuzzy system with fuzzifier and defuzzifier.

2.4.1.1 Membership Functions

As explained before, fuzzy systems are not as fuzzy as their name, the membership functions used to transform words into actual values have to be calculated, as such, in this subsection we will explore this process. There are various ways of representing a set and its members, we can list every element of that set or specify some conditions that represent the given set, therefore telling us if the element we are looking at is a part of that set or not:

$$A = \{x \in U \mid x \text{ meets some conditions}\} \quad (2.1)$$

There is yet another method called membership method. This method introduces a membership function, denoted by μ_A which output is either 1 or 0 if the element is respectively part or not of the set:

$$\mu_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \quad (2.2)$$

In the previous sub section we talked about the utilisation of fuzzy sets in order to use real valued variables in fuzzy logic. A fuzzy set is characterised by a membership function (μ_A) that instead of taking only the values of 1 or 0, takes values in the interval, $[0, 1]$. For example, if we want to evaluate if a number is close to 0, we can either set a threshold to consider this number close or not, making the output 0 or 1 depending on that value.

Another way to do this is to set a fuzzy set with a membership function that gives us how close is the value to 0. The fuzziness of a fuzzy set is mainly because the properties used to characterise it are usually fuzzy, for example, the numbers close to 0 can be evaluated in different ways.

With this in mind we can set different membership functions to characterise the same description, the equations below both describe numbers close to 0, the first one defines it as a triangular function while the second one defines it as a Gaussian function:

$$\mu_A(x) = \begin{cases} 0 & \text{if } x < -1 \\ x+1 & \text{if } -1 \leq x < 0 \\ 1-x & \text{if } 0 \leq x < 1 \\ 0 & \text{if } 1 \leq x \end{cases} \quad (2.3)$$

$$\mu_A(x) = e^{-x^2} \quad (2.4)$$

There are two approaches to define a membership function, we can either use the knowledge of human experts, or use data collected from various sensors to determine it. In both approaches we first define the structures of the membership functions and then fine tune the parameters based on the collected data. After defining these functions, we

need to use a fuzzy inference engine, this engine uses fuzzy logic principles to combine the fuzzy logic if-then rules into mapping from the fuzzy set A in U to a fuzzy set B in V. In order to use this membership functions as inputs and outputs we need to use and understand fuzzifiers and defuzzifiers.

2.4.1.2 Fuzzifiers

To transform a real valued variable into a membership function in a fuzzy set we need a fuzzifier.

There are some criteria to design this fuzzifier, at first, the fuzzifier should consider a point x^* where it has a large membership value. Second, If the input to the system has noise, it is important that the fuzzifier helps to suppress this noise. And third, the fuzzifier should help to simplify the computations made in the fuzzy inference engine.

We will now look into three types of fuzzifiers:

1. **Singleton fuzzifier:** This fuzzifier works in a way similar to a truth or false statement, it maps a real valued point $x^* \in U$ into a membership function which value is 1 at x^* and 0 at all other points in U:

$$\mu_A = \begin{cases} 1 & \text{if } x = x^* \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

2. **Gaussian fuzzifier:** The Gaussian fuzzifier maps $x^* \in U$ into a fuzzy set which has the Gaussian membership function

$$\mu_A(x) = e^{-\left(\frac{x_1 - x_1^*}{a_1}\right)^2} \star \dots \star e^{-\left(\frac{x_n - x_n^*}{a_n}\right)^2} \quad (2.6)$$

where a_i are positive parameters and the \star is usually chosen as algebraic product or min. The greater the value of a_i , the greater the noise cancelling ability;

3. **Triangular fuzzifier:** This fuzzifier maps $x^* \in U$ into a fuzzy set which has the following triangular membership function

$$\mu_A(x) = \begin{cases} \left(1 - \frac{|x_1 - x_1^*|}{b_1}\right) \star \dots \star \left(1 - \frac{|x_n - x_n^*|}{b_n}\right) & \text{if } |x_1 - x_1^*| \leq b_i, \quad i = 1, 2, \dots, n \\ 0 & \text{otherwise} \end{cases} \quad (2.7)$$

where b_i are positive parameters and the \star is usually chosen as algebraic product or min. The greater the value of b_i , the greater the noise cancelling ability.

We can make the following remarks about the three fuzzifiers:

1. The singleton fuzzifier greatly simplifies the computation involved in the fuzzy inference engine for any type of membership functions;

2. If the membership functions of the fuzzy rules are Gaussian or triangular, the Gaussian or triangular fuzzifiers, respectively, also simplify the computation in the inference engine;
3. The Gaussian and triangular fuzzifiers can suppress noise in the input, but the singleton cannot.

Depending on the membership functions defined, we may choose different fuzzifiers, to implement the one that best applies to the solution at hand.

2.4.1.3 Defuzzifiers

To get a useful and understandable output, it's needed to transform the membership function into a real valued output, to do this, we use a defuzzifier.

The defuzzifier is defined as a mapping from fuzzy set B in $V \subset R$ which is the output from the inference engine. The task of the fuzzifier is to specify a point in V that best represents the fuzzy set B' (which is the union or intersection of M fuzzy sets). There is a number of choices to determine the representing point. When choosing a defuzzification scheme the following three criteria should be taken into consideration:

1. **Plausibility:** The point y^* should represent B' from an intuitive point of view, for example, in the middle of the support of B' , or has a high degree of membership in B' ;
2. **Computational simplicity:** This is an important criterion to achieve a real-time operation which is needed in fuzzy controllers;
3. **Continuity:** A small change in B' should not result in a large change in y^* .

Next, with this criteria in mind, three types of defuzzifiers are proposed:

1. **Center of gravity defuzzifier:** This type of defuzzifier specifies y^* as the center of the area covered by the membership function, that is,

$$\frac{\int_V y \mu_{B'}(y) dy}{\int_V \mu_{B'}(y) dy} \quad (2.8)$$

where \int_V is the conventional integral.

If we take $\mu_{B'}(y)$ as the probability density function of a random variable, this defuzzifier gives the mean value of the random variable. It is also possible to eliminate the $y \in V$, whose membership values are too small, getting V_α defined as

$$V_\alpha = \{y \in V | \mu_{B'}(y) \geq \alpha\} \quad (2.9)$$

and α is a constant. While this defuzzifier has an intuitive plausibility it is also computationally intensive.

2. **Center average defuzzifier:** As B' is a union or intersection of M fuzzy sets, a good approximation of 2.8 is the weighted average of the centers of M fuzzy sets. Let \bar{y}^l be the center of the l 'th fuzzy set and w_l be its weight, this defuzzifier determines y^* as

$$y^* = \frac{\sum_{l=1}^M \bar{y}^l w_l}{\sum_{l=1}^M w_l} \quad (2.10)$$

This is the most commonly used defuzzifier. It is computationally simple and intuitively plausible.

3. **Maximum defuzzifier:** This defuzzifier chooses y^* as the point in V at which $\mu_{B'}(y)$ achieves its maximum value. Defining the set

$$hgt(B') = \{y \in V | \mu_{B'}(y) = \sup_{y \in V} \mu_{B'}(y)\} \quad (2.11)$$

$$y^* = \text{any point in } hgt(B') \quad (2.12)$$

The mean of the maximum fuzzifier is

$$y^* = \frac{\int_{hgt(B')} y dy}{\int_{hgt(B')} dy} \quad (2.13)$$

The maximum defuzzifiers are intuitively plausible and computationally simple. However, small changes in B' may result in large changes in y^* .

We may choose the defuzzifier that applies the best to our needs, evaluating each one.

2.4.1.4 Inference Engine

As explained at the start of this section, inference engines are used to map inputs and outputs in their respective fuzzy sets. To do this there are some commonly used formulas to define the outputs from A' to B' :

1. **Product inference engine:**

$$\mu'_B(y) = \max_{l=1}^M [\sup_{X \in U} (\mu_{A'}(x) \prod_{i=1}^n \mu_{A_i^l}(x_i) \mu_{B'}(y))] \quad (2.14)$$

2. **Minimum inference engine:**

$$\mu'_B(y) = \max_{l=1}^M [\sup_{X \in U} \min(\mu_{A'}(X), \mu_{A_1^l}(x_1), \mu_{A_2^l}(x_2), \dots, \mu_{A_n^l}(x_n), \mu_{B'}(y))] \quad (2.15)$$

This two inference engines are the most commonly used for fuzzy control systems, their main advantage is their computational simplicity. However, as a disadvantage is the fact that if at some $x \in U$ the $\mu_{A_i^l}(x_i)$'s are very small, then the $\mu_{B'}(y)$ will be very small. The next three fuzzy inference engines overcome this problem.

1. Lukasiewicz inference engine:

$$\mu'_B(y) = \min_{l=1}^M \left\{ \sup_{X \in U} \min[\mu_{A'}(X), 1 - \min_{i=1}^n (\mu_{A_i^l}(x_i)) + \mu_{B^l}(y)] \right\} \quad (2.16)$$

2. Zadeh inference engine:

$$\mu'_B(y) = \min_{l=1}^M \left\{ \sup_{X \in U} \min[\mu_{A'}(X), \max(\min(\mu_{A_1^l}(x_1), \dots, \mu_{A_n^l}(x_n), \mu_{B^l}(y)), 1 - \min_{i=1}^n (\mu_{A_i^l}(x_i))) \right\} \quad (2.17)$$

3. Dienes-Resher inference engine:

$$\mu'_B(y) = \min_{l=1}^M \left\{ \sup_{X \in U} \min[\mu_{A'}(X), \max(1 - \min_{i=1}^n (\mu_{A_i^l}(x_i)), \mu_{B^l}(y))] \right\} \quad (2.18)$$

If we take a look into the inference engines shown previously, we can see some differences between the two groups. If the membership value of the input at point x^* is very small, then the product and the minimum inference engines give very small membership values. Whereas the Lukasiewicz, Zadeh and Dienes-Resher inference engines give very large membership values. It is also important to keep in mind that the product inference engine gives the smallest output while the Lukasiewicz inference engine gives the largest membership value, the other three inference engines are in between.

2.4.2 ROS [14]

ROS is an open-source, meta-operating system for robots. It provides the services you expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. Some of the issues in the development of software for robots where ROS can help are:

1. Distributed computation, many modern robot systems rely on software that spans many different processes and runs across several different computers, for example, Some robots carry multiple computers, each of which controls a subset of the robot's sensors or actuators, or when multiple robots attempt to cooperate on a shared task, they often need to communicate with one another to coordinate their efforts. The common thread through these cases is a need for communication between multiple processes that may or may not live on the same computer. ROS provides two relatively simple, seamless mechanisms for this kind of communication
2. Software reuse, the rapid progress of robotics research has resulted in a growing collection of good algorithms for common tasks such as navigation, motion planning, mapping, and many others. Of course, the existence of these algorithms is only truly useful if there is a way to apply them in new contexts, without the need to reimplement each algorithm for each new system.
3. ROS's standard packages provide stable, debugged implementations of many important robotics algorithms.
4. Its message passing interface is becoming a standard for robot software interoperability, which means that ROS interfaces to both the latest hardware and to implementations of cutting edge algorithms are quite often available. This sort of uniform interface greatly reduces the effort to connect different blocks of code.
5. Rapid testing, one of the reasons that software development for robots is often more challenging than other kinds of development is that testing can be time consuming and error-prone. Physical robots may not always be available to work with, and when they are, the process is sometimes slow and finicky. Working with ROS provides two effective workarounds to this problem.

