Universiteit van Amsterdam

Cookin': Interactive Cooking Assistant

By:
Houda Alberts, 10740287
Urja Khurana, 10739947
Melissa Tjhia, 10761071
Richard Olij, 10833730

Coordinator:
Arnoud Visser
Sander van Splunter

Curriculum details
Media Understanding
6 EC

April 2, 2017

Abstract

This report focuses on an interactive system that helps people during cooking by guiding them through the steps of a recipe which is obtained with the Spoonacular API. This involves both text and speech instructions that the system gives to the user, while the user can interact with it by using their voice and poses. In literature, these systems have been explored before, but only certain parts were implemented or it only gave limited feedback during cooking. This system guides a user completely through the process by reading out the equipment, ingredients and steps. Each multimedia component, speech and visuals, will be explored separately and then each component will be connected to the actual system by a user interface. This results in a system that works with textual input at the start, and voice commands and poses during the process where machine learning is used for the recognition of poses and the speech is made more efficient by using more words to recognize commands. Speech recognition is done with the Google Speech Recognition API and Kinect is used for the detection of the joint coordinates of poses. Training is done on a dataset which contains the arm coordinates and that can be extended. For classifying, the k-nearest neighbor algorithm was used since it gives the highest accuracy (99.76%) and is also relatively fast. The value for k is chosen to be 8, based on the amount of poses and a noise class. In the future, other multimedia components can be added to the infrastructure, since it is easily adjustable.

Contents

1 Introduction

Nowadays cooking according to recipes is done with the support of electronic devices, such as smartphones or tablets, because it is easy to search for recipes online. While cooking, the device tends to get dirty easily since it is touched when going through the steps of the recipe and the screen tends to turn off between two steps. The user might miss some steps while reading since they are focused on cooking, which is not desirable.

To make cooking an easier process, the goal was to create a system that recognizes which chronological steps of a recipe are yet to be done through the process. In Figure 1 it can be seen how the system will be used and how it goes through the steps.

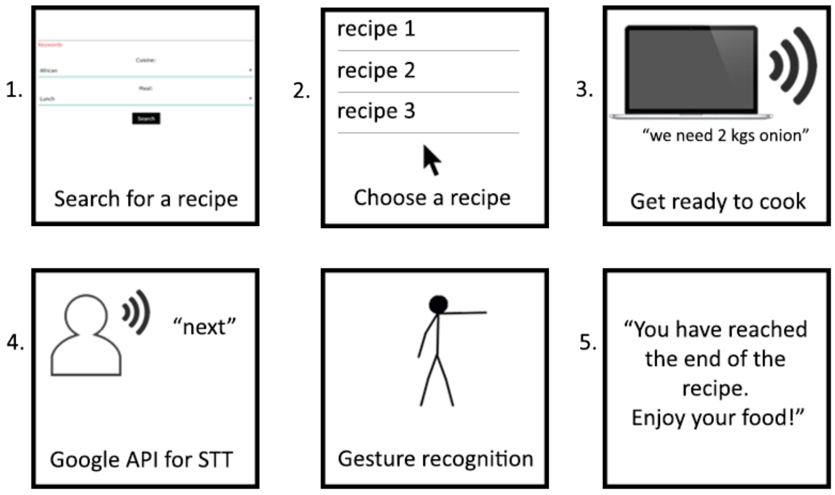

Figure 1: General flow of the system.

The system will read steps out loud in the order that they appear. The user indicates whether they want to go further, back, stop the system or repeat the step and these actions but can only be used when the system allows it, depending on which step the user is at in that moment. It is also possible to ask for the ingredients or equipment during the cooking steps. In these cases, the whole list of ingredients or equipment is read out, so that the program can stay at the current cooking step. As a proof of concept, the first two media used by the system are speech and visuals to detect certain words and poses respectively which are translated in the mentioned actions. For the visuals, the Kinect tracks the coordinates of poses and Python uses these coordinates to classify them into the previously mentioned actions that are used to communicate with the system. In Appendix C, the precise poses can be seen with their corresponding action. The latter medium, which is speech, has the same functionalities as the visual medium but it uses speech instead of movements.

2 Related Work

Some earlier work has been done in which systems were created that also use media to help users cook. For instance, the research of Aoyama *et al.* (2009) where they managed to detect the skill of a cook by the hand and head motions. Head motions were detected in three different ways and the hand motions were tracked by the index finger. Both motions were counted per time interval of a few seconds. Unlike this project, they did not use movements and speech to navigate between steps, but rather used it to detect the difference between the experience of a novice or an experienced cook. The experience level was learned based on the Maximum Likelihood estimation (Brand (1997)). Based on this information, the system gave more or less advice to the cook during cooking.

In the research of Hamada *et al.* (2005), the user was supported while cooking with the aid of text, video and audio instructions. In the case of this project, the program communicates with the user with the help of text and speech. Testing was done by letting the user cook multiple recipes at the same time. Information filtering was used to compose the Action Units, which together form a dish. Two examples of Action Units are “break eggs” and “fry eggs”. They found a way to organize the Action Units, based on the cooking skill and kitchen environment, in a way that the user could cook efficiently by letting them multitask. The goal of this project is not to improve the efficiency of the cooking orders, but rather to help the user go through the steps. The user communicated to the computer by clicking on a button to indicate that a step was completed. This is in contrast with Cookin', in which multimedia is used for input and the user does not have to touch anything to proceed. The outcome of the research was that the users relied on all of the media that was used.

