IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 April 2023, accepted 4 May 2023, date of publication 15 May 2023, date of current version 13 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3276234

RESEARCH ARTICLE

Recognition of Human Chef's Intentions for Incremental Learning of Cookbook by Robotic Salad Chef

GRZEGORZ SOCHACKI, ARSEN ABDULALI, (Member, IEEE), NARGES KHADEM HOSSEINI, (Member, IEEE), AND FUMIYA IIDA, (Senior Member, IEEE)
Bio-Inspired Robotics Laboratory (BIRL), Department of Engineering, University of Cambridge, CB2 1PZ Cambridge, U.K.

Corresponding author: Grzegorz Sochacki (gks33@cam.ac.uk)

This work was supported by Beko plc and Engineering and Physical Sciences Research Council (EPSRC) Agriforwards CDT Project EP/S023917/1.

ABSTRACT Robotic chefs are a promising technology that can bring sizeable health and economic benefits when deployed ubiquitously. This deployment is hindered by the costly process of programming the robots to cook specific dishes while humans learn from observation or freely available videos. In this paper, we propose an algorithm that incrementally adds recipes to the robot's cookbook based on the visual observation of a human chef, enabling the easier and cheaper deployment of robotic chefs. A new recipe is added only if the current observation is substantially different than all recipes in the cookbook, which is decided by computing the similarity between the vectorizations of these two. The algorithm correctly recognizes known recipes in 93% of the demonstrations and successfully learned new recipes when shown, using off-the-shelf neural networks for computer vision. We show that videos and demonstrations are viable sources of data for robotic chef programming when extended to massive publicly available data sources like YouTube.

INDEX TERMS Computer vision, hidden Markov model, learning by demonstration, robotic chef, salad chef.

I. INTRODUCTION

The availability of high-quality meals is an important indicator of the quality of life. The time spent on cooking correlates with health quality [1], but is falling steadily [2]. The most frequently reported reason for lack of cooking was lack of time [3]. Therefore, large benefits can come from the automation of cooking both in homes and hospitality sectors. Numerous experiments were made to explore the feasibility of robotic chefs. The trends vary from simple kitchen helpers like dishwasher packing robot [4], burger flipping robot [5], or sausage frying robot [6] to attempts at building whole robotic restaurant [7], [8].

Implementing a robotic chef is a complicated task, that requires the robot to be competent in many fields of robotics like manipulation, sensing, feedback, decision-making and perception. Recent developments in robotic chef manipulation include peeling lettuce [9], robotic cutting [10], stir-frying [11], tending to food during cooking [12] and in hand assessment of fruit ripeness [13], [14]. New ways of sensing were also implemented, including assessing the state of readiness of a dish [15], using conductance measurement for closed-loop cooking [16], and running various classification tasks with taste and smell sensors eg. classifying wine age [17], detecting mutton adulteration [18], assessing the quality of roasted coffee beans [19], sensing type of milk used for cheese production [20] and judging fish freshness [21]. Robotic chefs are becoming smarter and can learn from the video of human cooking [22], their own attempts at cooking [16], and can even use active sensing to improve their sensory ability [23]. Some experiments showed robotic chefs learning from human feedback on the taste of cooked dishes [24]. Some experiments with transcribing recipes into a set of actions [25] and robots cooking from recipes were done [26].

Computer vision is also frequently used in robotic cooking setups [4], [5], [6], [9]. Some of these experiments

The associate editor coordinating the review of this manuscript and approving it for publication was Rongbo Zhu.

TABLE 1. Ingredient list of salads used in the experiments.

	Broccoli Pieces	Carrot Pieces	Apple Pieces	Banana Pieces	Orange Pieces	Number of Salads Prepared
Recipe 1	0	1	1	0	0	2
Recipe 2	0	3	1	0	0	2
Recipe 3	0	1	2	0	0	2
Recipe 4	0	0	1	1	1	2
Recipe 5	0	0	1	1	3	2
Recipe 6	1	1	0	0	0	2
Recipe 7	3	1	0	0	0	2
Recipe 8	0	0	0	2	2	2

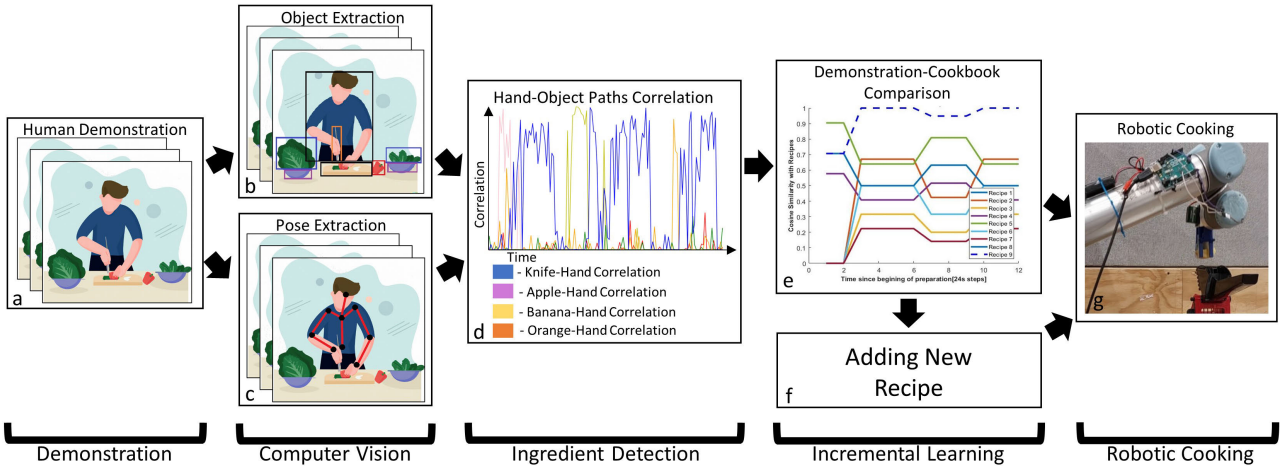

FIGURE 1. Schematics of the robot assessing novelty of new demonstration and learning new recipe when required. a) Robot observes the demonstration with a camera. b) relevant objects like produce and kitchen utensils are detected by using a neural network. c) Pose of the human chef is extracted using a neural network. d) Actions and objects used are determined by finding high correlations between the right hand and a specific object. e) Similarity of the demonstration with all the recipes in the robot cookbook is computed. If the similarity is low robot will proceed to add a new recipe. f) Robot learns new recipes by listing all actions seen in the demonstration and adds them as a new recipe to its cookbook g) Robot makes a salad according to one of the recipes.

use colour masking and feature extraction, while others use publicly available neural networks for object detection and pose extraction. Publicly available networks offering object detection include YOLO [27], SSD [28], RetinaNet [29] and YOLOR [30]. Most of them are trained on COCO dataset [31], which focuses on everyday objects photographed in everyday situations. Pose detection can also be done by publicly available neural networks eg. OpenPose [32], DCPose [33], DensePose [34] and HigherHRNet [35].

In this paper, we are implementing a robotic chef that has a sizable capability for learning new recipes from human observation. The robotic chef visually observes a human chef at work and recognizes the sequence of actions performed by the human, as well as the products they are performed on. This sequence is vectorized and compared with the current robot's cookbook. If a recipe with high similarity is found, the robot makes a salad according to that recipe. In cases when no existing recipes seem to match the human's intention, the list of ingredients is extracted and added to the robot's cookbook as a new recipe. Each of the recipes already in the cookbook can also run without additional demonstrations.

The system was tested on 16 salads prepared according to 8 known recipes listed in Table 1. The correct recipe was recognized successfully in 93% of the demonstrations, even though only in 83% of the demonstrations all actions were detected correctly. Some variations of the recipes were also demonstrated to the robot to show the robustness of the system to portion size, human mistakes and slight variations in the recipe. This assures that new recipes are not learned due to reasonably small variations in the demonstrations. All 8 salad recipes were made using the robot and all of the ingredients were added successfully across all these salads. The system also correctly recognized the demonstration of a new salad, added it to a cookbook and cooked it.

II. METHODS

A. OVERVIEW

The robot's operation schematic is shown in Figure 1 and starts with the observation using a camera as shown in Figure 1 a. Each frame of the resulting video is analyzed frame by frame using off-the-shelf neural networks. Openpose is used to detect the demonstrator's right wrist(Figure 1b) and YOLO detects all visible objects' positions (Figure 1c). The coordinates of each body part and each object are saved and form a path when extracted from multiple frames. Identification of a grasped item is done

by analysing Pearson's correlations between the demonstrator's right-hand path and the product/kitchen utensil path(Figure 1d). A high correlation is an indication of lengthy handling of an item and therefore is an indication of a certain action. This way the whole demonstration is translated into binary states which we filter with Hidden Markov Model to adaptively filter out observation noise and neural network mistakes. The demonstration is compared to the existing recipes by converting both of them into vectors and computing cosine similarity between the vectors(Figure 1e). If there is no similar recipe in the cookbook, the robot makes a new recipe that includes all ingredients used in the demonstration (Figure 1f), and proceeds to make the salad(Figure 1g).