ROS systems separate the low-level direct control of the hardware and high-level processing and decision making into separate programs. Because of this separation, we can temporarily replace those low-level programs (and their corresponding hardware) with a simulator, to test the behaviour of the high-level part of the system.

ROS also provides a simple way to record and play back sensor data and other kinds of messages. This facility means that we can obtain more leverage from the time we do

spend operating a physical robot. By recording the robot's sensor data, we can replay it many times to test different ways of processing that same data. In ROS, these recordings are called "bags" and a tool called rosbag is used to record and replay them.

Next we will be looking into some of ROS basic concepts:

1. **Packages:** All ROS software is organized into packages. A ROS package is a collection of files, generally including both executables and supporting files, that serves a specific purpose. Each package is defined by a manifest, this file defines some details about the package, including its name, version, maintainer, and dependencies. The directory containing package.xml is called the package directory. (In fact, this is the definition of a ROS package, any directory that ROS can find that contains a file named package.xml is a package directory.
2. **Nodes and Ros Master:** A running instance of a ROS program is called a node. One of the basic goals of ROS is to enable roboticists to design software as a collection of small, mostly independent programs called nodes that all run at the same time. For this to work, those nodes must be able to communicate with one another. The part of ROS that facilitates this communication is called the ROS master. The master should be running for the entire time that ROS is being used. Unlike killing and restarting the master, killing and restarting a node usually does not have a major impact on other nodes, even for nodes that are exchanging messages, those connections would be dropped when the node is killed and reestablished when the node restarts.
3. **Topics and Messages:** The primary mechanism that ROS nodes use to communicate is to send messages. Messages in ROS are organized into named topics. The way it works is as follow, a node that wants to share information will publish messages on the appropriate topic or topics, a node that wants to receive information will subscribe to the topic or topics that it's interested in. The ROS master takes care of ensuring that publishers and subscribers can find each other, the messages themselves are sent directly from publisher to subscriber.
4. **Services:** Even though messages are the primary method for communication in ROS, they have some limitations, an alternative method of communication is called service calls. Service calls differ from messages in two ways:
 - a) Service calls are bi-directional. One node sends information to another node and waits for a response. Information flows in both directions. In contrast, when a message is published, there is no response or guarantee that anyone is subscribing to those messages;
 - b) Service calls implement one-to-one communication. Each service call is initiated by one node, and the response goes back to that same node. On the other

hand, each message is associated with a topic that might have many publishers and many subscribers.

2.4.3 Node-Red [13]

Node-Red is a programming tool for wiring together hardware devices, API's and online services. It provides a browser-based editor that makes it easy to connect together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

It is built on Node.js, taking advantage of its event-driven, non-blocking model, making it ideal to run on low-cost hardware, such as the raspberry pi as well as in the cloud. As a way to easily cooperate while developing, the created flows are stored in json, which can be easily stored, imported and exported. Next we will explore some key concepts:

1. **Flows:** Node-red programs or flows are a collection wired together to exchange messages. If we take a deeper look at it, a flow is a list of JavaScript objects that describe the nodes and their configurations, as well as the list of downstream nodes they are connected to, the wires;
2. **Messages:** As said in the previous point, messages passed between node in Node-red are, by convention, Javascript Objects called "msg", consisting of a set of named properties. These messages contain a payload which is often their main property, although nodes can also attach other properties to a message, that may be used to carry other information into the next node in the flow. Messages are the primary data structure used in node-red and are, in most cases, the only data that a node has to work with when it is activated. This ensures that a Node-RED flow is conceptually clean and stateless, each node is self-contained, working with input messages and creating output messages, this means that the effect of a node's processing is either contained in its output messages, or caused by internal node logic that changes external things such as files, IO pins on the Raspberry Pi or Dropbox files; there are no side effects that could affect the behaviour of other nodes or subsequent calls to the same node. This is one of the key advantages of a flow-based language. Because nodes are self contained and typically only interact with other nodes using messages, you can be sure that they have no unintended side effects and so can be safely re-used when you create new flows. To safely re-use these blocks of code it is only needed to use a copy of the initial node.
3. **Nodes:** Nodes are the primary building block of node-red flows. When a flow is running messages are generated, consumed and processed by nodes. Nodes consist of code that runs in the node-red service and an HTML file consisting of a description of the node, so that it appears in the node pane with a category, colour, name and an icon, code to configure the node, and help text. Nodes can have at most one input, and zero or more outputs. During the initialization process, the node is loaded into

the Node RED service. When the browser accesses the Node RED editor, the code for the installed nodes is loaded into the editor page. There are 3 core node types:

- a) **Input Nodes:** Generate messages for downstream nodes;
- b) **Output Nodes:** Consume messages, either to an external service or pin on a device, or generate response messages;
- c) **Processing Nodes:** Nodes that process data in some way emitting new or modified messages.

In addition to these core types, there are two more categories:

- a) **Credential Nodes:** These are nodes that hold the credentials used by one or more nodes to connect to an outside system or service, these are created when it's needed to configure a node that requires credentials such as an API key or name and password. Once created, a credentials node can be reused by other nodes of the same type to connect to similar protocols or services. Even when all nodes that use those credentials are deleted from your flow, the credentials node will remain, so it's a good idea to remove unused credentials nodes when they are no longer needed.
 - b) **User Created Nodes:** Programmable nodes such as function nodes, or sub-flows are nodes created by you to do some custom work or reuse flow segments in other flows.
4. **Wires:** Wires define the connections between node input and output endpoints in a flow. They typically connect the output endpoints of nodes to inputs of downstream nodes indicating that messages generated by one node should be processed by the connected node next. Note that it is possible to connect more than one node to an endpoint using wires. When multiple nodes are connected to an output endpoint, messages are sent to each connected node in turn in the order they were wired to the output. When more than one node output is connected to an input endpoint, messages from any of those nodes will be processed by the connected node when they arrive. It is also possible to connect downstream nodes to upstream nodes to form loops.
5. **Context:** While generally messages are the only way to get data into and out of nodes, there is one exception to this rule which is available to function nodes. function nodes have access to a special object called context that is used to hold data in memory that lasts from one message arriving to the next[4]. This is important for nodes that need to maintain an index or count or sum data in messages. In addition to this local context, a global context `context.global` is available for sharing data between all of the function nodes of a flow. Some use cases for context will be covered when the function node is discussed in more detail.

6. **Function Nodes:** The function node is a node that's used when there is no existing node dedicated to the task at hand. It's used for doing specialised data processing or formatting for example. As the name implies, a function node exposes a single JavaScript function. Using the function node, it's possible to write our own JavaScript code that runs against the messages passed in and returns zero or more messages to downstream nodes for processing. Function nodes are written using the built-in code editor.
7. **Sub-flows:** As said in previous points, flows can be stored in nodes and become a sub-flow. While it's possible to save flow segments it's much better to have them as nodes, this a level of encapsulation and information hiding that importing saved flows doesn't offer. Encapsulation means that it is organised into a single node that can be referred to using a single name, information hiding means that the inner workings of the sub-flow are hidden, it's possible to change how the sub-flow does its job and the flows that use it will not change and don't need to take that change into account.

2.4.4 Behaviour Trees [7]

A Behaviour Tree is a mathematical model used to describe a plan or sequence of tasks. Their main advantage is the capability to create very complex tasks composed of simpler tasks that can be easily abstracted. Behaviour Trees are also very readable and easy to understand and debug making them less error-prone than other approaches. Behaviour trees can be pictured as usual trees, they start from the root and grow to the leafs, these leafs and roots are nodes, and there are six types of nodes, four control flow nodes, Fallback, Sequence, Parallel and Decorator, and two execution nodes, Action and Condition. The root is the node without parents, the nodes that don't have children nodes are the leafs, these nodes must be one of the execution type nodes, Action or Condition.

The execution of the behaviour tree works in a control loop and time steps, in each time step of the control loop the root of the behaviour tree is ticked. This tick is then progressed down the tree according to the types of each node. Once a tick reaches a leaf node, which is an action or condition node, the node makes the desired computation, possibly affecting some state or variable and then returns either success, failure or running. This return status is then progressed up the tree, back to the root. If a running node does no longer receive a tick, it has to stop (preempted). The stopping of a preempted action is implemented by the halt procedure. Before advancing further into the explanation of the behaviour tree, we will look into each type of node.

1. **Fallback:** Fallback nodes are used to find and execute the first child that does not fail. This node will return immediately with a status code of success or running when one of its children returns success or running. The children are ticked in order of importance, from left to right.

2. **Sequence:** As an opposite of the previous node, Sequence nodes are used to find and execute the first child that has not yet succeeded. A sequence node will return immediately with a status code of failure or running when one of its children returns failure or running.
3. **Parallel:** The parallel node ticks its children in parallel and returns success if $M \leq N$ children return success, it returns failure if $N - M + 1$ children return failure, and it returns running otherwise, where N is the number of children and $M \leq N$ is a user defined threshold.
4. **Decorator:** The decorator node manipulates the return status of its child according to the policy defined by the user, some common examples of this are, the decorator retry which retries the execution of a node if it fails and the decorator negation that inverts the success or failure outcome.
5. **Action:** An Action node performs an action, and returns Success if the action is completed, Failure if it can not be completed and Running if completion is under way.
6. **Condition:** A Condition node determines if a condition C has been met. Conditions are technically a subset of the Actions, but are given a separate category to improve readability of the behaviour tree and emphasize the fact that they never return running and do not change any internal states or variables.

AUTOMATIC DISPENSER

This chapter will explain how the automatic dispenser system will be implemented, we will first look into the software architecture followed by the hardware architecture and implementation and finally the software implementation. A diagram is shown in figure 3.1 as to show the connections between different software components, followed by an explanation on their interactions, what data is sent and received in each component and how is this data structured. For the hardware section, the physical behaviour of the system will be explained, this behaviour includes how it is built, how the physical system answers to the software instructions, and images that will show the full setup. The software implementation will show how the software works and how the tasks are processed, making use of diagrams and sequence charts.

3.1 Software Architecture

This machine's main goal is to follow a given recipe, going through every step in the list of steps that constitute the recipe, as it is possible to see in figure 3.1.