According to Hashimoto *et al.* (2008), many multimedia systems often distract their users from their cooking process by demanding some kind of input from them. However, Smart Kitchen was different and will let the user do the cooking without being bothered by the system. It only helped when it detected that the user needed aid. It recognized certain steps that could be done in the kitchen, it could recognize food and it could track the food which means they used classification for this. However, they were not combined into one system, so there were basically three different modules: stove, tabletop, and sink. Our system cannot recognize acts such as cooking and stirring. However, it might be something for the future version so it can improve the aid during cooking even more.

3 Theoretical Foundation

The main goal of the approaches found in the section Related work for the three systems was to improve the user's cooking skills. The goal of this project, however, is to help the user get through the steps of the recipe without having to touch electronic devices. The techniques in this project will also differ since pose recognition is used to go through the steps, as well as speech recognition, while the other papers introduced only one medium in which the user could communicate with the system.

To combine a system like this, some background knowledge is necessary. The first step is the user interface. It should be able to get input from the user and use this for the program. Important aspects include the communication between Python and the user interface, where the input is retrieved from. That is where Flask[1] will be used. This creates a user interface that can communicate with the programming language, Python, in which the system is written.

The second step is regarding text-to-speech and speech-to-text. The first part can be achieved by letting the system speak to the user based on the operating system and for speech-to-text it is necessary to obtain a good API[2].

Another part is the use of Kinect to get the arm joints. Kinect is a device which has sensors to detect visuals and sounds[3]. There are existing packages within a Kinect-based programming environment, Processing Environment[4], that can handle this information. In combination with the OpenNI library it is possible to do skeleton tracking[5] of which the coordinates of the arm joints can be extracted. A training set of the coordinates of these arm joints with labels of the corresponding action is needed. By using training data and multiple machine learning algorithms mentioned in Eidenberger (2011) to see which algorithm works the best, the system should be able to recognize the poses. Different types of classification methods are discussed here, rule-based categorization, distance-based categorization, and dynamic categorization. An essential property of machine learning algorithms that is explained in the chapter is the extent of rigidness or overfitting. The training data should consist of coordinates that vary in distance to the Kinect and is collected from different angles to prevent overfitting.

The speech-to-text and pose recognition should be working independently. The system should act depending on which of the two is executed first. The other process should then be ignored or terminated. Therefore, the use of asynchronous processes[6] is essential to make the system work.

4 Actual Work

4.1 Research Question

To present a proper working multimedia system, it is essential to have a clear goal. However, to reach that goal, there are several components (i.e. subgoals) that need to be addressed and finished first. The main essence of this project is to develop a system with various means of interaction with the user to guide them through the recipes while cooking. This bring us to the main research question of this project:
How can a user-friendly multimedia system be designed that guides

[1]`http://flask.pocoo.org/`
[2]`https://github.com/Uberi/speech_recognition/blob/master/examples/microphone_recognition.py`
[3]`https://developer.microsoft.com/en-us/windows/kinect/develop`
[4]`https://processing.org/reference/environment`
[5]`http://openni.ru/openni-sdk`
[6]`https://docs.python.org/2/library/threading.html`

users through recipes effortlessly while cooking?
To be able to give a proper answer to the question and actually develop the system, there are some sub-questions that are fundamental to be answered first. These have been chosen to give a clear picture of all the crucial information that contributes to the program (e.g components like speech and visuals). By knowing how each component works on itself and how it is combined, the whole system can be created and presented in a user-friendly way.

Sub-questions that need to be answered:

- How can the system get information about the wanted recipe?
- How can Kinect detect poses?
- How well can the speech commands be differentiated from each other?
- How can said components be combined?
- How can the system be presented in a user-friendly manner?

4.2 Research Method

To kick-start the project, the main priority was to look for features that would make the recipe tracking system a multimedia system, along with a user-friendly one. Since the main goal of the project is to avoid getting the screen dirty because of one's hands while cooking, the most obvious mean to communicate with the system to go to the desired step would be using speech. However, to add more functionality, the idea to use static gestures with Kinect seemed like a good fit. This way, there is no need for the user to touch the screen, however a person who cannot speak will still be able to use the system with poses.

Another factor that plays a huge role in the satisfaction of the user is the user interface of an application. People tend to dodge cluttered and unprofessional interfaces, or interfaces that would end up confusing them. This is why it was decided that an HTML page will be designed with a rather minimalistic look, inspired from Material Design by Google[7], to give the system a more slick feel.

Since the system heavily depends on recipes, retrieving those is the first step. This is done by using the Spoonacular API[8], where the desired recipe can be chosen by keywords. A call is made to the API in Python with Unirest, which then returns a dictionary out of the JSON format from the API, which can be used in Python. The user can also indicate which cuisine (e.g American, French) or what kind of dish (e.g.: dinner, dessert, breakfast) they want.