B. EXTRACTING THE ACTIONS FROM THE VIDEO

1) SCOPE AND HUMAN CHEF DEMONSTRATION

The project attempts to build and test in the lab conditions a framework for teaching the robotic chef recipes by demonstration and investigating its feasibility and limitations. Therefore, as many components as possible should be adopted from other branches of robotics, as redoing any of these would not contribute to the project. These limitations make the choice of the dish cooked very important. For example, trained neural networks for object detection exist, and training them from scratch does not contribute to the project, hence we use the available models, trained on large, already available datasets. In the case of item detection, the available networks are trained on the COCO dataset. It limits the system to the classes in COCO dataset. Available classes cover cutlery: knives, forks and spoons, some containers: bottles, glasses, cups and bowls, kitchen appliances: microwave, oven, toaster, sink and fridge, as well as 10 food products: banana, apple, sandwich, orange, broccoli, carrot, hot-dogs, pizza, doughnut and cake. Under this limitation, we have decided that a reasonable dish to teach to the robot is a variety of salads, as they use a larger portion of items from the listed classes. Moreover, making salads is possible with the dexterity of the current robots, especially if some hardware compromises are made in the kitchen.

Furthermore, we need to ensure some structure to the salads we produce for experimentation. This will be crucial both for the reproducibility of the results as well as for the ease of programming the physical setup. Therefore, we limit the salads to 5 ingredients: broccoli, carrot, apple, banana and orange. Moreover, we chose to define our recipes by the number of each fruit/vegetable added. The only exception is the broccoli where we add a smaller amount of it as a single unit. We also make sure that the salads produced are reasonable recipes whenever possible. Inspecting the list of the recipes used in the experiment shown in Table 1, we can see that recipes 1 through 3 are a variation of the popular apple and carrot slaw. Recipes 4, 5 and 8 are variations of a fruit salad, and recipes 6 ad 7 are variations of vegetable salads.

We choose to perform our work on various salads, as their production is relatively easy to automate, and relatively many ingredients are recognizable by the YOLO network. The next step is preparing a human demonstration, as its quality and camera angle can make the task significantly harder or easier. We make the demonstrations in the laboratory while making the space messy enough to simulate a busy kitchen, that seems to be utilized with no specific care. In the foreground, we set up a table with all ingredients used in the salads, a large number of kitchen utensils, as well as some food products that are not ingredients used by the robotic chef. We have the camera filming from the front, slightly from above. The demonstrator has a chair behind the table, from where he can prepare the salad. The setup is shown in Figure 2 a.

2) OBJECT AND POSE EXTRACTION

The learning procedure starts with extracting the pose of the demonstrator as well as the items present in the video. The pose of the demonstrator is extracted using the OpenPose neural network. Object detection is done using YOLOv5m neural network. The whole process of extraction is shown in Figure 2. Firstly, the video is separated into single frames for processing. Both neural networks are then applied to each of these frames. Pose Keypoints and bounding boxes of all identified items are saved as a JSON file. We also draw the pose and bounding boxes in each of the frames, as presented in Figure 2 a, b, and c, making sure that each of the networks is applied to an unaltered frame. As the last step, all the frames are stitched into a video again, making it easy to assess the network's performance and present the results.

3) GRASPED ITEM IDENTIFICATION

We use the pose keypoints and detected objects' positions from the previous step to identify which item is currently grasped and therefore used. This step is necessary to know which item is being used, as multiple objects are present in the frame while cooking. The pose keypoints come in the form of (x,y) coordinates for every major joint and some face features. Similarly, the YOLO network returns the (x,y) positions and class labels for each of the detected objects. These positions, when gathered across multiple frames of a video make up a trajectory for each of the objects, and the demonstrator's joints. Therefore wrist trajectory $\mathbf{S}$ can be described with the following equation:

$$\mathbf{S} \in \mathbb{R}^{n\times 2} \tag{1}$$

and the tracked object trajectory is described with the following equation:

$$\mathbf{O} \in \mathbb{R}^{n\times 2} \tag{2}$$

where n is the total number of frames, the first column is the x position and the second column is the y position. We can then use the similarity between the trajectories to find which object movements are correlated with movements of other objects or some important demonstrator's joints.

We aim to calculate this path similarity using Pearson correlation. The advantages of this technique include ease of

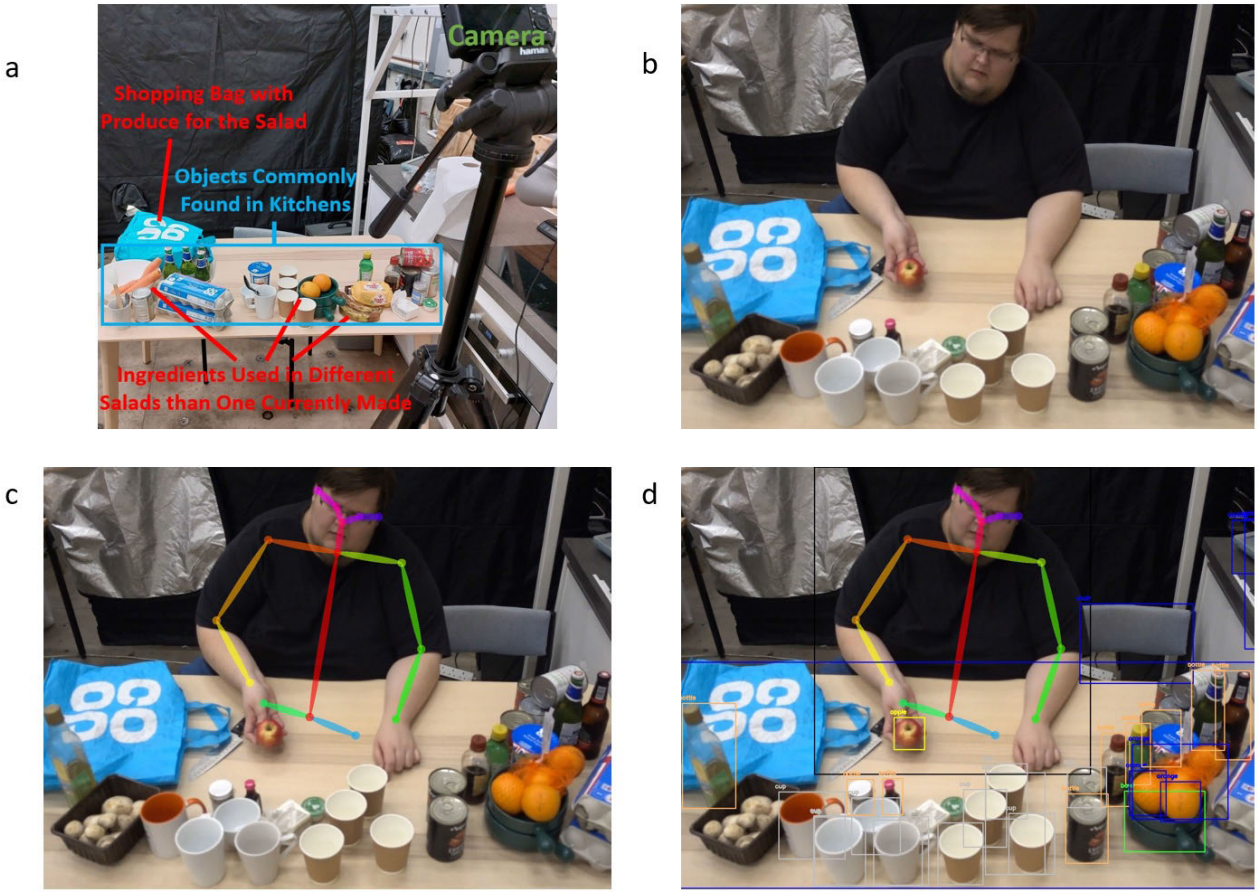

FIGURE 2. Procedure of recording and processing the human demonstration: a) Setup used to record the demonstration, b) single frame of a human-chef demonstration, c) the frame with body pose extracted, d) the frame with both body pose and items extracted.

application, in-built scaling, solid theoretical background and inherent affinity for scoring less noisy movements high. The in-build scaling is crucial for two reasons. Firstly, we should not be biased towards actions including large movements over those including only smaller ones, as for example noticing moving an ingredient across the table is no more important than noticing the cutting of it. Secondly, the in-build scaling allows a maximum correlation between two paths of the same shape, but different sizes. This is crucial as the path taken by the wrist is usually a smaller copy of the path of an object kept in hand. The Pearson correlation takes a value between -1 and 1, and can be computed for short parts of a video, in our case we used windows of 50 frames - the equivalent of two seconds.