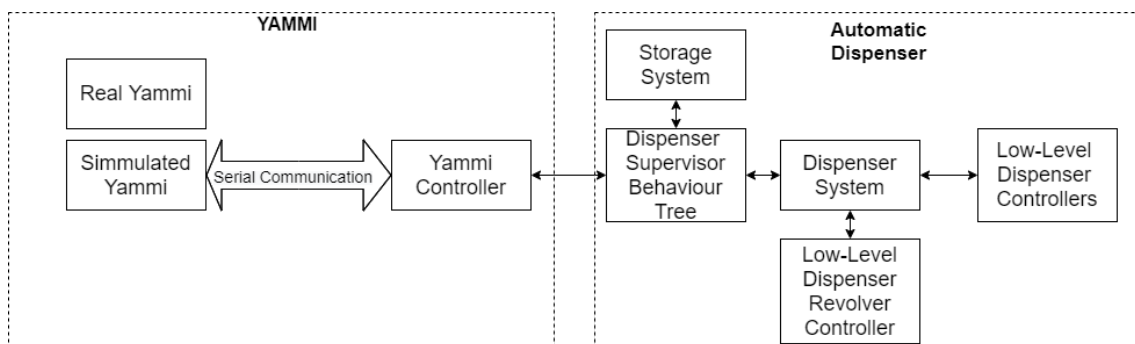


Figure 3.1: Software Architecture with all the modules and their direct connections

The dispenser module can be splitted in some submodules, these modules are:

1. **Dispenser Supervisor Behaviour Tree:** when a task arrives, this module, making use of the Behaviour Trees model, chooses which services must be used, these services is chosen based on the task at hand. If the task at hand requires the system to retrieve some information from the stored ingredients it has to communicate with the Storage System;
2. **Storage System:** this module holds the information of the ingredients and their container, this module can be used to either check an ingredient status, using the *CheckIngredient.srv*, or to update the storage information, using the *UpdateStorage.srv*;
3. **Dispenser System:** this module handles all operations with the dispensers, if a task requires an interaction with this system it must be done through the *Dispense.srv*. This system is responsible for the control of the platform revolver and dispenser containers. It can receive "Dispense" requests for a given ingredient and dispenser ID and sends commands to the motor controllers to activate for a certain time. When there is a need to rotate the platform, either to reset it or to position another container to dispense, this system uses the message type *JointState*, to send the desired rotation to the Dispenser Revolver Controller, if what is needed is to dispense an ingredient it will communicate with the Dispenser Controller Node-red flow;
4. **Revolver Controller:** when a message of the type *JointState* arrives this controller it will actuate on the motor that's responsible for the platform rotation;
5. **Dispenser Controller Node-red flow:** if the task arrives to this module trough the service *Dispense.srv*, this flow will retrieve the ingredient and container information, with this it will set the desired weight and will communicate with the arduino for the motors to start. The arduinos are then responsible to stop that motor when the weight is reached.

3.2 Hardware

To understand the hardware implementation of this system we will start by looking at the connections between the system components, these connections are shown in figure 3.2.

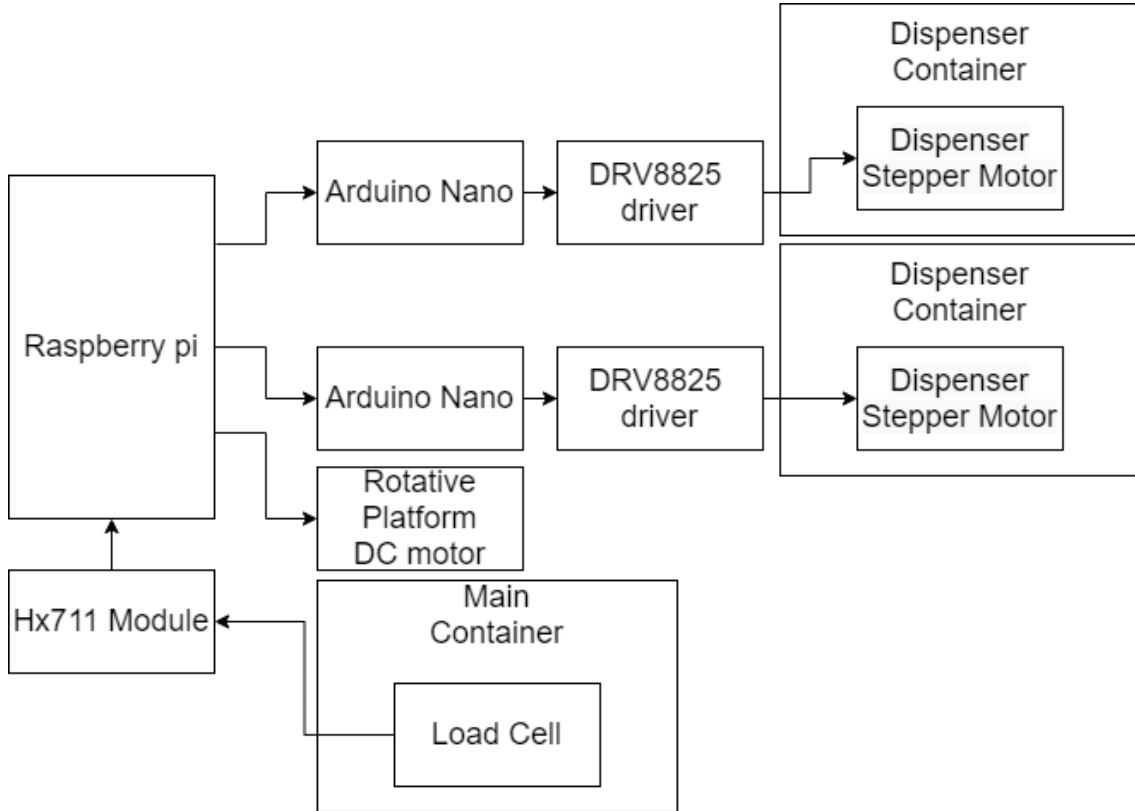


Figure 3.2: Hardware components connections diagram

As already said, the dispenser will be based on a rotative platform where the containers are placed, this platform can rotate so the right ingredient container stays above the main container where the ingredients are to drop. The dispensers in the platform are shown in the figures 3.3 and 3.4.

The rotation of the platform is handled by DC motors, this way, with the ROS implementation, it is possible to set the angle of the motor rotation, and set the platform position with precision, making it possible to correctly set the right container to drop or to reset the platform to its original position.

There are two types of containers, the ones for solid ingredients and the ones for liquid ingredients. The solid type ones, are made in the shape of a box, with a hole on the opposite for the ingredient to drop into the main container by means of an endless screw which function is to push the ingredient through it. The liquid type ones are made through a common container that can hold liquids and work by means of a valve, which pulls the liquid out of its container to the main one, there is also a small container, in order to store liquids that usually are required in less quantity, as can be seen in figure 3.4.

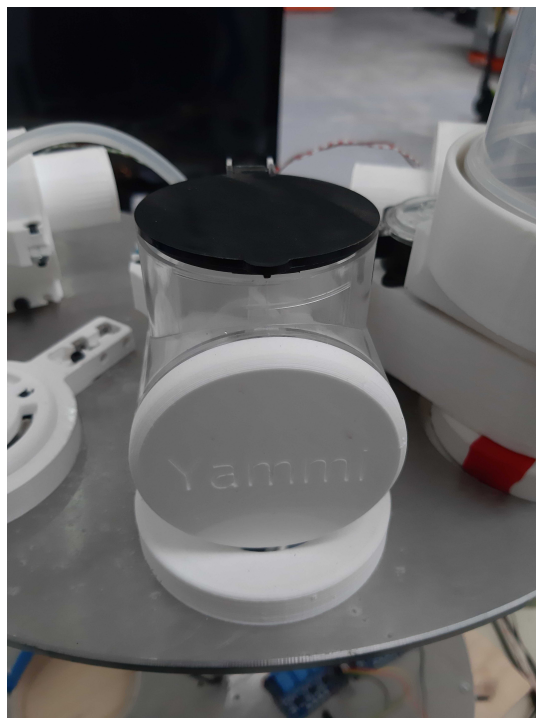


Figure 3.3: Solid ingredients dispenser on the platform



Figure 3.4: Liquid ingredients dispenser on the platform

The solid type containers will have their endless screw attached to a stepper motor, this way it is possible to control the velocity at which the screw rotates and consequently the quantity and velocity at which the ingredient is provided. The liquid type containers will also have a stepper motor attached to their valve, and in the same way as the other containers, this stepper motor can be controlled in order to control the velocity at which the liquid is poured. These stepper motors are controlled through a DRV8825 stepper motor driver, and these drivers are connected to arduinos, in order to set functions to control their velocity. The arduino and drives connections as well as both solids dispensers, the with the dc motor and the one with the stepper motor can be seen in the figures 3.5, 3.6, 3.7.

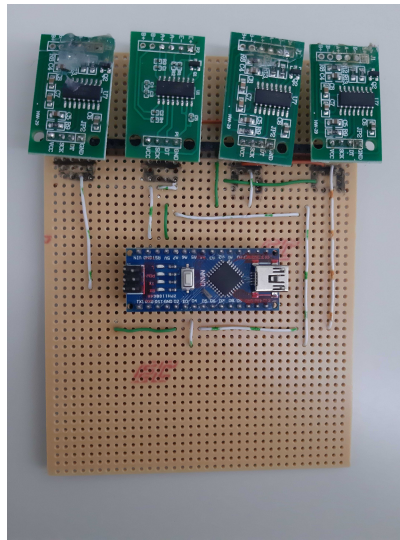


Figure 3.5: Arduino and DRV8825 stepper motor driver connections

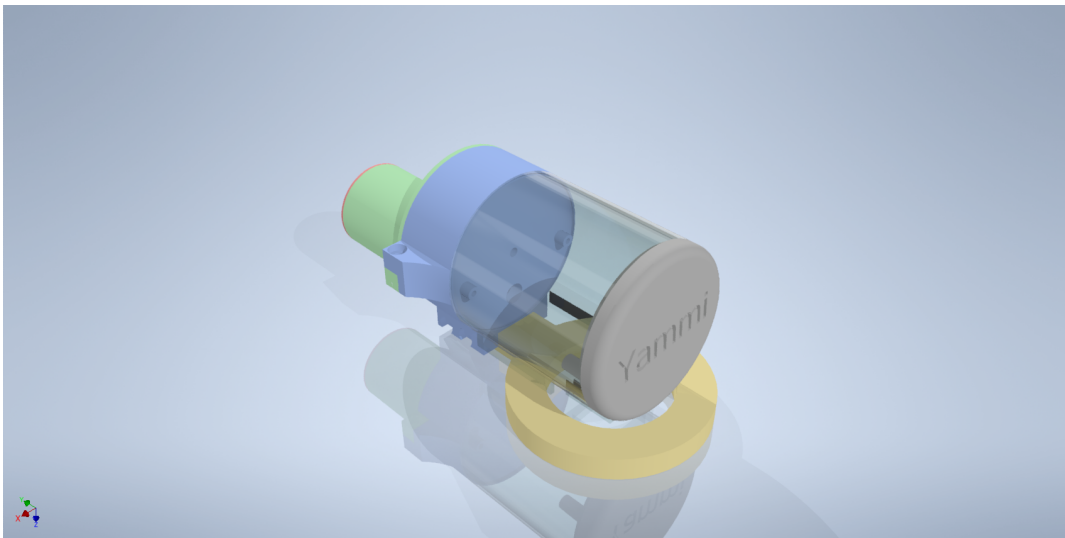


Figure 3.6: DC motor dispenser 3D model

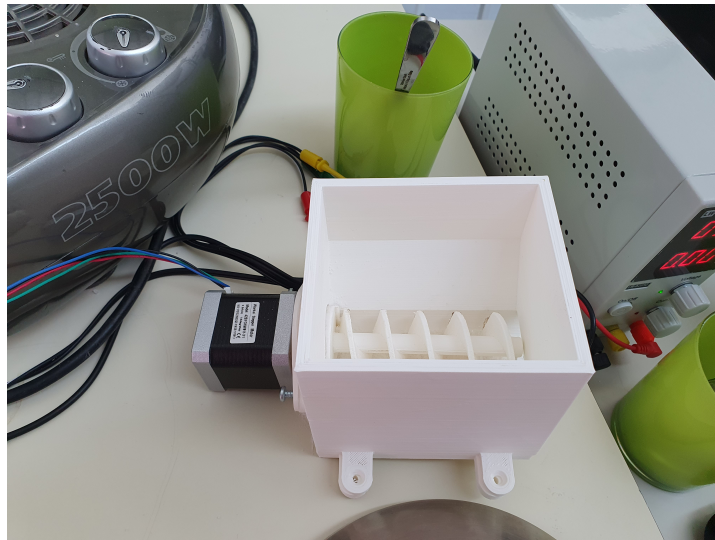


Figure 3.7: Stepper motor dispenser prototype

In addition, the main container will have a load cell, in order to track its weight so that the motors can be controlled and it knows when to stop the process, this load cell is connected to the raspberry pi by the means of an HX711 Module which amplifies and converts the analog signal in order to make it readable to the raspberry pi. Finally every arduino and DC motor will be connected to the main processor which is the raspberry pi, where the ROS services and node-red flows are running.