For the pose recognition, Kinect is used to detect the user. This choice was made since the Kinect is such a widely used medium for recognition of poses and it was more familiar, but in the future it can be extended to Kinect2, since this can track finer motions[9]. Processing 3.2.4 is used, which is a programming

[7]`https://material.io/guidelines/`
[8]`https://spoonacular.com/food-api`
[9]`http://support.xbox.com/nl-NL/xbox-on-windows/accessories/kinect-for-windows-v2-setup`

environment in which it is possible to program with the Kinect. This program should be downloaded and when the Kinect is connected, it should detect movements using the video stream. To be able to distinguish different gestures with the Kinect, the open source SDK OpenNI library is used. The skeleton tracking and the obtaining of the corresponding joint coordinates was done with the help of the SimpleOpenNI User Test example code[10].

Python has several built-in functions for Text-to-Speech and Speech-to-Text. These functions are used to make sure that the system can speak and listen. There are several ones for Text-to-Speech, so each will be individually tested to see how well it works (e.g eSpeak[11], win32com.client[12]). Speech-To-Text can be based on different API's, like Google[13] and IBM[14]. Multiple of those will be evaluated as well, to see what works best. Additional changes may need to be made if certain functions do not produce the desired result.

Finally, each of the components mentioned above should be all handled as asynchronous processes. Each of them is given a list of actions to which they should respond. If the user performs one of these actions, the process should return this action back to the system. The first action received by the system should then be used, the other processes should be ignored or terminated.

4.3 Expected Results

As a result of this project, a working program is expected. This program should have a clear interface in which the user can search through the recipes by giving textual input. After the user has selected a recipe, the program should give the ingredients and equipment that are needed through speech. The user should then be able to communicate with the program by speech or hand gestures. The system should recognize this as best as possible (e.g approximately 90% accuracy), to make sure that the user does not get irritated by a faulty system. The commands by which the user can communicate include previous, next, again, close, ingredients and equipment. Previous, next, again and close are used to navigate through the recipe. However, not all commands can be used during all the steps (e.g previous should not be able in the first step). When the ingredients or equipment commands are used, the whole list of ingredients or equipment is read out, while the program will stay in the same step in the recipe. The architecture of this system can be found in Figure 6 in Appendix A.

A crucial part of the architecture is the Step object which contains the information that is needed for a step in the recipe. This information is retrieved from the Spoonacular API. Only the ingredients, equipment and the instructions of the step are needed, so these need to be filtered from the all the information that is returned from the API. It also contains the possible actions for that

[10]`https://github.com/xohm/SimpleOpenNI/blob/master/dist/all/SimpleOpenNI/examples/OpenNI/User/User.pde`

[11]`http://espeak.sourceforge.net/`

[12]`http://stackoverflow.com/questions/12031231/usage-of-win32com-client-text-%2Dto-speech-speech-recognition-in-python`

[13]`https://cloud.google.com/speech/`

[14]`https://www.ibm.com/watson/developercloud/speech-to-text.html`

specific step. The Step objects are used throughout the program to be able to navigate through the recipe steps. The other aspects of the architecture in the previously mentioned figure speak for themselves.

4.4 Obtained Results & Evaluation

Based on previous experience (Albert Mwanjesa (10797874) (2016)), it was known that the Spoonacular API gives relevant output that can be used for this project. This API was used to get the recipe based on title, ingredients, kitchen and course. It produces useful recipes that the system is able to use. The API returned a dictionary which contained a lot of information, such as the quantity of ingredients, the names, images, and isle of the ingredients and equipment and further information about the recipe, such as price per serving and popularity. However, for this system only the quantity of the ingredients, the names of the ingredients and equipment and the recipe steps were needed and this was the information that was stored in the Step object.

The program can also use Text-to-Speech to give instructions to the user. The Operating Systems Ubuntu, Windows and iOS have different ways of handling speech. Just using one method resulted in errors on certain operating systems, which would not be user-friendly. Therefore, the system checks the OS and based on that it correctly chooses the code that has to be used for the Text-to-Speech communication.

The program is also able to detect speech and convert this into text which can be used to navigate through the program. The Google API[2] for Speech-to-Text was used because it gave the best results without requiring a license. Even though this API works quite well, it was found that the commands were not always recognized as the names of the commands had to match exactly. For example, the word “equipment” was often heard as “equipments”, so the system would not recognize it. Therefore, each command was tested 60 times, and the words for which the commands were misunderstood were tracked. The words that were heard most often were added to the words for which the action is performed. So now, for example, the system will also perform an action when the word “equipment” is recognized as “equipments”. All the recognized words can be found in Figure 2 per action.

Kinect is used to detect users. It can track the user’s body, as can be seen in Appendix B, but for this project only the poses of the arms are needed. Therefore, information filtering was applied and only the coordinates of the joints of the hands, elbows and shoulders are used. Before the system starts with the recipe, the user can collect training data. When the Kinect is used in new environments, it is actually encouraged to do so. Also, when there only has been trained on relatively short people, it works really well for short people and not for taller people, therefore it is essential to train on the people who use the system to obtain the best general recognition. A small script was written in Processing to collect training data in a simple way, in which the skeleton