Pearson correlation is not able to tackle the raw trajectories as it is a correlation between single-column matrices, while each trajectory is of shape n by 2. Therefore, the computation of the final correlation score must be split into a few stages. We will now follow this process on a wrist trajectory $\mathbf{S}$, but the process is the same for trajectory $\mathbf{O}$. We start by splitting the trajectory S (equation 1) into two separate single column matrices - one for storing x positions $\mathbf{S_x}$ and another for y positions $\mathbf{S_y}$:

$$\mathbf{S} = (\mathbf{S_x}, \mathbf{S_y}) \tag{3}$$

then we split these arrays into 50 frame windows. In the case of $\mathbf{S_x}$ these windows are specified by the following equation:

$$\mathbf{s_1^{(x)}} = \mathbf{S_x}[1:49] \tag{4}$$

for the first window s_1 and equation below for a window number t.

$$\mathbf{s_t^{(x)}} = \mathbf{S_x}[50(t-1)+1:50t] \tag{5}$$

therefore the x positions in trajectory($\mathbf{S_x}$) becomes a sequence of such windows:

$$\mathbf{S_x} = \{\mathbf{s_1^{(x)}}, \mathbf{s_2^{(x)}}, \ldots, \mathbf{s_t^{(x)}}\} \tag{6}$$

$\mathbf{S_y}$ also undergoes the same process, but for the sake of reducing the number of equations, we will follow only $\mathbf{S_x}$ for now. The object trajectory is transformed in the same way and the $\mathbf{O_x}$ and $\mathbf{O_y}$ also are transformed to the form shown below:

$$\mathbf{O_x} = \{\mathbf{o_1^{(x)}}, \mathbf{o_2^{(x)}}, \ldots, \mathbf{o_t^{(x)}}\} \tag{7}$$

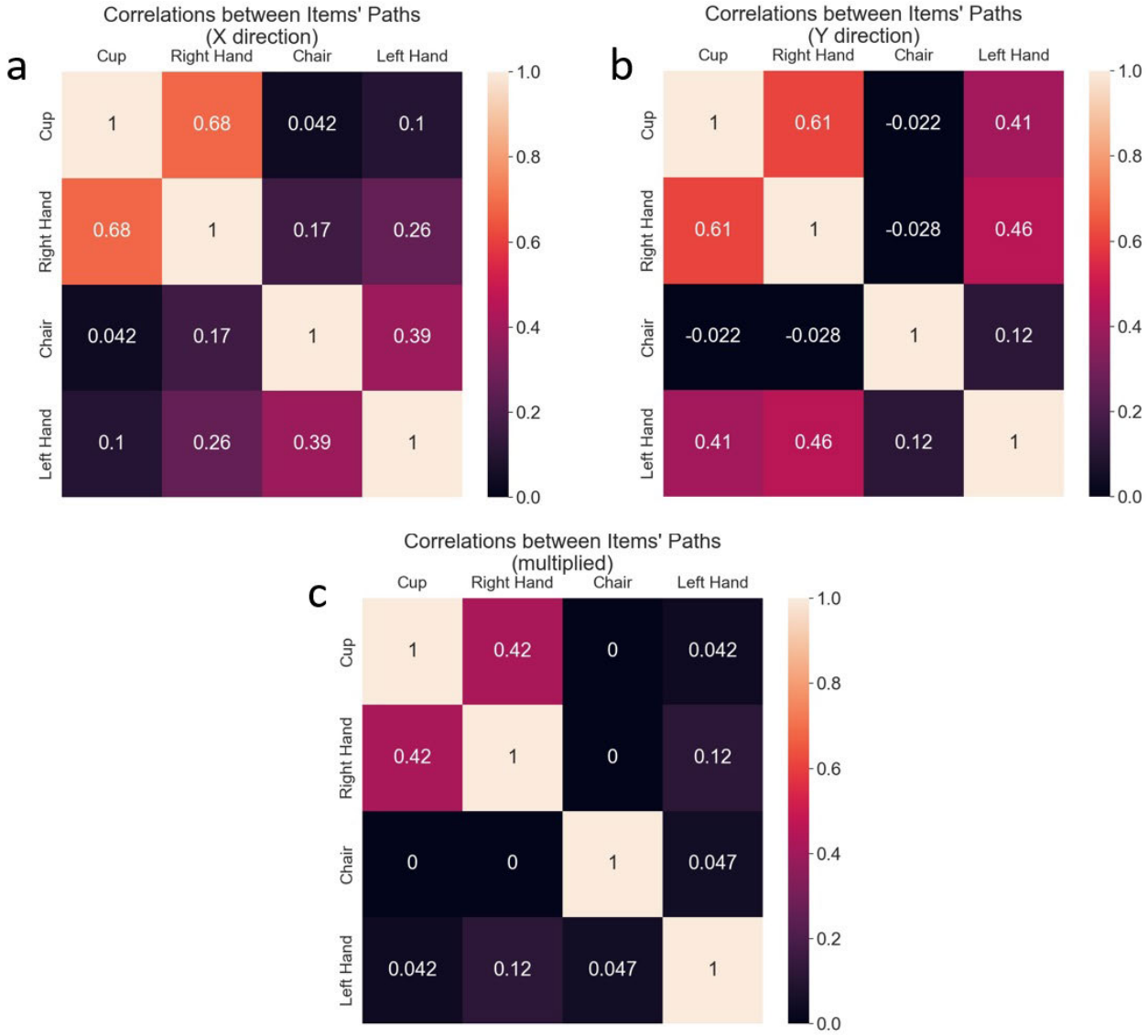

FIGURE 3. Figure showing the correlation between positions of 4 objects - left and right hands, a cup and a chair. The cap was moved around using the right hand. a) correlation between paths in x coordinate b) correlation between paths in y coordinate c) product of correlations in x and y coordinate.

where $\mathbf{o_m}$ denotes m-th window of the object trajectory. In this case, we will also follow the x direction only for now.

Now we can compute the Pearson correlation ρ between any window in the trajectory of an object and the trajectory of the wrist. For example, the correlation between the first windows would be expressed with the following formula:

$$\rho_1^{(x)} = \rho_{\mathbf{s_1^{(x)}}, \mathbf{o_1^{(x)}}} = \frac{cov(\mathbf{s_1^{(x)}}, \mathbf{o_1^{(x)}})}{\sigma_{\mathbf{s_1^{(x)}}} \sigma_{\mathbf{o_1^{(x)}}}} \tag{8}$$

Notice that this produces a correlation between paths either in x or y coordinates only, therefore the superscript x is used. Finally, we arrange the window-wise correlations in chronological order to build the correlation for the whole trajectory. This correlation $P^{(x)}$ between an arbitrary joint and object "x trajectories" $\mathbf{S_x}$ and $\mathbf{O_x}$ is denoted with the following equation:

$$\mathbf{P}_{\mathbf{S,O}}^{(x)} = \mathbf{P}_{\mathbf{S_x,O}_x} = \{\rho_1^{(x)}, \rho_2^{(x)}, \ldots, \rho_n^{(x)}\} \tag{9}$$

We have now produced two arrays describing the correlation between wrist and path trajectories - $P_{\mathbf{S,O}}^{(x)}$ and $P_{\mathbf{S,O}}^{(y)}$. We need to combine them into a single number explaining the correlation in each window. Let's try to find an element-wise operation to merge $P_{\mathbf{S,O}}^{(x)}$ and $P_{\mathbf{S,O}}^{(y)}$. We observe that similar paths would result in both $\rho^{(x)}$ and $\rho^{(y)}$ positive and significantly larger than zero. Therefore we come up with the idea of multiplying $\rho^{(x)}$ and $\rho^{(y)}$, which would give a high result in the case when both of them are high. The only edge case to take care of is when the movement of the object and hand are "reverse" of each other. In this case, multiplying two large negative numbers give a high positive

number. Therefore, to compute the final correlation we set every negative correlation and set it to zero, and then take a product. Therefore for the window number t, the correlation is computed with the following equation:

$$\rho_t = (max(\rho_t^{(x)}, 0) * max(\rho_t^{(y)}, 0)) \tag{10}$$

and the final correlation is given with the equation:

$$\mathbf{P_{S,O}} = \rho_1, \rho_2, \ldots., \rho_t \tag{11}$$

This results in a single array containing a correlation between a joint and an object for each 50-frame window in the video. This operation can be done on arbitrarily many pairs of joints/objects.