3.3 Software

To better understand the software in this system we will look into the architecture shown before, but this time with a little more detail as can be seen in figure 3.8, this figure while similar to the one in the architecture section, it also shows the services and node-red flows used during the system operation. This systems software is built based on two modules, the dispenser supervisor behaviour tree and the dispenser system, one controls the flow of operations and the other handles all dispense and revolver tasks. Both can be seen in the diagram below.

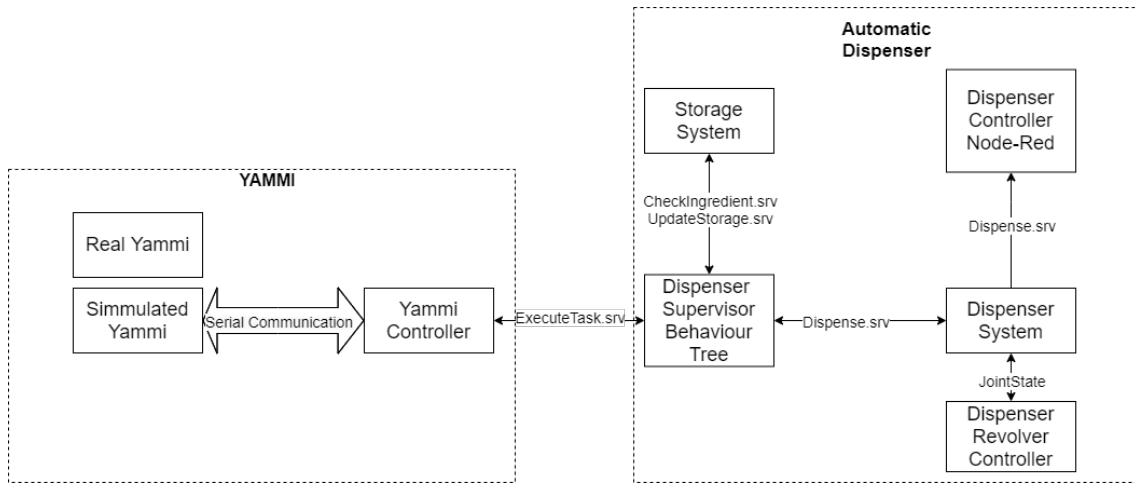


Figure 3.8: Software Architecture with all the modules, connections and services

The process starts when a recipe is selected from the dropdown in the main machine, when this recipe is selected, the list of steps which constitutes the recipe is set as the operation flow, as explained before, a step is built of different parameters that depend on the task at hand. If the task is other than the ones handled by the dispenser, for example heating or mixing, the task is not sent to the dispenser supervisor, instead, it stays the main machine controller, in order to execute it, these tasks can be, for example, heating or mixing. If the task is one that the dispenser module can handle, it is sent to the dispenser supervisor one step at a time through the *ExecuteTasksrv*, when the request is received by the supervisor through the *ExecuteTasksrv* service it stores it in a queue.

The dispenser supervisor follows a behaviour tree algorithm, following nodes of the tree with them returning success, running, or failure, defining the next step depending on the return status of each node, this response is different depending on the used control node. This algorithm is used in order to get the next task and execute every step in the most efficient way.

The supervisor continues to the get next task node, this node checks if the queue has any pending task, if not, it keeps reentering the node and checking the queue until a new task shows up, when a task shows up, it sets it as the next task and follows the nodes down the tree. After explaining the way the flow of information and actions works, we can take

a closer look on how the services interact.

The Dispenser GUI Dashboard interacts with the Automatic Dispenser primarily through two distinct services, *ExecuteTask* and *ReportTaskCompletion*. Additionally it also uses *Trigger* and *SetBool* services in the Calibration process. The *ExecuteTask* service works as a direct way to control the automatic dispenser and the ingredients within its containers. There are four different types of tasks the dispenser can perform, *Dispense*, *Refill*, *Empty* and *Reset revolver*. The following sequence charts and explanations reflect the flow of each process.

To create a recipe the Yammy Controller must beforehand verify if the required ingredients are available for the chosen recipe. In order to do that the *CheckRecipe* service must be used. If an ingredient is either missing or in lower quantity than required for the recipe the response will arrive with an *ErrorId* (*UNSUFFICIENTQTY/INGREDIENT-MISMATCH*) and an error message stating the origin of the error. If there is no error the controller can then send an *ExecuteTask* (*DISPENSE*) stating the ingredients names and quantities as they are required in the recipe. This process can be seen in the figure 3.9. The first node the behaviour tree ticks is the “*IsReset*” node, to check if the request type is *Reset revolver*, to reset the dispenser positions, if it is, it enters the “*Dispense*” node, which, using the *dispense* service, sends a request to the dispenser system for the dispenser positions to be reset. This process can be seen in the figure 3.10. When that is not the case, the next sequence branch checks if the request type is *Empty*, if so, it completely empties the dispenser given in dispenser ID. The current quantity is set to 0 in the Storage System and the dispenser is free and can be filled with other ingredients. This process can be seen in the figure 3.11.

If that is not the request type, the next sequence branch starts by checking if the request type is *Refill*, if so, it initiates the refill process. The quantity of a given ingredient is added to the storage system, if the dispenser was empty prior to the refill request, the calibration process takes place. The dispenser will automatically perform calibration, prompting the user to confirm the calibration was finished correctly using the *Trigger* service. The GUI should then respond to the supervisor using the *SetBool* service with the user selection. In case the calibration is not finished yet another *Trigger* for the GUI is sent and the process repeats itself until it finishes successfully. This process can be seen in the figure 3.12.

Every time that the system uses the *UpdateStoragesrv* service before updating with the new value, being it a new ingredient, an old ingredient but in a new container, or just a refill of a container, the system always verifies if the type of the container is the same as the ingredient, if there is already an ingredient in the container checks if the one being added is the as the one already there, and in case of *dispense*, after checking that it’s the right ingredient, it checks if it has at least the needed quantity. If the request type is “*Dispense*”, if it is, it checks if the ingredient is available, this checks if the ingredient is in the system storage, if it has at least the needed quantity and if the ingredient in the selected container matches the one in the parameters of the task(both name and container

number sent in the step), if so, it tries to dispense the required ingredient using the *Dispense.srv* to the dispenser system that will use the other *Dispense.srv* to communicate with the Dispenser Controller node-red, both the sequence chart and the node-red flow can be seen in figure 3.13.

When the request is received by the ROS service node the flow checks which container should dispense, and how much should be delivered, activating the motor of that container, while the arduino connected to it controls the motor speed and direction, while always checking the weight. When the weight is reached the flow builds the response message and sends it back to the dispenser supervisor through the dispenser service, returning success if the ingredient was correctly dispensed, or failure, if it was impossible for some reason. If the return state was success, it updates the ingredient storage and follows the tree to report task completion. If neither of the branches return a successful task completion, the system informs the user that the task was not completed. When the task is completed, the main machine controller is informed through the *ReportTaskCompletion*, it checks if it's from a sent instruction, if not, it shows an error to the user, otherwise it sets the task as fulfilled and continues to the next one.