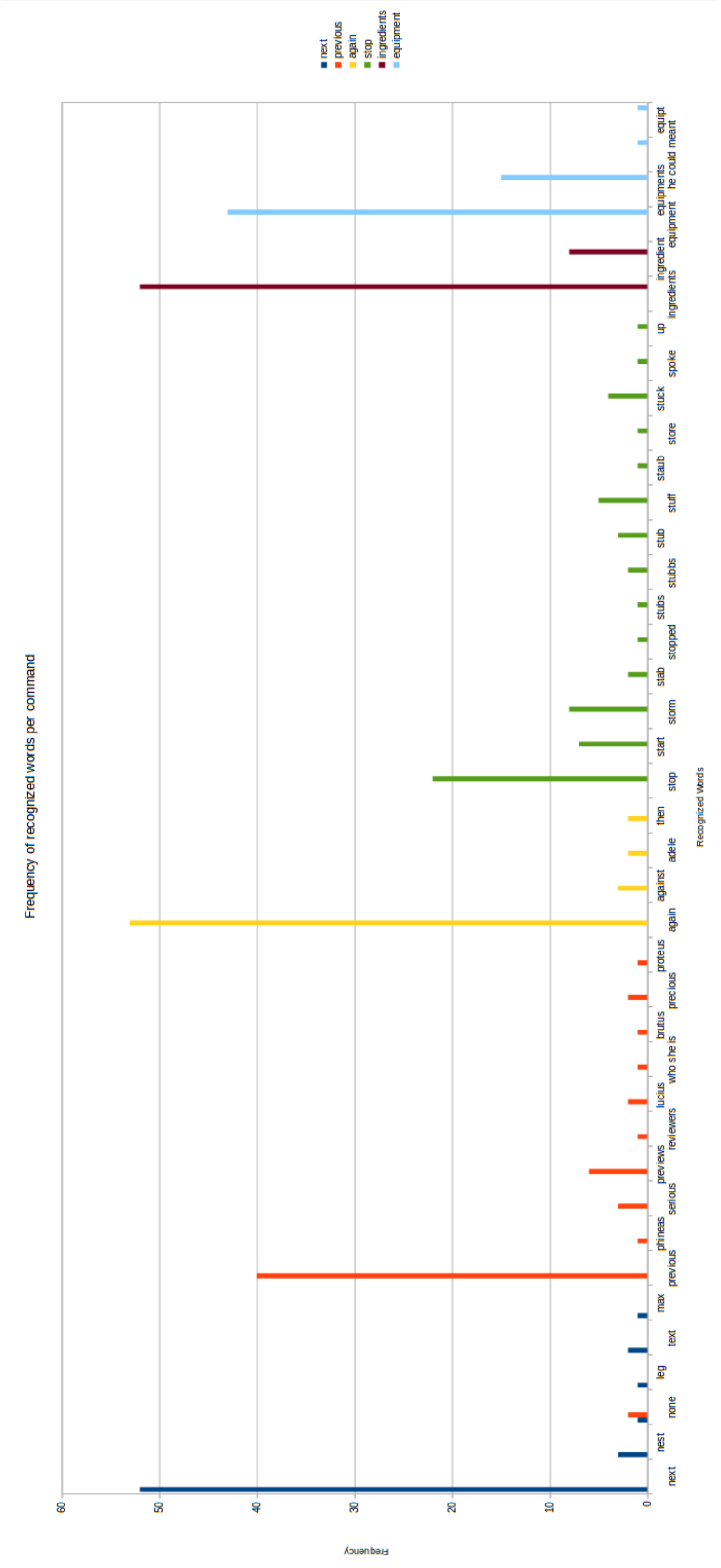

Figure 2: Frequency of recognized words per possible action

Tracking is done with code from SimpleOpenNI User Test example code[10]. The user chooses for which action the training data should be created. Then, the user takes the position with little changes, for example coming closer to Kinect and going further or posing from different angles, that corresponds with this action. These little changes are done so that the algorithm will not overfit the data on very specific poses. Every second the coordinates of the user's hands, elbows and shoulders are written 30 times per second to a text file which will contain the data for that action. These steps are done for each action, as well as the creation of noise. As can be seen in Figure 3, with 10.000 training samples, some algorithms have accuracies of over 97%, where 80% is training data and 20 % test data. So with 30 fps (frames per second) $\times$ 8 poses (including noise) $\times$ 60 seconds, it will generate 14400 training examples, where the accuracies of k-Nearest Neighbor and RBF SVM are 97.84% and 98.47%, respectively. So it takes less than one minute of posing per action to generate enough training data. Because it is possible to generate new data easily and quickly, the poses of the system are always adjustable.

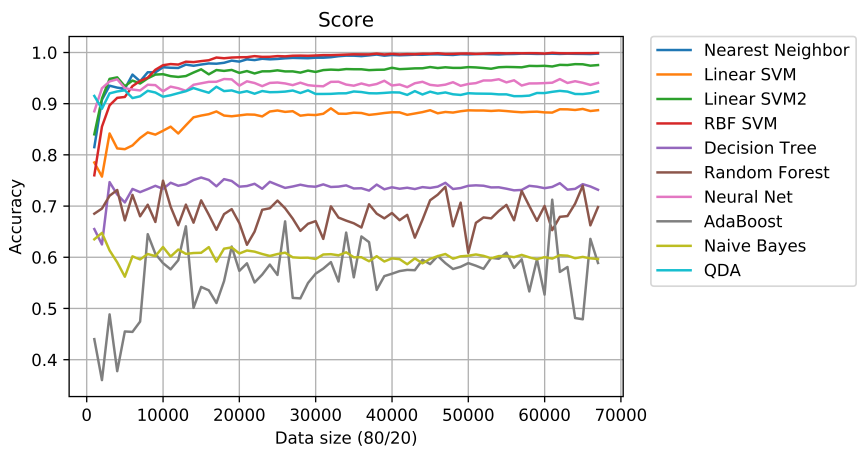

Figure 3: The training algorithms start to get steady accuracies with 10000 training examples. Nearest Neighbor and RBF SVM have the highest accuracy of respectively 97.1% and 97.5%. At the maximum of 67000 training examples they are still both the most accurate algorithms with accuracies of 99.76% and 99.86%.