The best way to show this in operation is by using a matrix with different objects/joints corresponding to columns/rows. This is due to the fact that this correlation can be computed between any pair of joints/objects, therefore every matrix entry can correspond correlation between the row name and column name. We use an example video of a cup held in a hand to show correlations of X coordinates, Y coordinates, and finally the final correlation score for all possible correlation pairs between the hand, the cup, as well as a chair and another hand visible on the video. These results are shown in Fig 3.

4) ACTION IDENTIFICATION WITH HIDDEN MARKOV MODEL

The next step is to identify the actions needed to describe salad-making and assign each of them to a correlation between pair of objects/joints described in the previous section. The minimum necessary number of actions should be listed in order to minimize the data required for machine learning. The solution we choose to this problem is to have an action of adding each of the ingredients, an action of cutting and an action of ''resting''. The ''Resting'' action covers everything that is not any of the other actions. Therefore, the following actions must be extracted from the video: *Resting, Cutting/Chopping, Adding Broccoli, Adding Carrot, Adding Apple, Adding Banana, Adding Orange*.

These actions are to be found based on the correlation score between a pair of joints/objects discussed before. We assign adding a specific ingredient to the correlation between the path of the right wrist and the path of the said ingredient. This will be high when the movements of this wrist and that specific product are synchronized. We choose the right wrist specifically as most people are right-handed, making it a dominant wrist. This can be easily changed to the left wrist if needed, or both wrists could be used. We choose to stay with the use of only the right wrist for this experiment as we noticed that the non-dominant hand is usually used for not crucial tasks while the dominant hand is performing tasks crucial for the recipe. Detection of cutting follows a very similar procedure, but this time it is a correlation between the right wrist and a knife. A low score in all of the above correlations is considered detection of ''resting'' or no action.

This kind of action detection can result in both false positive and false negative detections, therefore we intend to incorporate some sort of smart filtering. We propose to use the Hidden Markov Model(HMM) for that purpose. We base this decision on our experience with analyzing preliminary results, where a human would look at the correlation scores across the whole length of a video and deduce the most probable sequence of actions. Finding the most probable sequence of hidden states is one of three fundamental tasks of HMM model, making it a great match. Moreover, the relatively small amount of data necessary to train the model is a great asset when working with food and producing more data is labour-intensive and cost-intensive.

Applying the HMM to this problem will require some data processing. Firstly, we need to change the correlation scores into action detection. We do compare the correlation scores to the item/joint pairs discussed before. Each window is then assigned an action with the highest corresponding correlation score. We also add a threshold of 0.5, and if the largest corresponding correlation is smaller than this threshold, we set the action during the window to ''resting''. Having strictly one action assigned to each of the time windows we can proceed to design the HMM. This application of HMM relies on a clever yet intuitive interpretation of hidden states and observations. We treat our action detections as observations, while the hidden states are the actual true actions happening on the video. The graph showing the model is shown in Figure 4. The first step is to estimate the transition matrix T and emission matrix E. This can be done by simply counting the number of times a specific transition and dividing it by the total number of transitions. The same thing can be done for emissions. The only hurdle is that we need a ground truth sequence of states to do this learning. Therefore, firstly we create a human ground truth on what happens in the video. Then we are using half of the salads produced to train the model. This effectively produces a transition matrix T and observation matrix E that can be used to do the other fundamental task of HMM - given the observation, return the most probable sequence of hidden states. Viterbi algorithm is used for this task. Looking back on Figure 4, we run the computer vision discussed before to get the sequence of observations y, to then calculate the most probable sequence of human actions x. Running this task on any new set of observations is effectively a smart filter to get rid of false detections in a video. In the main experiment, we train the filer on half of the data and use it to filter another half. As each of the recipes was done twice, the training sets are balanced, and we effectively do a two-fold validation.

After HMM filters out the actions detected we expect to see a sequence of windows with the same actions detected. The ingredients added to a given salad can then be easily found as the salad will be represented there as a sequence of blocks where each block consists of handling an ingredient and then cutting it. Of course, there would be multiple points where ''nothing'' is detected. These are ignored during the ideal operation of the program but also can introduce errors in a situation when they occur during handling a single ingredient, causing a single portion to be seen as two portions. Ideally,

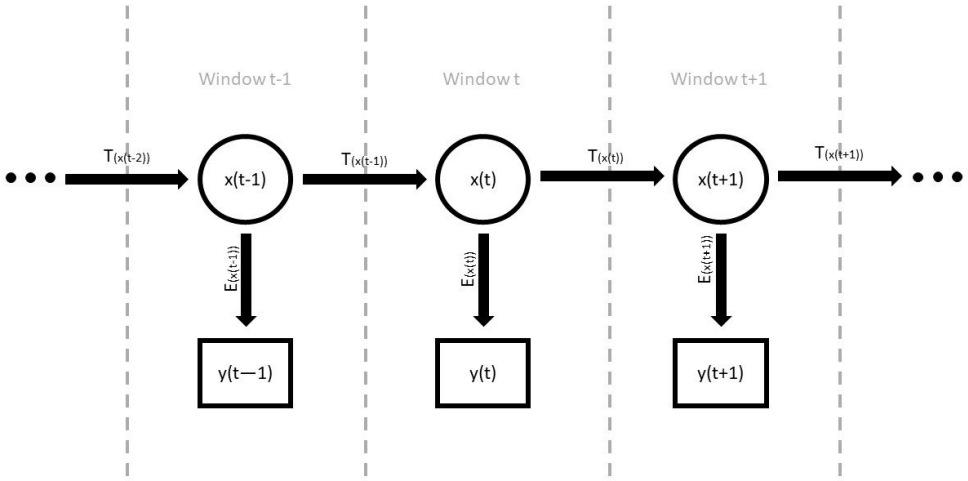

FIGURE 4. Graph of Hidden Markov Model used. "x" represents a hidden state which is the actual action performed by the human in the demonstration. These are modelled as a probabilistic transition governed by transition matrix T. Transition probability depends only on the previous state. "y" represents emissions of the hidden state, which is what the system detects. It depends only on the current state and emission probabilities are stored in emission matrix E.

this part of processing scans the feed from the camera or a video and returns the ingredients used so far.

5) CHOOSING THE INTENDED RECIPE

Finally, we need to compare how similar the already seen part of the human demonstration is to each of the recipes. To do this we take inspiration from the vord2vec [36] algorithm that is commonly used in Natural Language Processing(NLP) to describe each word by a feature vector. These features are usually very impressive at capturing the meaning of the words [37], allowing comparing the similarity of the worlds [38]. In this paper, we use a simpler vectorization, where the amount of each ingredient makes up a feature of the feature vector, but similarly to NLP, we use the cosine similarity to find the similarity between the vectors, hence the recipes. The vectorization of a demonstration $\mathbf{D}$ is described with the following equation:

$$\mathbf{D} = [B_d, C_d, A_d, Ba_d, O_d] \tag{12}$$

Similarly, vectorization of a recipe $\mathbf{R}$ is given by:

$$\mathbf{R} = [B_r, C_r, A_r, Ba_r, O_r] \tag{13}$$

B, C, A, Ba, O are numbers of portions of respectively broccoli, carrot, apple, banana and orange added. Subscript $_r$ means that it is the number seen in the recipe, while subscript $_d$ means that it is the number of ingredients detected in the human-chef demonstration. Finally, the cosine similarity S_{cos} between them is computed, by definition, with the following equation:

$$S_{cos}(\mathbf{D}, \mathbf{R}) := \frac{\mathbf{D} \cdot \mathbf{R}}{||\mathbf{D}|| * ||\mathbf{R}||} \tag{14}$$

The feature vector can be computed both for the already-seen part of the cooking demonstration and for each of the recipes. The cosine similarity can be computed between any arbitrary pair of these vectors. For finding the most probable intention we compute the cosine similarity between the detections in the camera feed and each of the recipes. The most probable intended recipe is the one with the highest similarity to the video demonstration. Moreover, using the vectorization of the salad during the preparation makes the computed cosine similarities can change frequently, possibly every 50-frame window. Therefore, the prediction of human intention can also change, which we hope to see clearly in the results.

C. INCREMENTAL LEARNING

Incremental learning of the cookbook is implemented in two stages. Firstly the robot watches the demonstration of a human making a specific salad and compares its similarity to all recipes in the cookbook. In the case when at least one of the similarities is higher than an arbitrary threshold, the robotic chef proceeds to prepare a salad according to the recipe with the highest similarity. This condition is expressed by the following equation:

$$max(S_{cos}(\mathbf{D}, \mathbf{R})) > t, N = \{1, 2, 3, \ldots, L\} \tag{15}$$

where t is an arbitrary threshold, N is a number of a recipe, L number of recipes in the robot's cookbook, and the rest of the symbols have the same meaning as in equations 12, 13 and 14.