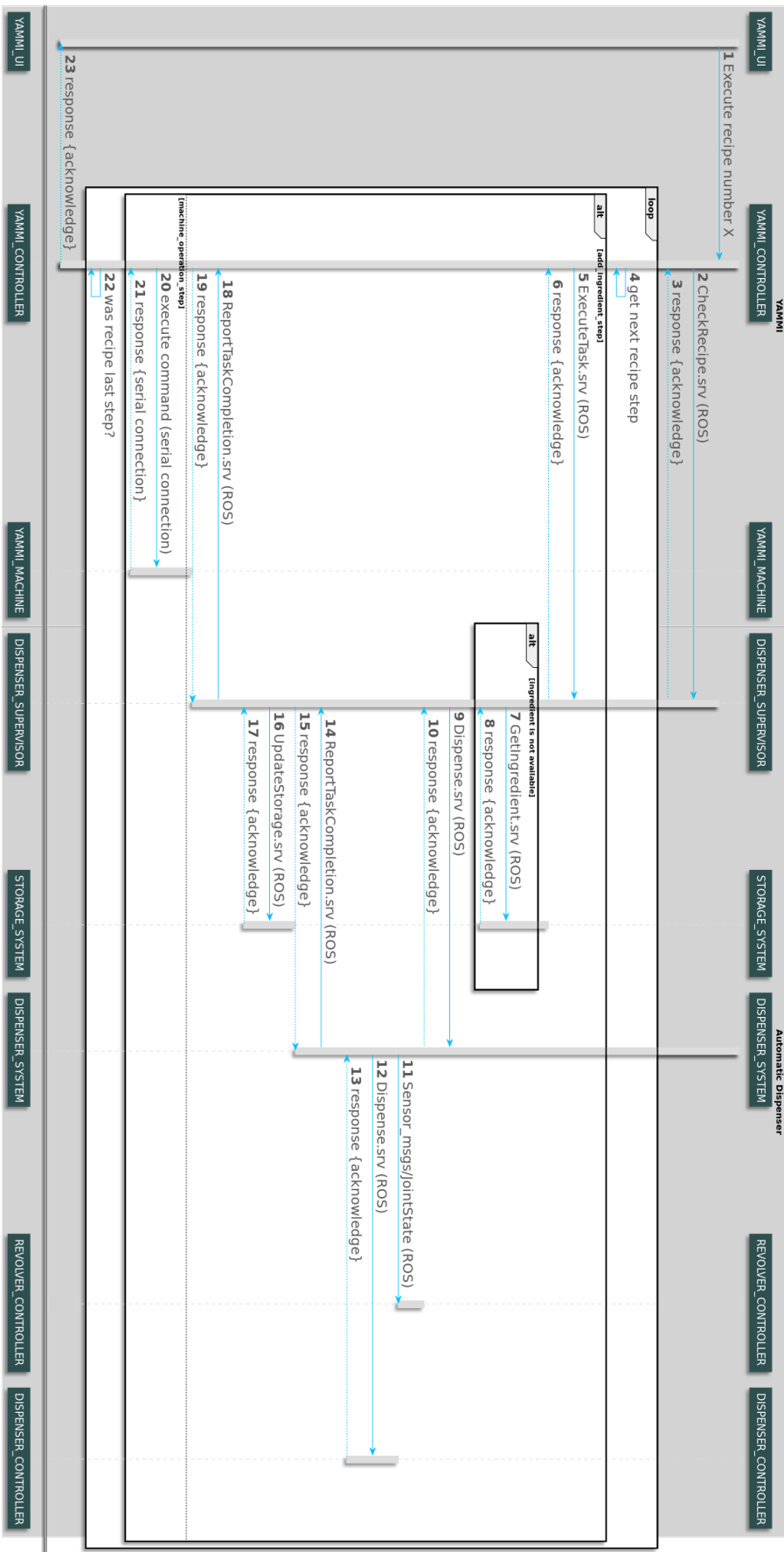


Figure 3.9: Recipe sequence chart

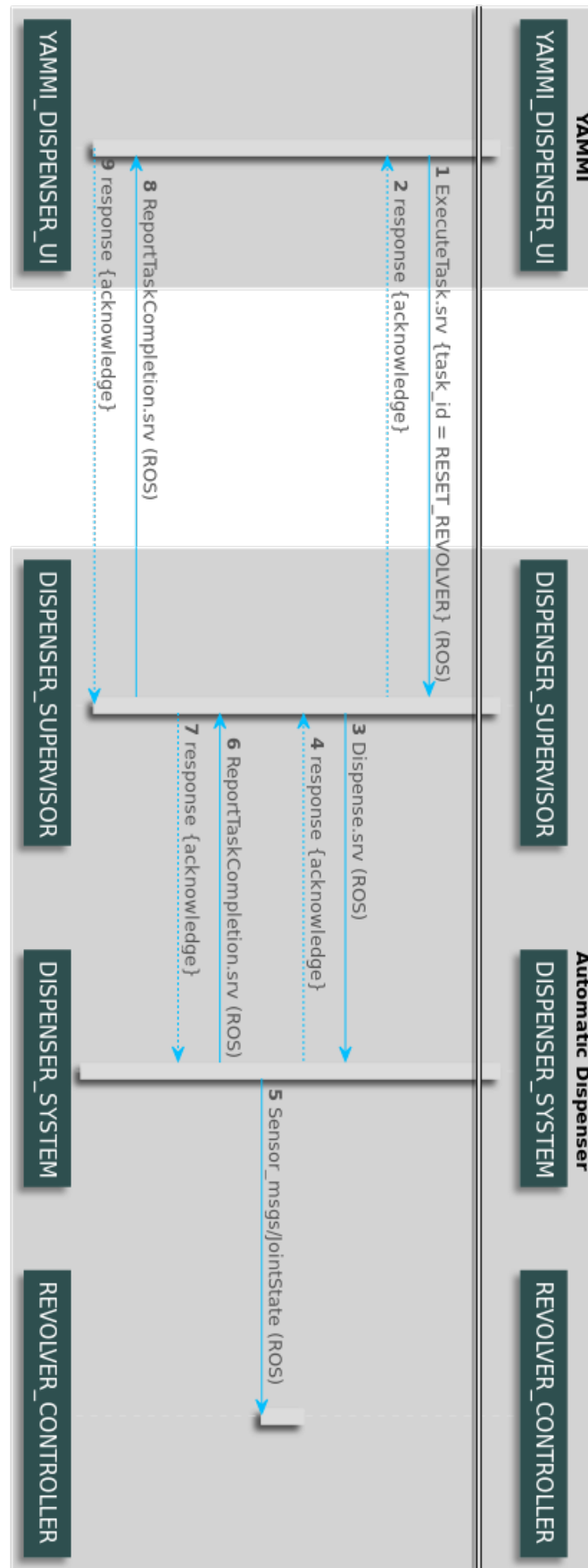


Figure 3.10: Reset revolver sequence chart

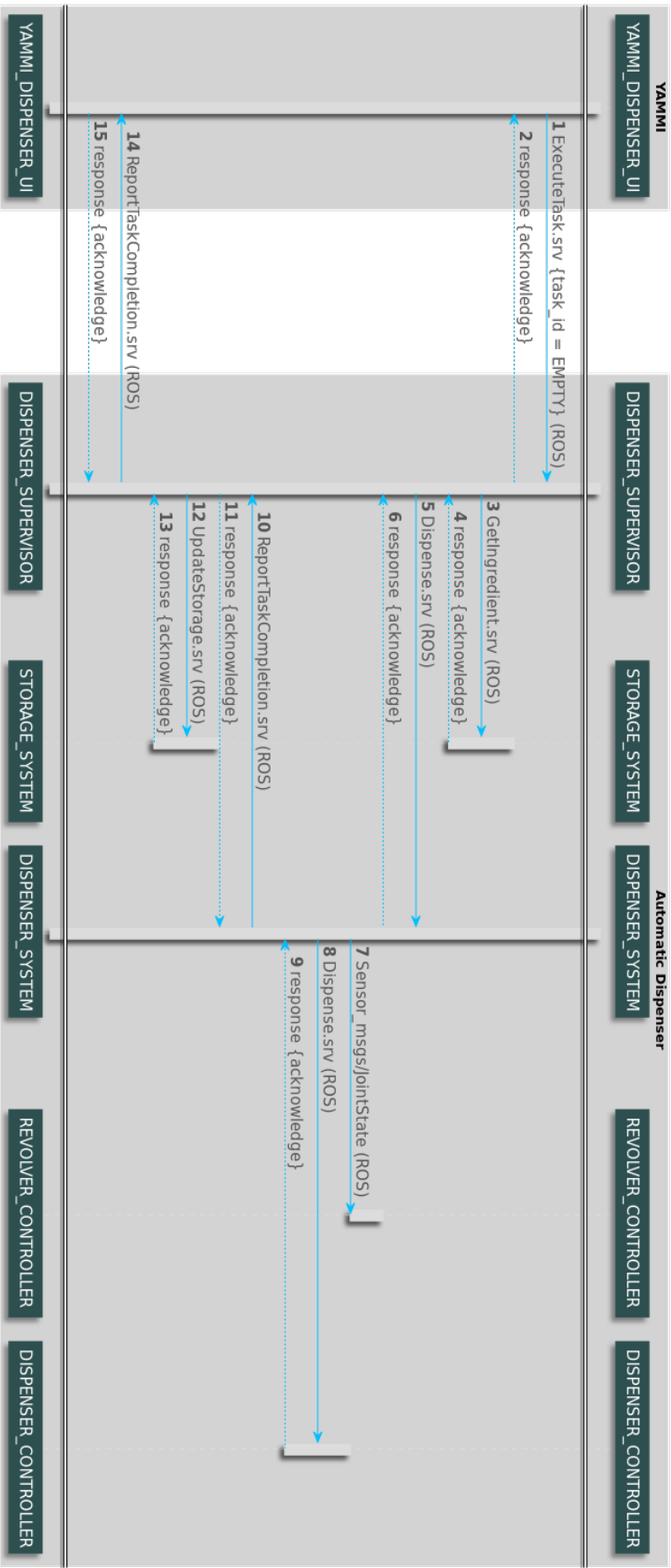


Figure 3.11: Empty sequence chart

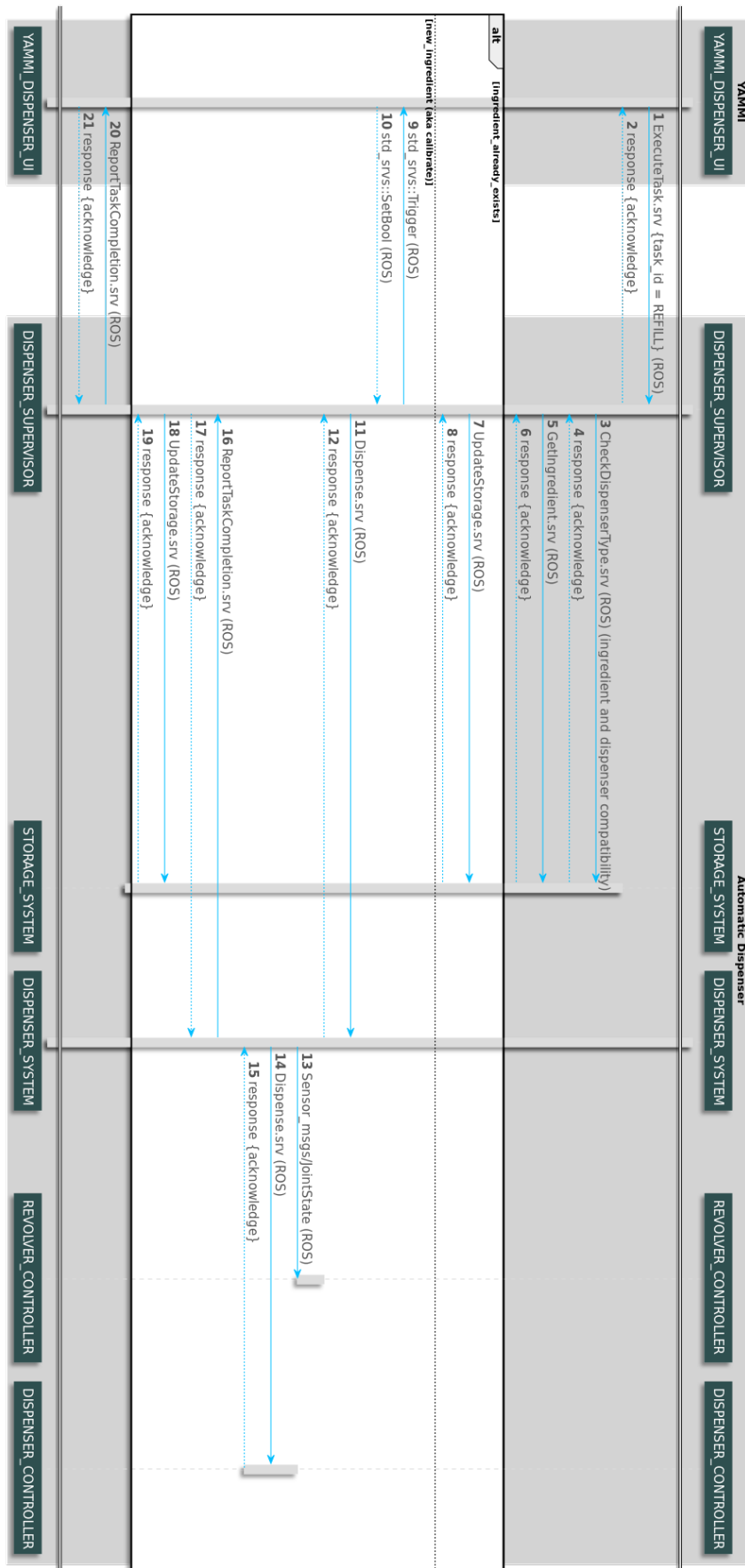


Figure 3.12: Refill sequence chart

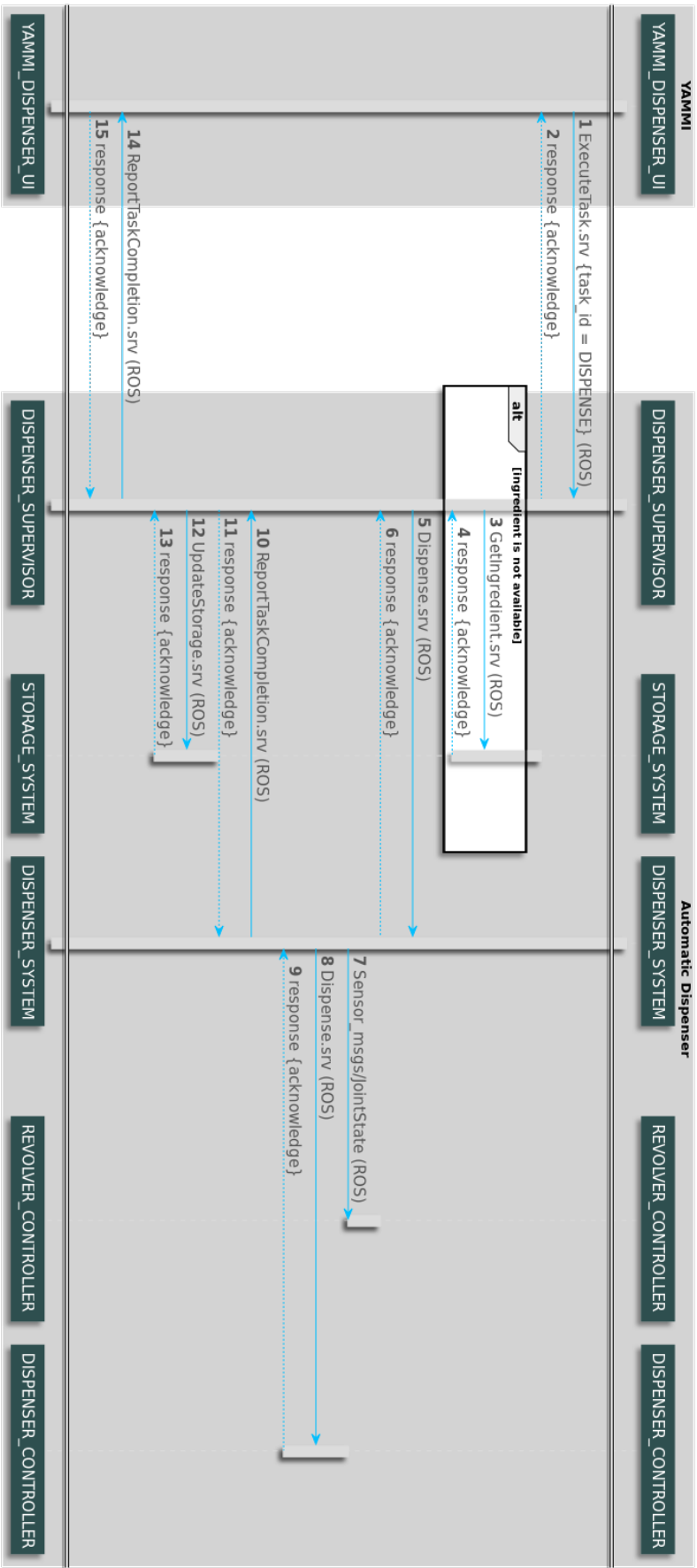


Figure 3.13: Dispense sequence chart

EXPERIMENTAL RESULTS

The architecture and physical structure defined in the previous section will now be used for the tests. The velocity controller will be tested only in a simulation environment as it was not possible to implement it in the complete system setup and the whole system will be tested using different setups, like different dispensers structures and types, which results will be compared in the end.

Before starting the tests, it is important to understand which ingredients will be used and why, the dispenser will handle most of the ingredient requisitions, as such it has to be able to hold these ingredients as well as dispense them. As a way of verifying this capability, the tests will cover more than one type of ingredients, being the two major groups liquids and solids. For the liquids group, the only ingredient that will be tested will be olive oil, since it's a liquid that have some viscosity properties, it will represent an harder to deliver liquid and consequently represent the worst case scenario. For the solids group, it is important to know that these ingredients can come in many forms and sizes, as such we can take into account two types of ingredients and one of them can still be divided in two groups, the big ones, like slices of an apple, and the granular ones, such as salt, and even in this group there can be different levels of granularity, as such, the tests will be made with salt and pepper, while the pepper has really small grains, salt has some larger ones. With this three types of ingredients, most of the possible ingredients are represented.

After defining the ingredients that are going to be used, it's needed to decide how will these ingredients be dispensed, first we will look into the liquids group, the liquids to dispense can be stored in a small container that can be placed on the rotative platform or, if there is the need to it, in a larger container outside of the platform to store liquids that are used in bigger quantities, if the container has a size that is not supported by the platform. An example of this difference can be water and olive oil, usually the water is

used in much bigger quantities than olive oil, therefore, needing a much bigger container which is not suited for the rotative platform. This is possible thanks to the way that the liquid will be pushed to the main container, it will be made by using a pump that will be controlled by a stepper motor, so it will be possible to control the velocity and to stop at will, the rubber tube passing through the pump just has to be long enough to reach both containers, since the size of this tube and the container used will affect the test results, both of them will be the same for every test made with this ingredient. The main goal of the test will be to check the amount of time it takes to deliver the right quantity and if it is possible to stop at the right weight with as better precision as possible. This system can be seen in the figures 4.1 and 4.2.



Figure 4.1: Rubber tube in the rotative platform from an outside container

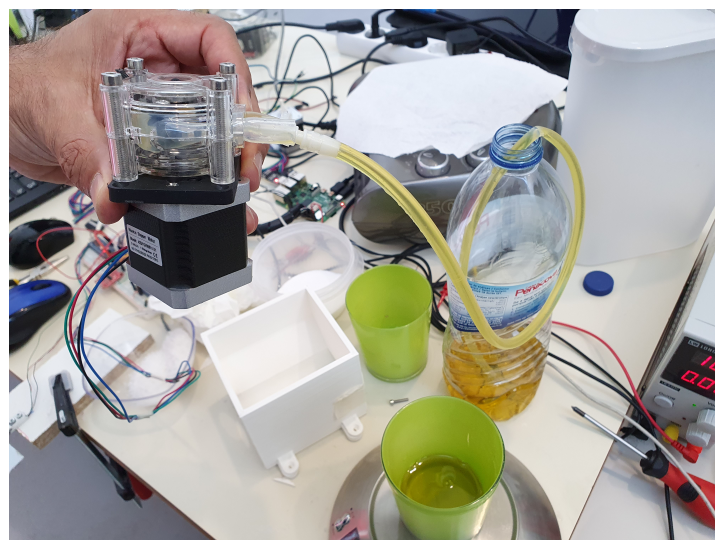


Figure 4.2: Pump connected to outside container

It is possible to dispense different ingredients using the same container as long as it is

part of the same group, for example, the liquids container will be used to dispense olive oil, however, it can also be used to dispense water, since the system to push the liquid from one container to the other will work in the exact same way.

Now that the way liquids will be dispensed is defined, we will define how the solid ingredients dispense will work. As said before, it is possible to split this big group into two smaller ones, big solids and small solids, the big ones, like chunks of meat or slices of fruit, have no effective way to dispense and get an accurate weight, as the size of this slices may vary and have different weights. As such, in a way to overcome this difficulty and still provide an intervention free experience to the user, the dispense of this type of ingredients will be made through a door in the container, letting all of the content fall into the main container, for this to happen, the desired weight of the ingredient must be checked and added to the container before the recipe begins, after this, when the step for this ingredient is received by the controller, it checks the type of the controller and just opens the door in order to dispense the desired ingredient in the inserted quantity.