As can be seen in Figure 3, the algorithms k-nearest neighbor and RBF SVM have the highest accuracy of 99.76% and 99.86% at the maximum amount of 67000 training examples. As mentioned above, even at 10000 training examples they scored the highest on accuracy. However, Figure 4 shows a clear exponential growth in training time for the RBF SVM, which is therefore not a good candidate for this system since the accuracies are similar to the significantly faster Nearest Neighbor. The Nearest Neighbor was one of the fastest algorithms as seen in Figure 5 and therefore, it was used to classify the live poses of the user. In Processing, the user coordinates are written to a text file. Python reads these coordinates and empties the text file, so that it will not read the same coordinates in case the Kinect has not yet written new coordinates. The

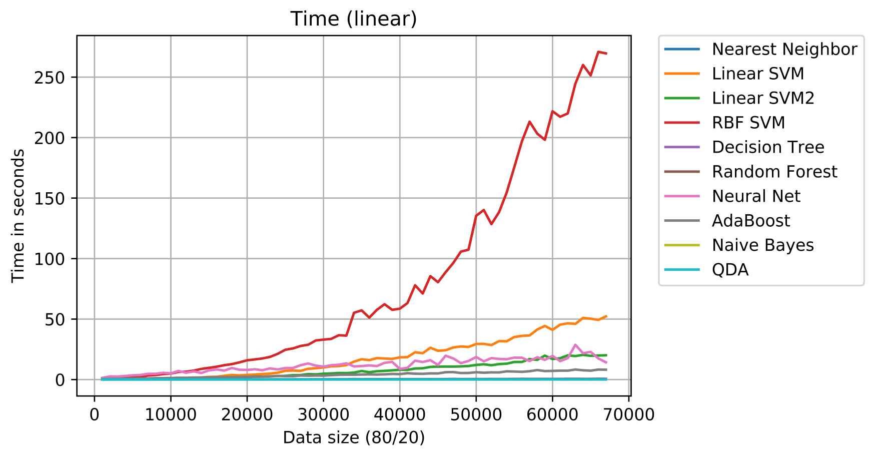

Figure 4: RBF SVM is a significantly slower trainings algorithm than all other algorithms and the training time grows exponentially. Training with the maximum of 67000 training examples took 269.6 seconds while the 5 fastest algorithms Decision Tree, Random Forest, Nearest Neighbor, QDA and Naive Bayes (that appear as a single line) took respectively 0.57, 0.32, 0.1, 0.07 and 0.03 seconds.

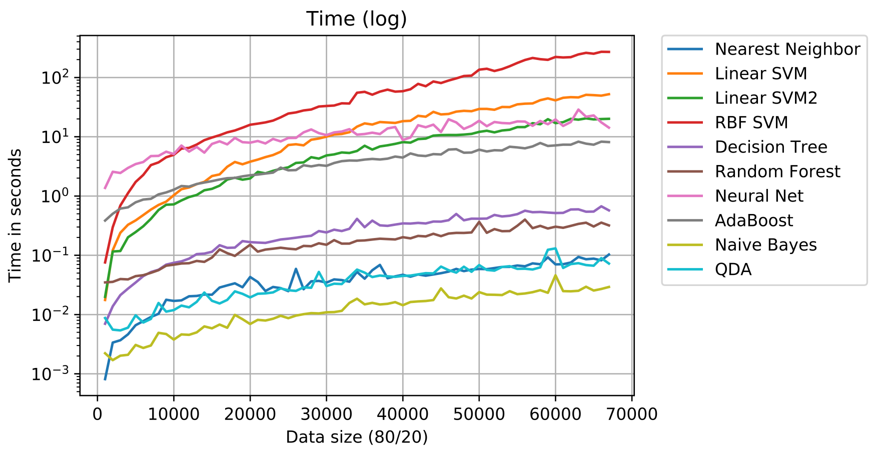

Figure 5: RBF SVM, Linear SVM (linear) and Linear SVM2 (rbf) have the fastest exponential grow, whereas Naive Bayes, QDA, Nearest Neighbor, Random Forest and Decision Tree are significantly faster.

Coordinates are then used as input for the algorithm 30 times per second, which returns the corresponding class. When the same class has been chosen for at least 95% of the 60 last classifications, that class will be chosen as the action that must be performed. This has been done to make sure that the correct class is chosen and this whole process is done in a blink of an eye.