If on the other hand, the similarity to all of the demonstrations is below the threshold, the robot adds a new recipe to the cookbook with a list of ingredients detected in this human demonstration. Therefore the new L+1th recipe vectorization becomes:

$$\mathbf{R}_{(L+1)} = [B_d, C_d, A_d, Ba_d, O_d] \tag{16}$$

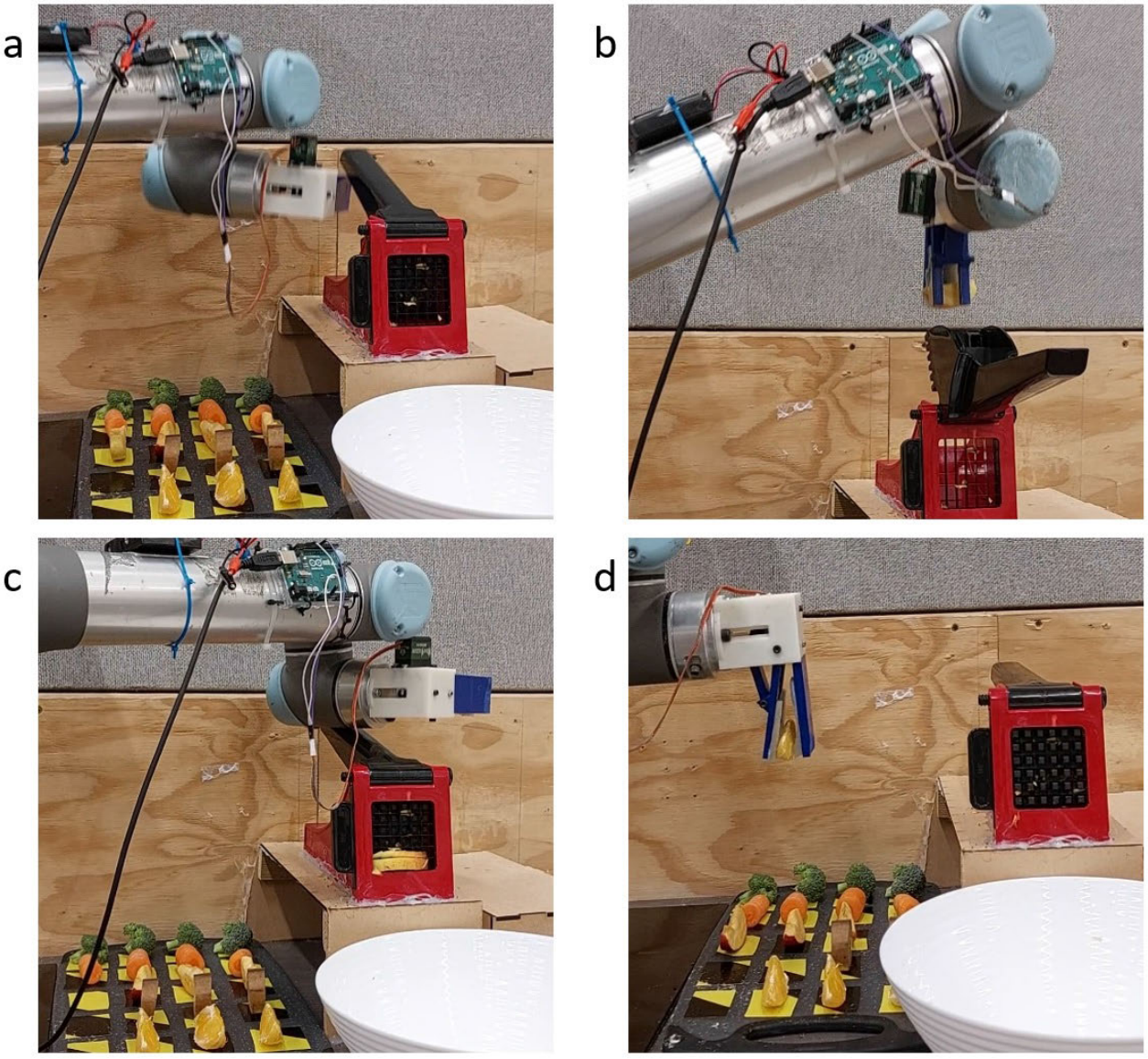

FIGURE 5. Robotic chef making salads a) opening the fry cutter b) placing the ingredient inside the cutter c) cutting the ingredients by closing the fry cutter d) delivering orange straight to the salad bowl.

This new recipe can now be cooked by the robotic chef without further programming thanks to modular programming of the cooking process.

D. ROBOTIC CHEF IMPLEMENTATION

1) ROBOTIC KITCHEN SETUP

We are now implementing a robotic chef to prepare the recipe that was determined to be the intended recipe. The robotic chef will be a robotic arm hanging downwards from a metal cage around the kitchen. Most of the operation is based on moving objects between known positions as the kitchen setup is highly structured and designed with the robot in mind. The setup consists of a fry cutter, an ingredient storage area and a bowl for the salad. The fry cutter allows for cutting all the ingredients easily when using the robotic arm. Moreover, it improves safety by eliminating the need to use a knife. Furthermore, the fry cutter provides a force multiplier for the arm, due to an in-built leaver. This is important as free-standing robotic arms have a limitation on the maximum force provided. The storage area keeps the ingredients in predefined positions, from where the robotic arm can pick them up. Finally, we place a bowl in front of the fry cutter to collect the salad.

2) MOVEMENT PROGRAMMING

The UR5 arm was programmed using Python. This allowed the integration of the robotic arm control and computer vision into one program. The program was structured in a hierarchical manner so the most basic moves like going to the home position, movement at a certain speed and opening/closing

the gripper were programmed first. These functions were then used to program the next layer which consisted of the following functions:

- Opening fry cutter (Figure 5a).
- Moving ingredients from starting location to the home positions.
- Moving ingredients from the home position to the fry cutter (Figure 5b).
- Closing fry cutter (Figure 5c).
- Moving (some) ingredients straight to the bowl (Figure 5d).

The functions were then put together into larger functions that combine all steps needed to add a single ingredient to the salad. An example function of this kind consists of opening the cutter, picking up a piece of carrot, moving it to the home position, then dropping it inside the fry cutter and cutting it. Different snapshots of the robot performing this function are shown in Figure 5. These functions can be easily combined into recipes, by executing a few of them in a sequence.

III. RESULTS

A. CORRELATION COMPUTATION

Correlations are computed based on paths extracted by YOLO and openpose neural networks. We do not report on their performance as there is ample literature available on their performance. Nevertheless, we provide both the raw human demonstration footage as well as the footage with the pose and detection boxes drawn on each frame for an interested reader to investigate. The raw correlations extracted from a demonstration of a salad cooked with recipe number 1 are shown in Figure 6 a. The human eye can easily notice a few regions of elevated correlation and identify the story of what happened. We clearly see a small area, where ''nothing'' is going on, followed by an area where correlation to the apple seems to be highest, then to the knife, followed by a short break, and then by carrot, knife and again a break. It is quite intuitive for a human to see that the short deeps in the correlation scores are probably an error rather than a break between handling two different apples. This is due to the fact that each activity takes multiple frames, therefore any change is quite an unlikely occurrence. In the next section, we will train a HMM, with the hope that it makes use of this quality.

B. ACTION EXTRACTION MARKOV FILTERING

The initial action extraction is made with the previously described algorithm and the result can be seen in Figure 6 e, as a red ''x'' line. As we can see there are a lot of false detections that make each action segmented, and therefore potentially considered as multiple actions. Therefore, we are training the HMM as discussed in the methodology section - the identified actions will be considered observations and the human ground truth is considered a hidden state. As each salad was done twice, we always train the model on one batch of 8 salads and use it for predictions on the other batch. Below we show the transition matrix computed from the first batch of 8 salads:

$$\begin{bmatrix} 0.882 & 0.016 & 0.028 & 0.024 & 0.016 & 0.024 & 0.008 \\ 0 & 0.909 & 0 & 0 & 0 & 0 & 0.090 \\ 0 & 0 & 0.888 & 0 & 0 & 0 & 0.111 \\ 0 & 0 & 0 & 0.888 & 0 & 0 & 0.111 \\ 0 & 0 & 0 & 0 & 0.871 & 0 & 0.129 \\ 0 & 0 & 0 & 0 & 0 & 0.877 & 0.122 \\ 0.051 & 0 & 0 & 0 & 0 & 0 & 0.948 \end{bmatrix}$$

Firstly, we should discuss the meaning of each of the entries in the transition matrix. Each entry is a probability that a hidden state will transition from a state represented by its row to a state represented by its column. The order of state representation is the same for both rows and columns. The first row/column represents the state of no action. The rows/columns 2-6 represent handling ingredients, the order is the same as their listing in Table 1 - broccoli, carrot, apple, banana and orange. The last row/column represents cutting.