For the small solids group we will have a different approach, these ingredients are used in a variety of recipes, has such they will have a container in which it is possible to save them for a long time, with this in mind, it is impossible to have the same dispense mechanism. As explained before, this mechanism will consist of a motor connected to an endless screw that will push the ingredient to the hole on the opposite side of the container, the mechanism will be the same for every small ingredient, as the screw is large enough to grab the ingredient regardless of the grain size. This can be seen in the figure 4.3.

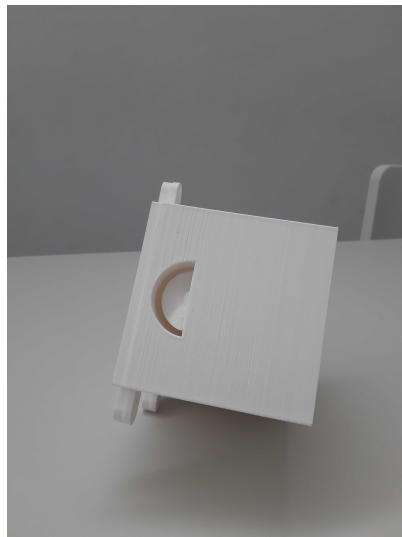


Figure 4.3: Side view of the solids dispenser, showing the exit of the ingredients and the endless screw

After testing the whole system, the velocity controller tests will start, as explained before, the tests will be made in a simulation environment, which will be made in a node-red flow that will replicate the sequence made in the actual system. After selecting the desired ingredient, the flow receives every information that is needed for the normal

operation and begins to dispense it. The velocity at which the ingredient is dispensed will be made using the data received during the whole system tests, in order to get a representation as close as possible to a real implementation. The controller will be tested using different defuzzifiers to decide which one is the best and the results will be compared to the ones obtained in the whole system tests with a constant velocity. In the following subsections we will look into each test with more detail and discuss the obtained results.

4.1 Complete System

After defining the needed concepts, objectives and procedures the tests on the physical components for the whole system started. As explained before, the containers are set on a rotative platform, which positions the right one above the main container, there are no tests results for this as its just a simple rotation, and after testing it several times, the selected container is always on the right place. Before testing the container with the previously described setup, with the stepper motors and drivers, the dispensers were built in a simpler way to test if it would be efficient enough, and to test in a easier way the whole process of dispensing an ingredient. On this initial setup, the dispensers were built with a smaller container and endless screw, and connected to a dc motor. With this setup the velocity is not controlled and the dc motor can be directly connected to the raspberry pi. To test the whole process, a complete recipe is inserted into the main machine controller node-red flow, after this, the whole recipe is processed, and each step is verified and sent to the dispenser supervisor going through the validation processes, that were previously explained. In every test the verification processes were always correct and the selected container was the right one, as such, the evaluation of results will be based on the provided quantity of the ingredient, and on the amount of time it took to deliver that same amount. The tests setup can be seen in the figures [4.4](#), [4.5](#), [4.6](#), [4.7](#).



Figure 4.4: Rotative platform with dc motor dispensers

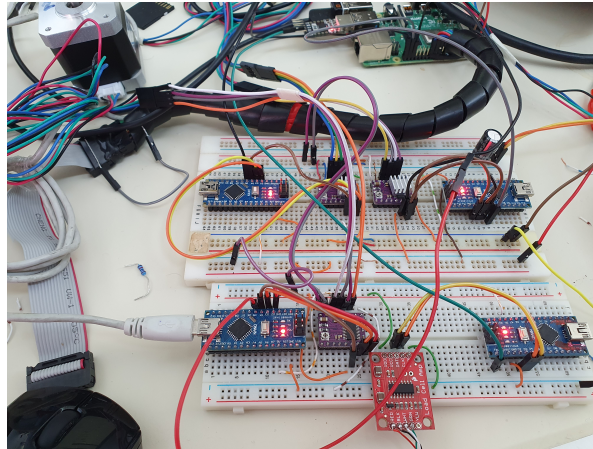


Figure 4.5: Components connections



Figure 4.6: Pump and stepper motor



Figure 4.7: Solid ingredients dispenser with stepper motor

The ingredients chosen for the test were, salt, pepper and olive oil. These ingredients were chosen because not only are the three of them widely used, they represent different kind of dispensing, while the salt and pepper are solid ingredients, olive oil is liquid, and between the salt and the pepper the granularity is different, so it is possible to evaluate the performance of the dispenser with different granularities. As there was no weight control implemented, and the system had to be manually stopped, and to better replicate what would be the average performance of the dispensers, for each time there were three tests for the solid ingredients which results can be seen in the tables 4.1, 4.2 and 4.3.