An explanation for the high performance of k-nearest neighbor is that this classification problem in itself is a rather simple one. While other algorithms (e.g. Adaboost or Random Forest Trees) try to fit a complex model to the classification problem, k-nearest neighbor is a distance-based classification algorithm which is heavily dependent on the large and diverse training data which is provided. Because the training data collected by the system is done in such a systematic way, k-nearest neighbor is able to classify new poses quickly and properly by matching them roughly with the training data. Besides that, because of the diversity of the training set it is also able to properly deal with poses from a different distance or angle. Noise is also dealt with by having a class with noise, so that it will not choose a wrong class, that is an actual action, with a relatively small distance compared to others.

While the system works well, there are still some aspects of the system that show malfunctions when evaluated in depth. The first is regarding the voice of the laptop. During the process of cooking, the user gets information by the program by the voice of the laptop. While the voice on a Mac sounds pretty smooth, the one for Ubuntu sounds rather static and unpleasant. People tend to get annoyed by such voices, which means that people with other operating systems than Mac may not experience a pleasant interaction.

Another big aspect of this evaluation is that the system is only tested under a toy model. Real life situations during cooking are not considered, while they can have a great influence on how the system works. For example, an exhaust hood produces a lot of sounds during cooking, which introduces more noise in the voice communication with the system that would not work as well as tested in a silent environment.

One other thing that does not work well, is the detection of a user at the start, when more people are present. During the process of cooking, the Kinect can distinguish the main user from other people that may come in the frame, but at the start, it is quite hard to detect the correct main user. This is also quite a difficult problem to solve, since there is not a specific position where the main user should start.

5 Conclusion

In conclusion, a proper working system was made that guides a user through the steps in a recipe. The first part of this was regarding the information of the recipe. This was obtained by using the Spoonacular API, which returned all the necessary information about the recipe. Then Processing & the Kinect are used to detect poses. OpenNI provides frameworks to translate visuals to joint values. Speech does work and the differences between most of the commands are quite clear. However, some are not still extremely well recognized, but it is sufficient. Afterwards, these components are combined using asynchronous processes that can run the multiple media together. Lastly, the user interface is clear, simple and easy to start. This makes it user-friendly as well. So, the system works, uses multiple media components to guide a user through a recipe, while still thinking of the user (user-friendly), which was the goal and main research question of this project.

6 Discussion

Most problems were related to Kinect. At first, the Kinect was not recognized on the computer when using the Kinect for Windows SDK. This was solved by using the Processing Environment. One of the biggest problems that was encountered was the communication between Python and Processing. The main program and the speech parts were written in Python, while the tracking of the arms was done in Java in Processing. An attempt was made to communicate with Processing with the Processing Environment. However, it failed to get it to work during this project as there was poor documentation, so an alternative was found. The output of Processing with the user's coordinates are written to a text file. This text file is read by Python so that it can be used as input for the learning algorithm. This way direct communication between Python and Processing is not needed anymore and performance of the program has not suffered since the writing and reading is done fast enough.

7 Future Work

Future additions can be made easily because of the structured and documented implementation. Other media components can be added easily, because the system works with asynchronous processes. A functionality that might be added in the future in this project is that it would be possible that the system recognizes which action the user is executing at the moment, like stirring a bowl. The system would then be able to tell the user how long they should be stirring to complete the current step. The next step will then follow automatically. Another possibility for the future is to have the system learn certain speech commands instead of using the Google API. Lastly, the system currently starts from a webpage. An addition can be made that the system can start with speech recognition as well to minimize the use of a computer instead of having to search with textual input. This way, it would also be possible to use the system for illiterate or disabled (e.g. blind) people.