The first characteristic we can see is that the largest probabilities are strongly concentrated on the diagonal of the matrix. It means that it is very likely that an action performed in the current 50-frame window will be performed in the next window. Another interpretation is that the actions performed during the cooking are on average done in blocks much longer than the detection window itself. It means that any short disruptions in the action detection will likely be filtered by the HMM, as sequences with such occurrences will look less probable. Furthermore, we see a lot of non-zero entries in the top row. These are transitions from a ''nothing'' state to one of handling ingredients or cutting states. These numbers change how likely the model is to accept such a transition. As these numbers are quite low, the most likely sequence of states computed by the model is likely to contain these transitions only when supported by very strong observations. This is the method to dismiss false positives. Another observation is a large non-zero cluster of values in the last column. This is a chance that from a state designated by the row, we move to ''cutting''.

The next step is to analyze the emission matrix. The row/column coding here is the same as in the transition matrix, but this time rows are hidden states, and the columns are emissions. The matrix, computed from the first bath of 8 salads, is shown below:

$$\begin{bmatrix} 0.979 & 0 & 0 & 0 & 0 & 0 & 0.020 \\ 0.613 & 0.386 & 0 & 0 & 0 & 0 & 0 \\ 0.317 & 0 & 0.682 & 0 & 0 & 0 & 0 \\ 0.129 & 0.018 & 0 & 0.85 & 0 & 0 & 0 \\ 0.290 & 0 & 0 & 0 & 0.709 & 0 & 0 \\ 0.244 & 0 & 0 & 0 & 0 & 0.755 & 0 \\ 0.286 & 0 & 0 & 0.002 & 0.002 & 0 & 0.709 \end{bmatrix}$$

Analyzing the matrix we first see that most non-zero entries are clustered on the diagonal and first column. The diagonal is the probability of observing a state as the same action as the one represented by this state. These values are the probability of our system(before HMM filtering) of correctly

interpreting the state in the video in a single window. The higher this number is, the more serious the detections of the corresponding action will be treated by the HMM when filtering. The second cluster is in the first column and represents the situation when some state was detected as ''nothing''. This shows that it is relatively common for the system to just miss a detection. This is now considered by HMM and it is likely to calculate that it is more probable to ''ignore'' moments where nothing is detected a find a sequence of states where it is more likely that some action actually continued and was simply not picked up by the computer vision. The number is also a lot of zero-entries. This means that in the training dataset, this observation never happen. This shows that computer vision together with Pearson correlation is very good and prevents false detections.

Finally, we can analyze the effects of HMM filtering. Firstly, we investigate the improvement in action detection for a single frame by looking at the confusion matrix before filtering(Figure 6 b) and after filtering(Figure 6 c). The apparent change we can see is a migration from the top row to the diagonal. It means that the HMM has successfully corrected many instances when an action was not detected by the computer vision employed. This is also confirmed by plotting a human-made ground truth, raw detections of our system and filtered detections, as we have done in Figure 6 e. We can make a few interesting observations about the performance of the filter based on this figure. Firstly, we see that it indeed ignores short periods when the computer vision failed with detection. We see examples around windows no. 40, 80 and 150. Furthermore, the model seems to filter out misdetections around the time when the demonstration switch from handling the ingredient to cutting. It correctly assumes that it is more probable for the human to instantly start cutting after he has finished handling the ingredient, rather than have a random rest between these actions. We see an example of this around windows no. 25-30.

C. INTENTION RECOGNITION WITH COSINE SIMILARITY

The filtered signal allows now to easily extract the ingredients added to the salad at any point in time. It is then vectorized in the same manner as the recipes and cosine similarity between this vectorized demonstration and vectorized recipes can be computed. We will now follow the changes of these cosine similarities in time for different scenarios.

Firstly, we investigate the similarity computation during observation of a demonstration of recipe no. 2 in Figure 6 d. All actions were detected successfully. The order of the ingredients added was ''carrot'' and apple three times. Right at the beginning of the plot we already have some non-zero correlation value for all the recipes, meaning that the first ingredient is already added in the first 24 seconds of the video. As we see recipes 6, 7 and 8 have a zero correlation. This is because they contain no apple making them orthogonal to the current demonstration vector containing only the apple at this point. Other recipes have some similarities to the demonstration at this point, but their values depend on how ''apple-heavy'' the recipes are. The one with the highest score at the beginning is recipe 3, which has apples as two out of three ingredients. Then carrot is added three times in a row, and these additions correspond to the three changes in the plot. At the first change, we see that recipes 6 and 7 increased their similarity as they contain some carrots, while recipe 8 is still at zero as it contains no carrots or an apple. The rest of the recipes changed their similarity. We will now focus on recipes 1, 2 and 3 as they have more interesting trends, even when more of the same ingredient is added. Firstly, recipe 1 peaks all the way to the similarity of 1 and then falls with the subsequent additions of carrot. This is because the demonstration becomes more and more carrot heavy, while the recipe is calling for an apple-to-carrot ratio of 1. Recipe 2 is the one that was indeed demonstrated, and we see a constant trend towards the similarity of 1. Interestingly, each of the ingredients added that gets closer to the exact copy shows diminishing returns. Recipe 3 has a very similar path to recipe 1, but never as close to the similarity of 1, as it is in reality apple-heavy.

We will now investigate the behaviour of the system in less optimal conditions. We will use recipe 1 for this test as a base, as it will make our analysis easier. Three scenarios will be tested here, and three times more ingredients will be used than in the recipe itself to allow for more interesting scenarios. Firstly, in Figure 7 a, we can the evolution of the demonstration similarity for a demonstration where a portion of apple and a portion of carrot is added three times in the alternating pattern. One of the ingredients was not detected by the computer vision in this demonstration, and this misdetection corresponds with the long flat area in the plot. In this case, we see that adding another pair of ingredients makes the similarity oscillate. Also, we see that the system has successfully found the intended recipe. The score for recipe 2 was also very high. That is because the missed ingredient was an apple, making it a carrot-heavy apple-carrot slaw, which is what recipe 2 is.

Figure 7b, shows a scenario of adding an error in a form of a different ingredient. For this ingredient to be an error at another part of a recipe it needs to be in a relatively lower concentration than the ingredients listed in the recipe. Therefore as this error ingredient, we use a single piece of an orange, while we use free pieces of apple and three pieces of carrot for the base recipe. This is triple the number of ingredients used in a recipe and we added them in the alternating pattern. The resulting figure of cosine similarities is very interesting. In the beginning, when the first ingredient was found by the algorithm already we see that most of the ingredients have a 0 cosine similarity. This is because most of the recipes do not contain oranges at all. This is also true about our target recipe therefore we are likely to see a huge transition when the remaining ingredients will be added. We can see three major trends in the plot. Firstly, the recipes that included orange start with a high cosine similarity which steadily falls down. Those are recipes that did not contain never cared or an apple and each addition of any of those ingredients is moving the