Time (s)	Average Weight (g)	Standard Deviation
3.6	2.5	0
4.3	2-5	1.41
8.7	8-11	1.12
17.4	22-24	1.83
26	32-37	1.80
34.8	43-48	1.92
43.5	54-56	1.24
52.2	64-70	2.62

Table 4.1: Test results for salt using dc motors

Time (s)	Average Weight (g)	Standard Deviation
2	1.67	0.47
5	3.67	0.47
10	8.33	0.47
15	11.67	0.47
20	16.33	0.47
25	20.67	0.47

Table 4.2: Test results for pepper using dc motors

Time (s)	Weight (g)
30	50
60	100

Table 4.3: Olive Oil test results using dc motors

For this system, as the load cells were not implemented yet, these results were used not only to test the velocity of the motors, but also to set the time needed to reach the any weight using linear regression. The graphs built based on the tests above are shown in figures 4.8 and 4.9.

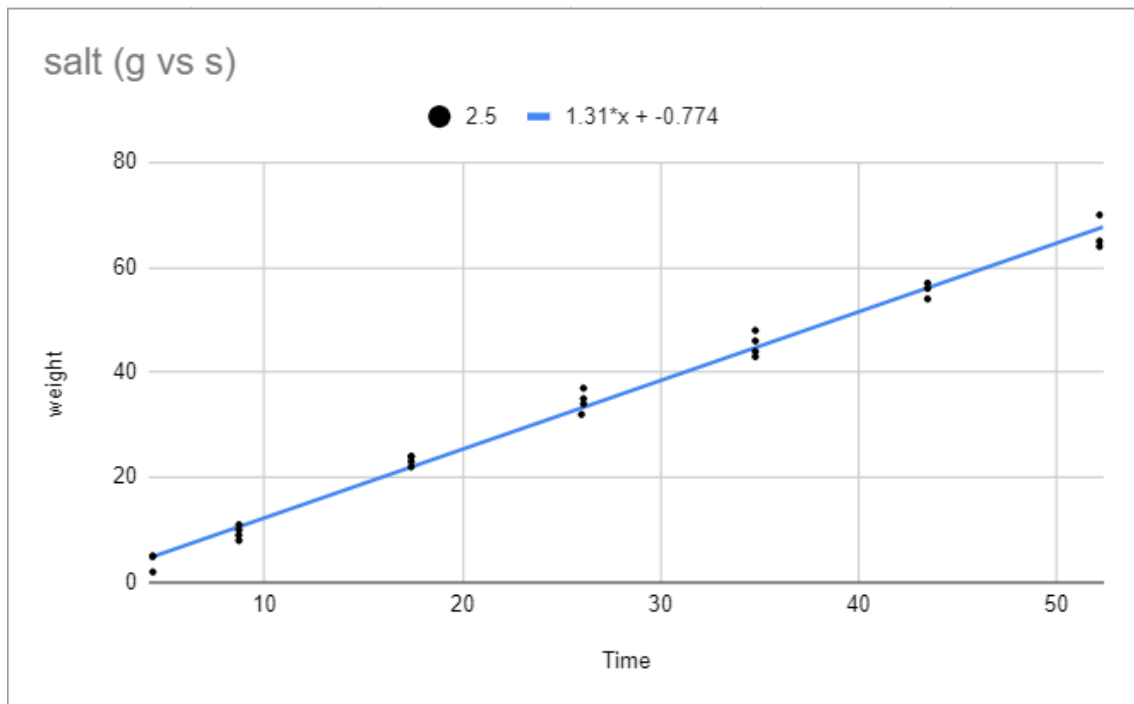


Figure 4.8: Salt drop graph

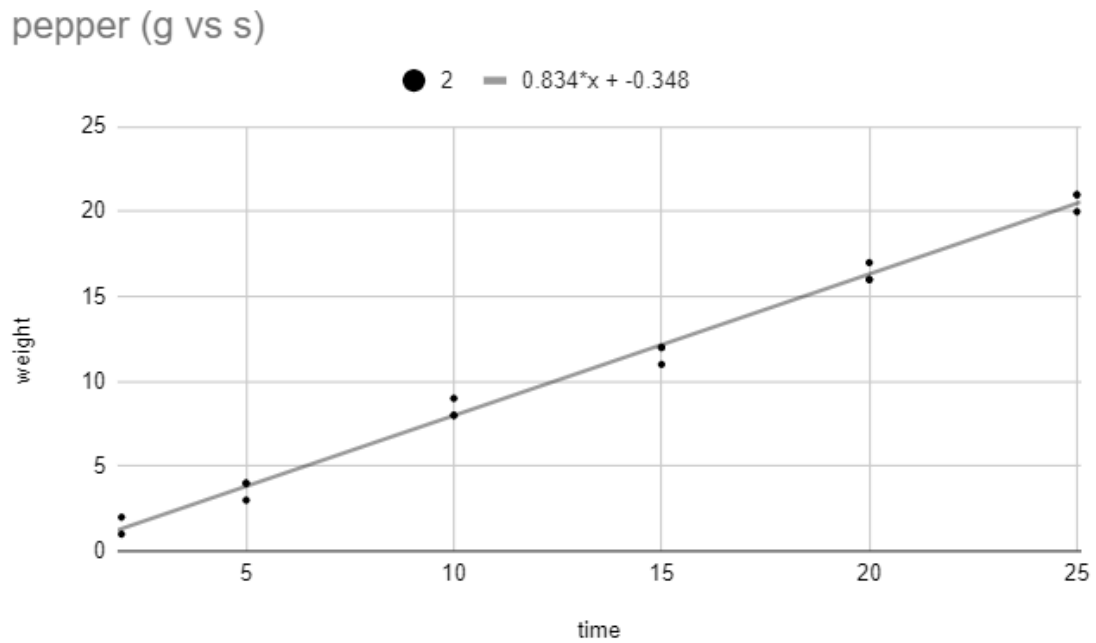


Figure 4.9: Pepper drop graph

After these first set of tests, and realising that the time they take to dispense a given weight is higher than what is desired, the dispenser containers and motors were changed, and the load cells to measure the weight were installed on the container in which the ingredients are to fall. As explained before, each container has a stepper motor attached to its endless screw, which is connected to an arduino nano through a DRV8825 stepper motor driver, to control its rotation speed and direction, and then connected to the raspberry pi. To have a fair comparison, the ingredients used were the same, following the same reasoning. Has the new containers were much bigger than the previous ones, they couldn't be installed on the rotative platform, and were tested separately, since this dispenser is made to be modular, it doesn't affect the final result, even when the dispensers will be set on the platform. Before starting the tests, since the load cells can't be used on the ingredient container and have to be set on the main one, it's needed to compensate for the time it takes for the ingredient to fall, as such, the weight limit is set 2 grams lower then the desired one, this way, after the motor stops the final weight is reached thanks to the material that is already falling into the main container. The test results for this new system are shown in table 4.4.

Expected	Time (s)	Average Weight (g)
2.5	1.1	3.8
5	2.3	5.2
10	2.9	11.5
25	10	27.5
35	14	36.2
48	21	50.1
65	25	65
70	26.2	71

Table 4.4: First test results for Salt with stepper motors

This time only the salt was tested as it was already possible to see a large disparity between the expected weight and the delivered one, as such, instead of adjusting the weight limit to stop again, the dispensers were tilted 30 degrees up, this way when the screw is stopped, there is less chance that the ingredient falls after stopping, resulting in a much better end result. After this change the previous tests restarted, and we can already see that the disparity between what is expected and the real outcome is never higher than 0.3 grams. This is better than what is necessary in a real application, since there is no recipe which measures have this precision. The results obtained are in the tables 4.5, 4.6 and 4.7.

Expected	Time (s)	Average Weight (g)
2.5	1	2.4
5	2	4.8
10	2.9	10
25	9.8	25
35	14.5	34.8
48	20	48
65	25.1	64.8
70	26.5	70.3

Table 4.5: Salt test results for stepper motors after adjustments

Expected	Time (s)	Average Weight (g)
2	1.3	2
10	3.2	9.8
20	9.8	20
40	17	40.2

Table 4.6: Pepper test results for stepper motors after adjustments

Time (s)	Weight (g)
22.5	50
45	100

Table 4.7: Olive Oil test results

With all the tests made it is now possible to evaluate the obtained results. First we will look at the liquid ingredients results, we can see a direct comparison by looking at the average weight dispensed by second, this is possible to see in table 4.8.

Motor Type	Weight/Time (g/s)
DC	1.67
Stepper	2.22

Table 4.8: Olive Oil test comparison

Comparing the amount of olive oil dispensed in one second, within the same environment, being the only difference the motor that was used, with the stepper motor it dispensed almost one more gram per second than with the dc motor, this happens thanks to the higher rotation speed and force that is possible to produce using the stepper motor. There is, however, a downside to the much faster stepper motor, if the motor is rotating too fast, the pump will start to pull bubbles of air with the liquid, resulting in a slower deliver time, as such it is important to keep an high speed rotation while taking this into account and defining a suitable velocity as was done in the test.

Looking now at the results for the solid ingredients, similarly to the previous comparison, a direct comparison between the weight dispensed by second is present in tables 4.9 and 4.10.

Motor Type	Weight/Time (g/s)
DC	1.25
Stepper	2.55

Table 4.9: Salt test comparison

Motor Type	Weight/Time (g/s)
DC	0.88
Stepper	2.28

Table 4.10: Pepper test comparison

It is possible to see that the use of the stepper motors provide a much more efficient way of dispensing the tested ingredients, with the salt, the weight dispensed is more than twice the amount on the DC motors, and in the pepper test the gap is even wider, this is easily explained by the fact that the stepper motors are not only faster than the DC ones, but also stronger, this enables the use of a larger endless screw which catches more of the ingredient to dispense, both of this aspects combined result in observed result.

4.2 Velocity controller

As already said, the velocity controller was tested in a simulation environment, the user can select one of the two possible ingredients, each one has its own weight drop rate, depending on the input velocity, these equations were obtained based on the previously done tests with the dc motors. Before selecting the desired ingredient the user has to provide the desired weight, and then press on the ingredient. Making use of the equations defined on the first tests of the real system, the flow starts the simulation, the output of the velocity will be a percentage of the maximum velocity, this way the evolution of the output can be pictured in any motor as long as we know its maximum velocity, and can predict the outcome when implemented in a real system.

Every 0.4 seconds the flow checks how many of the weight has already dropped, how much is left and the current velocity, with these parameters it makes the necessary computation in order to increase or decrease the velocity of the motor or to stop it. The main goal of this controller is to get the maximum possible velocity at each time while keeping the weight drop as precise as possible, since if it was always kept at the minimum speed it would take too long and it was the opposite, the maximum speed, there was a risk of delivering too much of the ingredient because of the gap between the weight measured on the main container and amount of ingredient that is falling into it.

The results from these tests are represented by the graphs 4.10, 4.11 and 4.12, each graph and defuzzification method will be evaluated.

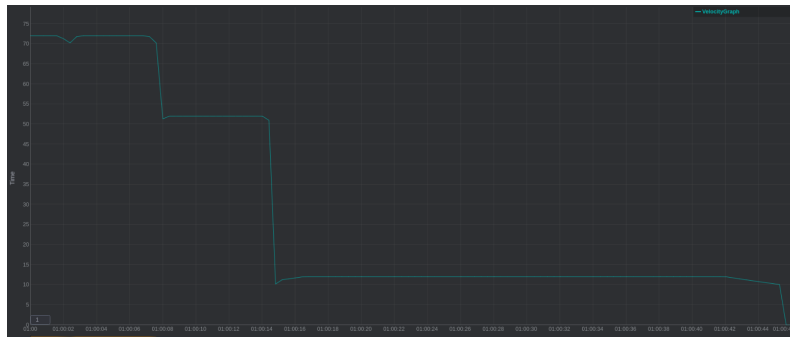


Figure 4.10: Velocity output with smallest of maximum defuzzifier

As we can see in the figure 4.10, the velocity percentage suffers great changes when a certain point is reached, as explained before, with this type of defuzzification small changes in the input can create great changes in the output, and that's possible to see here, as the weight left to drop is lowering, when a certain value is reached the velocity output lowers drastically, creating a step like response.