A System Properties

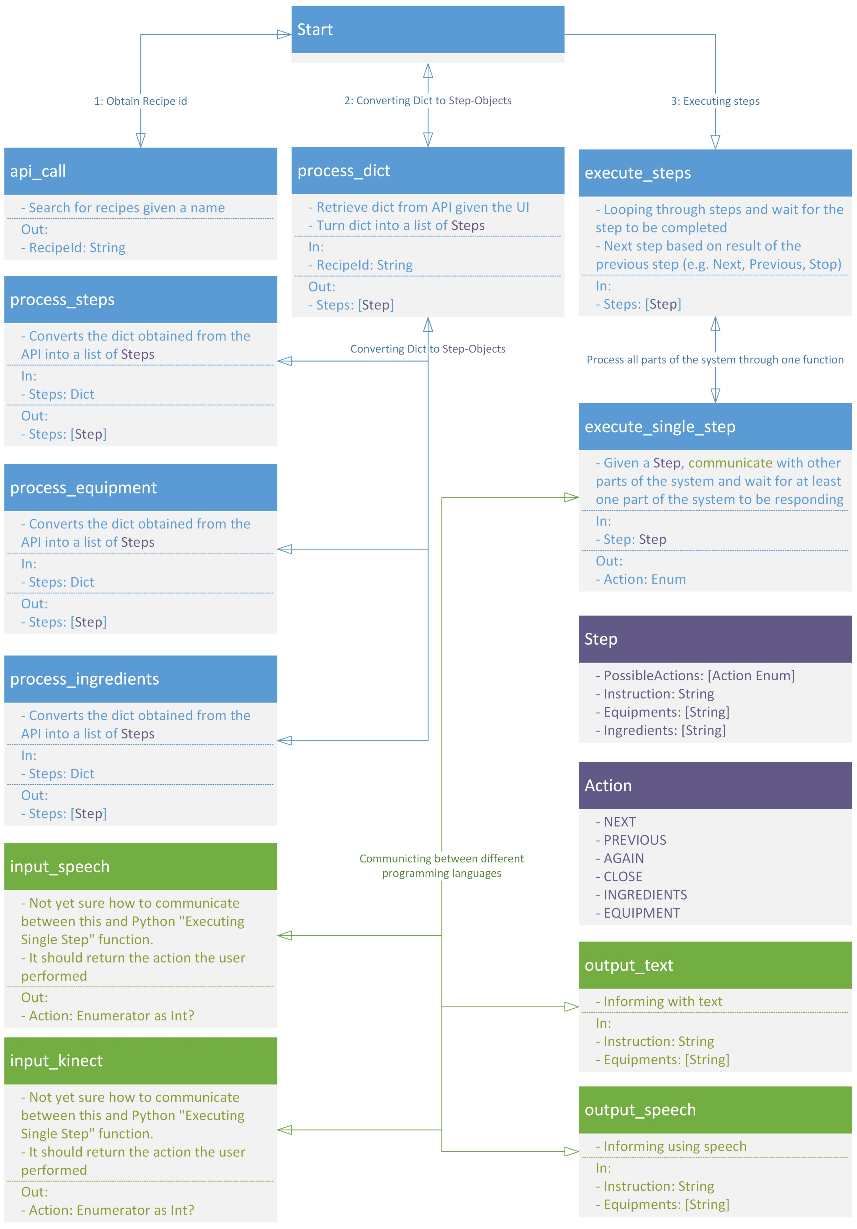
Start
1: Obtain Recipe id
2: Converting Dict to Step-Objects
3: Executing steps
api_call
- Search for recipes given a name
Out:
- RecipeId: String
process_dict
- Retrieve dict from API given the UI
- Turn dict into a list of Steps
In:
- RecipeId: String
Out:
- Steps: [Step]
execute_steps
- Looping through steps and wait for the step to be completed
- Next step based on result of the previous step (e.g. Next, Previous, Stop)
In:
- Steps: [Step]
Converting Dict to Step-Objects
Process all parts of the system through one function
process_steps
- Converts the dict obtained from the API into a list of Steps
In:
- Steps: Dict
Out:
- Steps: [Step]
execute_single_step
- Given a Step, communicate with other parts of the system and wait for at least one part of the system to be responding
In:
- Step: Step
Out:
- Action: Enum
process_equipment
- Converts the dict obtained from the API into a list of Steps
In:
- Steps: Dict
Out:
- Steps: [Step]
Step
- PossibleActions: [Action Enum]
- Instruction: String
- Equipments: [String]
- Ingredients: [String]
process_ingredients
- Converts the dict obtained from the API into a list of Steps
In:
- Steps: Dict
Out:
- Steps: [Step]
Action
- NEXT
- PREVIOUS
- AGAIN
- CLOSE
- INGREDIENTS
- EQUIPMENT
Communicting between different programming languages
input_speech
- Not yet sure how to communicate between this and Python "Executing Single Step" function.
- It should return the action the user performed
Out:
- Action: Enumerator as Int?
output_text
- Informing with text
In:
- Instruction: String
- Equipments: [String]
input_kinect
- Not yet sure how to communicate between this and Python "Executing Single Step" function.
- It should return the action the user performed
Out:
- Action: Enumerator as Int?
output_speech
- Informing using speech
In:
- Instruction: String
- Equipments: [String]

Figure 6: The general overview of the architecture of the system, where blue boxes are the primary processes to make the system function, the green boxes are purely for communicating between the user and the system and the purple box is the object that will be used throughout the system.

B System View

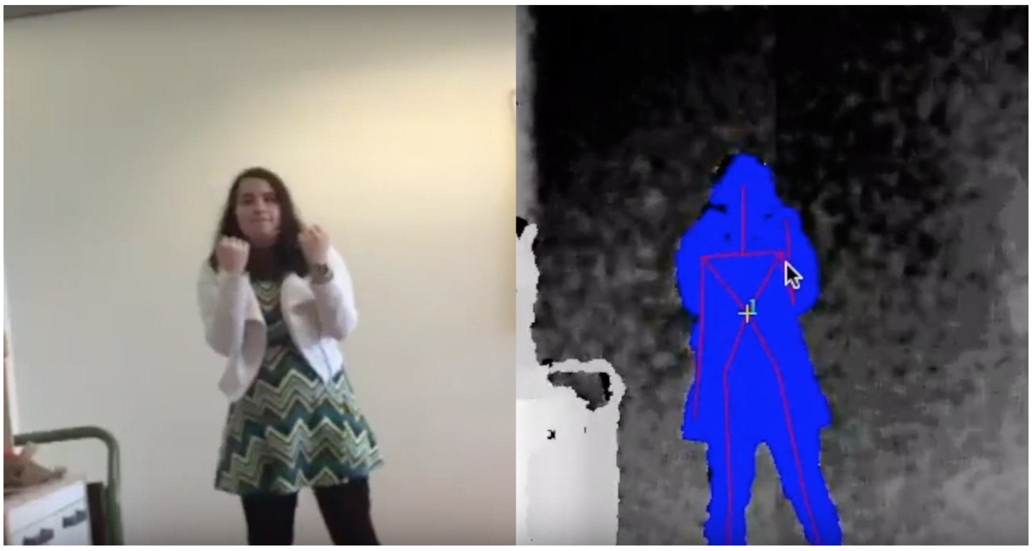

Figure 7: The joints of the user are detected as a skeleton.