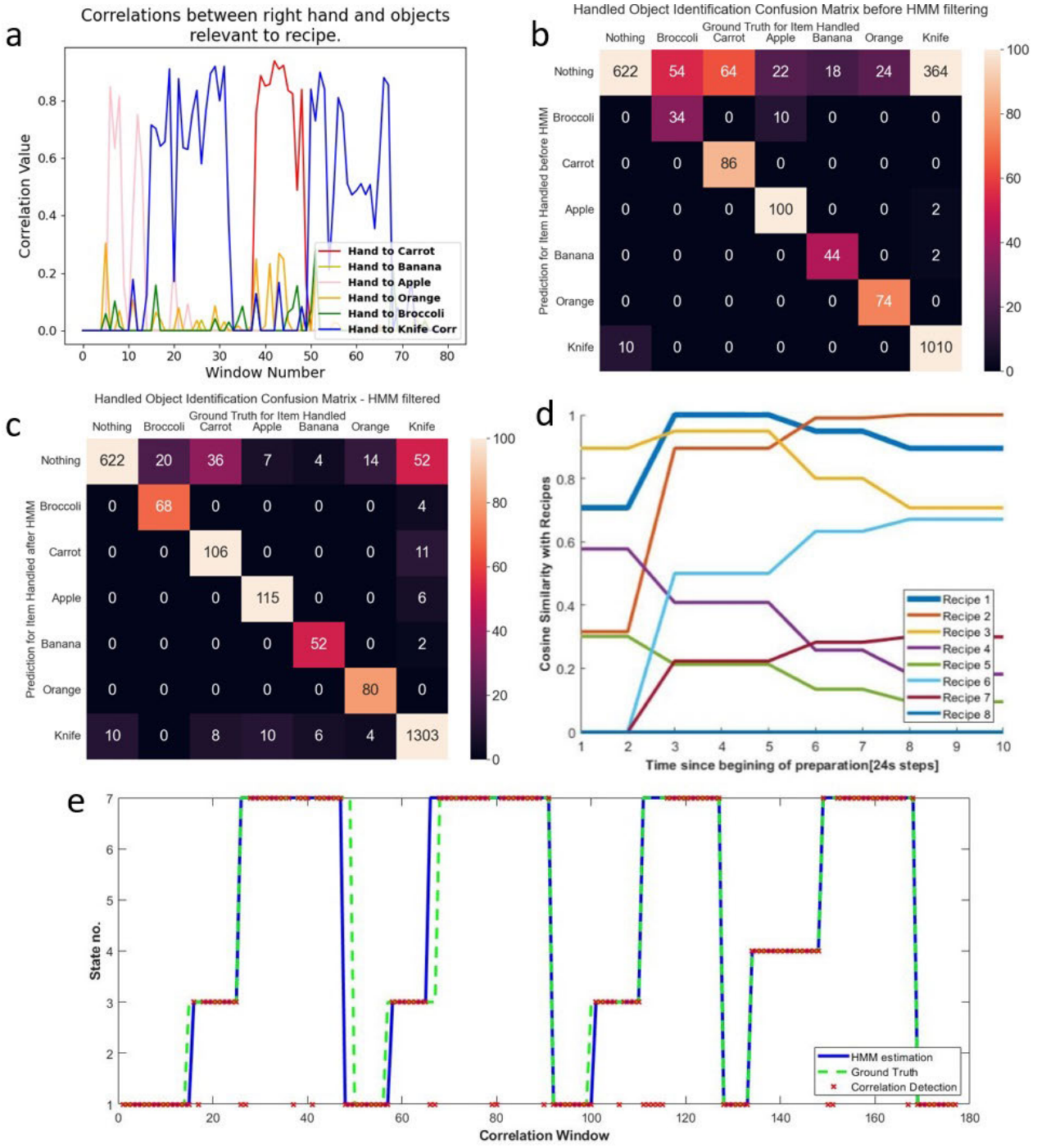

FIGURE 6. Results of robot reacting to a known recipe. a) Raw correlation scores between paths of the right hand and objects relevant to tasks done when assembling a salad. Correlations are computed in every 50-frame window across the whole human cooking demonstration of recipe 1. b) Confusion Matrix showing the results of extraction of the handled item before Markov Filtering and c) is the corresponding matrix after filtering. Both are compiled based on demonstrations of making 16 salads according to 8 recipes. Two-fold validation was used. d) Evolution of cosine similarity between the human demonstration of making a salad with recipe no. 2 and all 8 recipes present in the starting cookbook. e) Visualization of Markov filtering. Demonstration of cooking according to recipe no. 2 is used. States 2,3 and 7 correspond to handling a carrot, handling an apple and cutting. The actions detected by computer vision are marked as red crosses. Ground truth is shown as a green dashed line and a solid blue line in the sequence after HMM filtering.

vector further away from the vector of those recipes. The second group are recipes in which science similarity tends to oscillate. Those are recipes that contain either carrot or apple, therefore, the addition of one of those is increasing their similarity and adding another one is decreasing them. Finally, we see our recipe 1 which starts from a cosine sim-

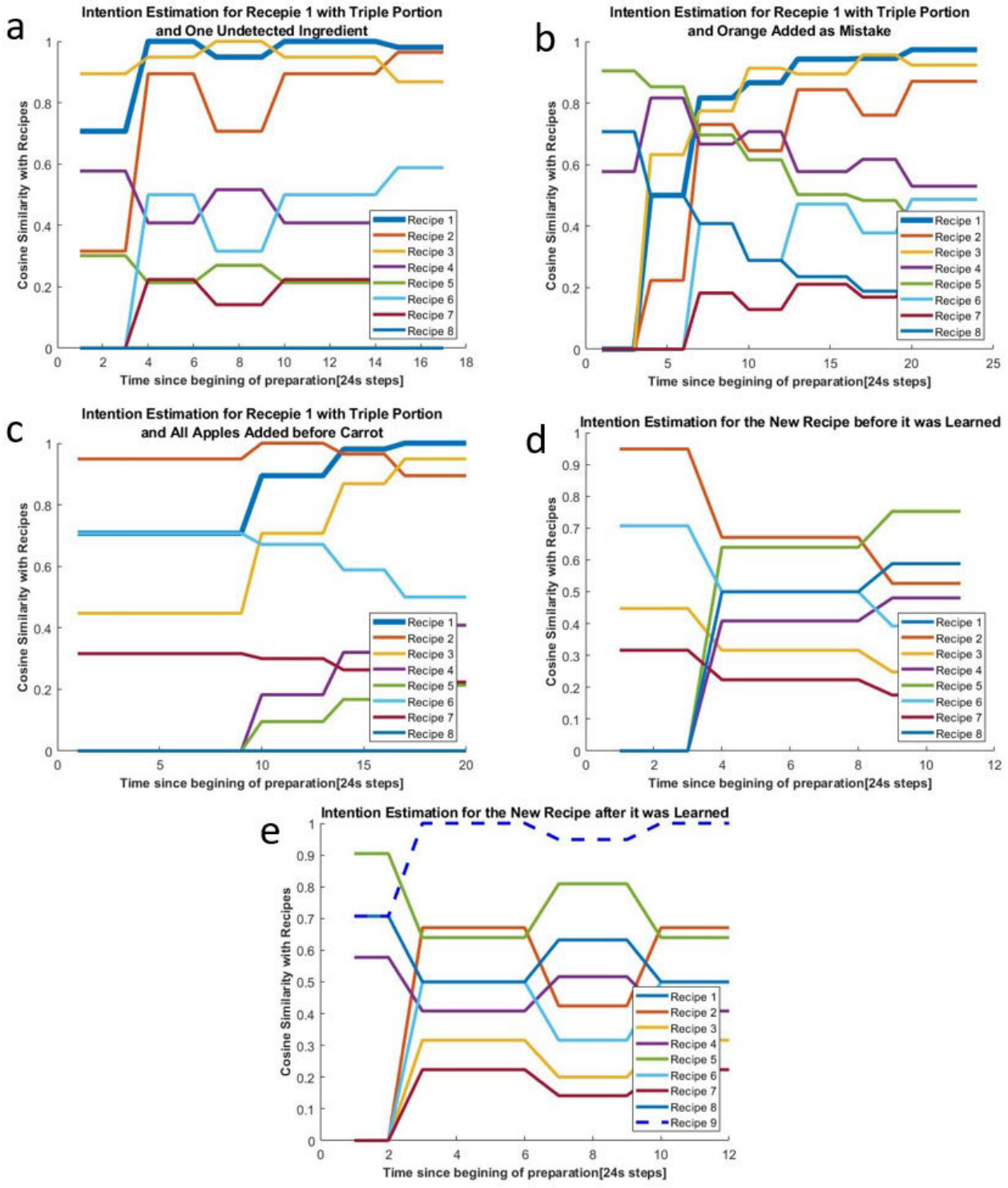

FIGURE 7. Robot's reaction to salad compositions not present in the cookbook. a) demonstration of recipe 1, but with triple the amount of ingredients added and one ingredient misdetection. The robot classifies it as recipe no. 1. b) again recipe 1 with triple the amount of ingredients but also a single piece of orange added as a decoy. The robot still classifies it as a recipe. 1 c) again recipe 1 but now all apples are added before the carrot. Cosine similarity is constant regardless amount of apples added at the beginning, as the robot understands the proportions of ingredients well. d) New recipe consisting of 2 carrots and 2 oranges is demonstrated and the robot does not find the recipe of high similarity in the cookbook. e) After the new recipe is added the robot clearly recognizes this recipe in another demonstration of this recipe where ingredient order was mixed.

ilarity of zero and steadily climbs up in similarity with the addition of any of these ingredients. This is because we are adding apple and card in an alternating fashion which means that each pair of addition is increasing a component parallel

TABLE 2. Experimental results showing the amount of the ingredients extracted from 16 demonstrations of 8 salad recipes, and the number of ingredients successfully added by the robot replicating that recipe.

	Broccoli Pieces (Extraction 1/ Extraction 2/Robot)	Carrot Pieces (Extraction 1/ Extraction 2/Robot)	Apple Pieces (Extraction 1/ Extraction 2/Robot)	Banana Pieces (Extraction 1/ Extraction 2/Robot)	Orange Pieces (Extraction 1/ Extraction 2/Robot)
Recipe 1	0/0/0	0/1/1	1/1/1	0/0/0	0/0/0
Recipe 2	0/0/0	3/3/3	1/1/1	0/0/0	0/0/0
Recipe 3	0/0/0	1/1/1	2/2/2	0/0/0	0/0/0
Recipe 4	0/0/0	0/0/0	1/1/1	1/1/1	1/1/1
Recipe 5	0/1/0	0/0/0	1/0/1	2/1/1	3/3/3
Recipe 6	1/1/1	1/1/1	0/0/0	0/0/0	0/0/0
Recipe 7	3/3/3	1/1/1	0/0/0	0/0/0	0/0/0
Recipe 8	0/0/0	0/0/0	0/0/0	2/2/2	2/2/2

to the vector of recipe 1, therefore, diminishing more and more the starting error of adding an orange. In this case, the system again successfully finds recipe 1 as the most similar to the demonstration. Recipe 1 starts to win after adding only three ingredients out of seven. It has some close contenders because recipes two and three are also carrot-apple slaw, just with different proportions of ingredients, hence the similarity between them in the plot.