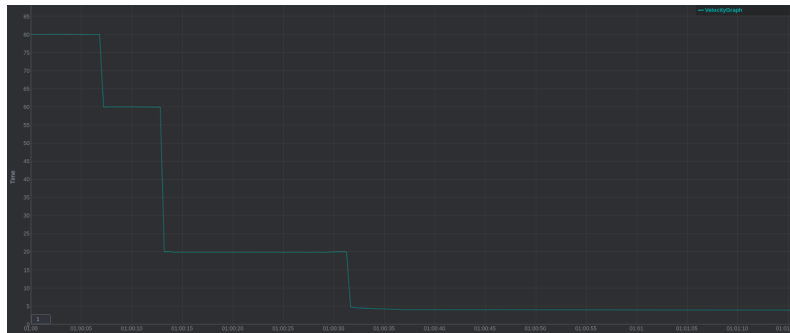


Figure 4.11: Velocity output with mean of maximum defuzzifier

In the figure 4.11, as the method used for defuzzification is the mean of maximum, the response pattern is similar to the previous one, however, with this defuzzifier these changes in the output occur earlier in the process, this happens because as the weight drops, the highest membership value output taken into account is higher than the previous test, in this system, an higher output value equals to a lower velocity.

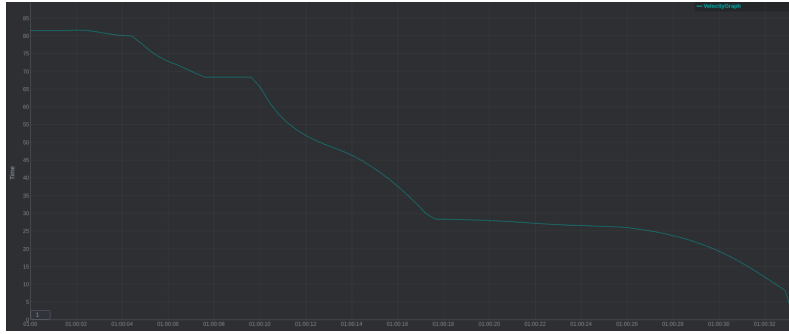


Figure 4.12: Velocity output with center average defuzzifier

Contrary to the previous two results, in the figure 4.12 the velocity output is much more linear, as the method for defuzzification is the center average defuzzifier when a small change occurs in the input it doesn't create a drastic change in the output, instead it slowly changes, even if the input suffers a big change, the change in the output will not directly scale with it, instead it will be smaller, turning the output in a more linear variation over time. When taking all of the results and discussion into account the most desirable solution would be the center average defuzzification method, since the three methods take a similar amount of time while this method put less strain on the components, since there are smoother transactions in speed output throughout the process.

CONCLUSIONS

In the beginning of this document, the goal of this dissertation was defined, this goal was to develop an automatic dispenser that would be attached to its main machine in order to dispense the required ingredients based on its recipe. Although a final fully functional system with all the capabilities was not possible to be developed, a functional prototype could be built, this prototype had most of the desired main functionalities like ingredient storage management, some of the ingredient types dispensing and a communication with a simulated main machine.

The functions that were not possible to implement were also covered in this document, either by explaining or simulating them or by developing them without the implementation on the full setup. The velocity controller, while it could not be implemented, it was still developed and implemented on a simulation environment, and the dispensers with the stepper motors and load cells, were built and tested in a real environment but were not tested in the rotating platform. However, the rotating platform is a very reliable system and so no problem should arise in that integration process.

Even with these drawbacks, the main goal was achieved, while this dispenser is not a final version, it is already a functional one, and the next steps could already be defined, some of the future capabilities could also be tested and even without the full implementation it was possible to evaluate these new solutions and, with the obtained results, make some decisions on what will be the best to implement when developing this dispenser further.

5.1 Future Work

In the course of this dissertation some of the possible solutions were explored and implemented, however, it was only possible to build one physical working model. As the dispenser will be modular it was possible to test it without the main machine integration without compromising the final result, since when the main machine is included in the setup, only the communication between both must be changed. When testing the liquid ingredients, the container used was out of the platform and too big to be a reasonable solution, also, there was no physical model for the big solid ingredients. As the hardware used for the tests was not the optimal one, it was impossible for the velocity controller to be implemented in the the real environment tests, it was also not possible to test every controller configuration in the simulation environment. This also affected the number of ingredients tested, since there were only three of them, and the load cells used needed to be installed on the main container thanks to the way they work, also, these load cells have a maximum weight capability much lower than what is desired for the final solution.

If we take a look at the obstacles described above, it is possible to see improvement opportunities, first of all, as of now the connection method between the main machine and the dispenser module is serial, when the main machine will be assembled, it will be possible to use a more efficient way to communicate, also the way the connection is structured, like the data format, may change and become more efficient. This data structure new configuration may also imply a software new structure, so it is important to also keep the software implementation as modular as possible as to keep this improvements possible in a single module if only that is needed, instead of changing the whole system.

For the dispenser module physical model, some improvements may also be explored, the new containers for the small solid ingredients may be adapted to the rotative platform and the liquids big container should be designed so that it could be attached to the module, so when the module would be assembled, it would be only one complete piece. The rotative platform may also be changed when it will be tested with the main machine, the idea is to mount it on it, so it is important to have a good fit, when the main machine physical module will be known, the overall system is to be adapted in order to improve its efficiency in dispensing the ingredients. Also, the load cells should be changed for new weight measure devices as the weight they need to hold would be much bigger than the one used during the test phase, this new weight measure component should also be evaluated for the need to place a device on each individual dispenser or in the main one, since this may imply a slower connection, a more expensive approach or much harder implementation.

For the velocity controller to be implemented in the full setup, the hardware needs to be improved, a research on new and better hardware should be conducted, however, this will imply a more expensive solution, so the best would be to evaluate different solutions and agree on the more balanced one. When this new hardware is implemented, a new controller should be explored and not only that, but experiment with the different possible

configurations for each controller, since one important feature of this module should be how fast it can dispense the ingredient, since there are recipe where this could be important, the velocity controller would optimise this field of work while keeping the module precise. It is important that not only after these changes but also before them, to serve as performance comparison, more ingredients should be tested in the dispensers, these new ingredients should cover a wider range of peculiarities, such as humidity, different granularities, cluster formation and more.

BIBLIOGRAPHY

- [1] H., R., O., J., I., I. Ide, S. Satoh, S., S., T., and H. “Cooking navi: Assistant for daily cooking in kitchen.” In: Jan. 2005. DOI: [10.1145/1101149.1101228](https://doi.org/10.1145/1101149.1101228).
- [2] A. Amrutha, R. Lekha, and A. Sreedevi. “Automatic soil nutrient detection and fertilizer dispensary system.” In: *Proceedings of 2016 International Conference on Robotics: Current Trends and Future Challenges (RCTFC)* (2017), pp. 1–5.
- [3] *Bimby*. <https://bimby.vorwerk.pt/bimby/>. [Online; accessed 19-July-2019].
- [4] *Burger-flipping robot takes four-day break immediately after landing new job*. <https://www.theverge.com/2018/3/8/17095730/robot-burger-flipping-fast-food-calibur-miso-robotics-flippy>. [Online; accessed 19-July-2019].
- [5] *BWorld Robot Control Software*. <http://www.kenwoodworld.com/pt-pt>. [Online; accessed 19-July-2019].
- [6] *Chef EXPRESS Pingo Doce*. <http://www.chefexpress.pt/>. [Online; accessed 19-July-2019].
- [7] M. Colledanchise and P. Ögren. “Behavior trees in robotics and AI: An introduction.” In: (2017). DOI: [10.1201/9780429489105](https://doi.org/10.1201/9780429489105). arXiv: [1709.00084](https://arxiv.org/abs/1709.00084).
- [8] *Cooki: a Desktop Robotic Chef That Does Everything*. <https://spectrum.ieee.org/autamaton/robotics/home-robots/cooki-a-desktop-robotic-chef-that-does-everything>. [Online; accessed 19-July-2019].
- [9] *Downtown Boston’s Robotic Restaurant Gets \$21 Million to Fund Expansion*. <https://boston.eater.com/2018/9/7/17832612/spyce-robotic-restaurant-21-million-dollar-series-a-funding>. [Online; accessed 19-July-2019].
- [10] *Flippy | Miso Robotics*. <https://misorobotics.com/flippy/>. [Online; accessed 19-July-2019].
- [11] *Moley – The world’s first robotic kitchen*. <http://www.moley.com>. [Online; accessed 19-July-2019].
- [12] Y. Nakauchi, T. Fukuda, K. Noguchi, and T. Matsubara. “Intelligent kitchen: Cooking support by LCD and mobile robot with IC-labeled objects.” In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (2005), pp. 2464–2469.

- [13] *Node RED Programming Guide Programming the IoT*. [Online; accessed 20-February-2020]. URL: <http://noderedguide.com/>.
- [14] J. M. O’Kane. *A gentle introduction to ROS*. 2.1.3. 2016, p. 155. ISBN: 978-1492143239.
- [15] *OneCook: the Robotic Private Chef to Free Your Cooking Time by Team TNL — Kickstarter*. <https://www.kickstarter.com/projects/tech-no-logic/onecook-the-robotic-private-chef-to-free-your-cook>. [Online; accessed 19-July-2019].
- [16] *Prometheus Kitchen*. <http://www.prometheuskitchen.com/>. [Online; accessed 19-July-2019].
- [17] *Robot de cocina “Ladymaxx Gourmet” Instrucciones de uso*. https://www.singer.ag/fileadmin/ladymaxx/Ladymaxx_UserManual_ES_PT.pdf. [Online; accessed 19-July-2019].
- [18] *Robotic Kitchen - High Tech Cooking Robot Chef*. <https://www.indiegogo.com/projects/robotic-kitchen-high-tech-cooking-robot-chef>. [Online; accessed 19-July-2019].
- [19] R. B. Rusu, B. Gerkey, and M. Beetz. “Robots in the kitchen: Exploiting ubiquitous sensing and actuation.” In: *Robotics and Autonomous Systems* 56 (10 2005), pp. 844–856.
- [20] A. Sato, K. Watanabe, and J. Rekimoto. “MimiCook: A cooking assistant system with situated guidance.” In: Feb. 2014, pp. 121–124. DOI: [10.1145/2540930.2540952](https://doi.org/10.1145/2540930.2540952).
- [21] *Sereneti | Sereneti*. <http://sereneti.com/>. [Online; accessed 19-July-2019].
- [22] *Spyce – Culinary excellence elevated by technology*. <https://www.spyce.com>. [Online; accessed 19-July-2019].
- [23] L.-X. Wang. *A Course in Fuzzy Systems and Control*. 1997. ISBN: 9780135408827.
- [24] *Who Are Moley Robotics and Why We Will Change Your Future*. <https://medium.com/foodofthefuture/who-are-moley-robotics-and-why-we-will-change-your-future-1ef7bf588fc0>. [Online; accessed 19-July-2019].
- [25] *Yammi*. <https://www.yammi.pt>. [Online; accessed 19-July-2019].