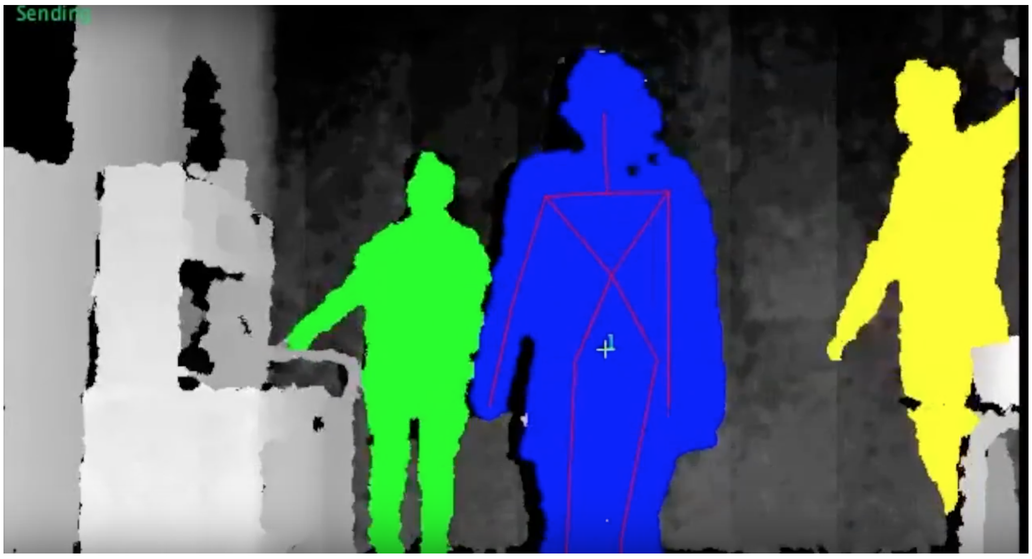

Figure 8: During the steps, other users are recognized but not tracked.

C Poses as Stick figures

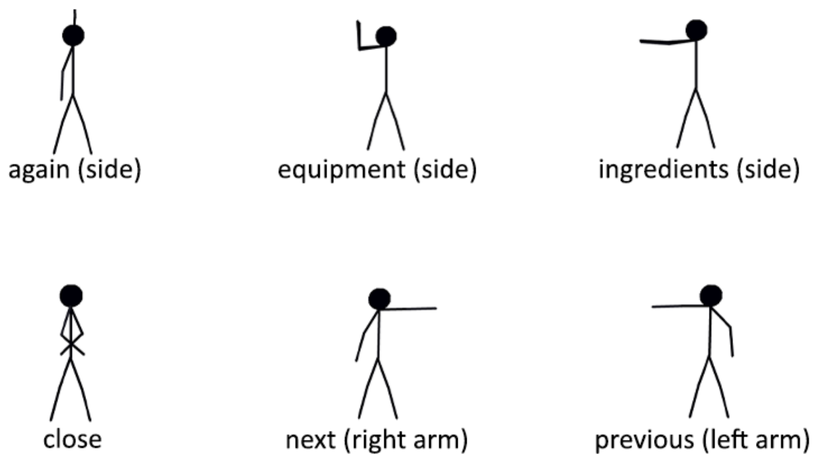

Figure 9: The poses and the corresponding actions. For "again" there are two classes, so that it does not matter whether the left or the right hand is raised.

References

Albert Mwanjesa (10797874), Tobias Garritsen (10779582), Urja Khurana (10739947). 2016. *Fuzzy Foodie Farm: Fuzzy Food Recommendation Using Nutrient Data, For Fundamentals of Fuzzy Logic, Not Published.*

Aoyama, Hideki, Ozeki, Motoyuki, & Nakamura, Yuichi. 2009. Smart cooking support system based on interaction reproducing model. *Pages 39–46 of: Proceedings of the ACM multimedia 2009 workshop on Multimedia for cooking and eating activities.* ACM.

Brand, Matthew. 1997. *Coupled hidden Markov models for modeling interacting processes.*

Eidenberger, Horst. 2011. *Chapter 8: Simple Categorization Methods, Fundamental Media Understanding.*

Hamada, Reiko, Okabe, Jun, Ide, Ichiro, Satoh, Shin'ichi, Sakai, Shuichi, & Tanaka, Hidehiko. 2005. Cooking navi: assistant for daily cooking in kitchen. *Pages 371–374 of: Proceedings of the 13th annual ACM international conference on Multimedia.* ACM.

Hashimoto, Atsushi, Mori, Naoyuki, Funatomi, Takuya, Yamakata, Yoko, Kakusho, Koh, & Minoh, Michihiko. 2008. Smart kitchen: A user centric cooking support system. *Pages 848–854 of: Proceedings of IPMU*, vol. 8.