In Figure 7c, we see another case. In this experiment, all ingredients were detected correctly but they were added in a specific sequence. First, all the carrots were added and then all the apple pieces were added. This has resulted in a very specific plot where for the first half of the cooking cosine similarity didn't change a lot. This is because adding more portions of the same ingredients only elongated the vector which does not change the cosine similarity with any other vector. On the other hand, we see that when we started adding apples each addition changed the cosine similarity. This is because each addition now changes the direction of the vector in the space. We can also see that in the second half of the demonstration, we get closer and closer to the recipe that was demonstrated. This is because indeed adding each piece of apple is slowly moving the vector direction towards the direction of the vector of recipe number one.

D. LEARNING NEW RECIPES

We now present a scenario of learning a new recipe. For this purpose, we use a recipe for carrot orange slaw. This recipe contains 2 pieces of orange and 2 pieces of carrot. This recipe is significantly different from all recipes in the initial cookbook. We have made 2 demonstrations for this recipe with ingredients added in a different order. The first demonstration was used as a presentation of a novel recipe that the robot needs to recognize and learn. We can see the evolution of this demonstration similar to the other recipes in Figure 7 d. We can clearly see that the demonstration is not very similar to any of the recipes. Therefore, the system adds this new recipe to the cookbook, with the number of ingredients extracted from the video. Then we use the second demonstration to see the reaction of the robot to this recipe after it was learned. The evolution of the cosine similarity is shown in Figure 7 e. We clearly see that this demonstration was recognized as the new recipe.

E. ROBOTIC SALAD MAKING

Finally, we report on the robot's cooking performance across multiple experiments. We used 16 demonstrations, consisting of two salads of each recipe, where the order of ingredients in one of them was the reverse of the other one. Markov model needs to be trained, therefore we split the data into two groups to do two-fold validation. Each of the demonstrations is run through computer vision, and the robot cooks a salad according to each of the recipes. The results of these experiments are shown in Table 2, where we report the number of ingredients detected in the recipe for both variants of each recipe demonstrated, as well as the number of ingredients added by the robot when assembling the salad. Comparing these results to the recipe list we can see that we have had a few salads where the number of ingredients extracted was not correct. This error happened in one of the demonstrations of recipe 1 where one of the ingredients was not detected. The recipe that proved especially hard for the detection was recipe 5, where there were errors in both variations of demonstrations. Those errors were both double detections of an ingredient and some ingredients were also missed. Out of those three demonstrations that contained errors only one was classified as a wrong recipe. This was the case for recipe one, likely due to the recipe containing only two ingredients, which made missing the detection of one of those a relatively big error. This makes the system recognize all other ingredients correctly in 83% of demonstrations, and predict the intention of the human cook in 94% of cases. The salad-making task proved feasible and we assembled all of the salad successfully.

IV. CONCLUSION AND FUTURE WORK

The paper has shown a system that can recognise human intention and produce food according to it using a mixture of human demonstration and previous knowledge (recipes). It can also incrementally learn new recipes from human demonstration, effectively making its recipe base larger. The robot has done it on a rather limited scale and with a limited variety of dishes. These are the bottlenecks that need to be overcome to make it more useful practically. The major

limiting factor was the number of classes of objects that we were able to detect. This is mainly due to the nature of the data set the neural network used was trained on. The COCO data set was used here, which had over 50 classes, but only 10 of them were related to food, and an even lower number of them were potential salad ingredients. In reality, kitchens contain many more kinds of utensils, and recipes contain many more types of ingredients. We are sure that this bottleneck could be overcome with the current technology as we already use a network that detects many more classes than we use. The problem was that most of these classes are not relevant to cooking. We propose that this problem can be solved by producing data sets specific to the robot cooking applications and retraining the current neural networks with this data set. Possibly, parts of pre-existing datasets could be merged to make such a dataset.

Moreover, we believe that the action identification pipeline can be further improved. So far we have used a mixture of hard coding, feature extraction and machine learning to build the pipeline. While the pipeline was successful in the laboratory environment, we believe that in the future the whole system should be put into one, single machine-learning model that is based on end-to-end learning by neural networks. Such an approach is unfortunately out of the scope of the current work due to the large amount of data that would need to be gathered. This could bring a lot of benefits for example the Pearson correlation used to identify the grasp item could be replaced by a feature learned by the convolutional neural network, and those tend to perform better than human-chosen features. The neural network could also improve the filtering that we have done with the hidden Markov model as they can find more complicated patterns and model the influence of a large number of past states. All of those are again conditioned by our ability to gather a large robotic cooking-specific data set. Automatic generation of such data could be one possible direction to move robotic cooking forward. The large-scale experimentation approach is one of the possibilities. Therefore we propose a setup that produces cooking-specific data automatically as the future direction. We do not yet specify which information is necessary and most useful for robotic cooking and we leave that decision to future research. Few likely candidates are videos both in the visible and infrared spectrum of the robot cooking, the text of the cookbooks, but also less known information for example taste-like sensing.

Finally, we would like to add a few words about the manipulation ability of the setup used and the path to economic viability. Robotic control and manipulation were not the focus of this work and can be further improved if needed. Future work should make the setup more robust and above all cheaper. We have used a general-purpose robotic arm, that was much more capable than needed, but this setup could be done with cheaper more task-specific robots. In this case, a big portion of the deployment costs will be coming from the programming of the robots. This is where incremental learning from demonstration comes into the picture and makes programming the setup for a specific task much faster and easier.

ACKNOWLEDGMENT

REFERENCES

[1] R. Erlich, A. Yngve, and M. L. Wahlqvist, "Cooking as a healthy behaviour," *Public Health Nutrition*, vol. 15, no. 7, pp. 1139–1140, Jul. 2012.

[2] L. P. Smith, S. W. Ng, and B. M. Popkin, "Trends in U.S. home food preparation and consumption: Analysis of national nutrition surveys and time use studies from 1965–1966 to 2007–2008," *Nutrition J.*, vol. 12, no. 1, pp. 1–10, Dec. 2013.

[3] N. I. Larson, C. L. Perry, M. Story, and D. Neumark-Sztainer, "Food preparation by young adults is associated with better diet quality," *J. Amer. Dietetic Assoc.*, vol. 106, no. 12, pp. 2001–2007, Dec. 2006.

[4] Y. Lin, A. S. Wang, E. Undersander, and A. Rai, "Efficient and interpretable robot manipulation with graph neural networks," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 2740–2747, Apr. 2022.

[5] T. H. Davenport and S. M. Miller, *Fast Food Hamburger Outlets: Flippy— Robotic Assistants for Fast Food Preparation*, 2022, pp. 147–150.

[6] F. Mauch, A. Roennau, G. Heppner, T. Buettner, and R. Dillmann, "Service robots in the field: The BratWurst bot," in *Proc. 18th Int. Conf. Adv. Robot. (ICAR)*, Jul. 2017, pp. 13–19.

[7] M. Barakazi, "The use of robotics in the kitchens of the future: The example of Moley robotics," *J. Tourism Gastronomy Stud.*, vol. 10, pp. 895–905, Jun. 2022.

[8] F. Fusté-Forné, "Robot chefs in gastronomy tourism: What's on the menu?" *Tourism Manage. Perspect.*, vol. 37, Jan. 2021, Art. no. 100774.

[9] J. Hughes, L. Scimeca, I. Ifrim, P. Maiolino, and F. Iida, "Achieving robotically peeled lettuce," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4337–4342, Oct. 2018.

[10] I. Mitsioni, Y. Karayiannidis, and D. Kragic, "Modelling and learning dynamics for robotic food-cutting," in *Proc. IEEE 17th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2021, pp. 1194–1200.

[11] J. Liu, Y. Chen, Z. Dong, S. Wang, S. Calinon, M. Li, and F. Chen, "Robot cooking with stir-fry: Bimanual non-prehensile manipulation of semi-fluid objects," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 5159–5166, Apr. 2022.

[12] B. Cobley and D. Boyle, "OnionBot: A system for collaborative computational cooking," 2020, *arXiv:2011.05039*.

[13] E. Almanzor, T. G. Thuruthel, and F. Iida, "Automated fruit quality testing using an electrical impedance tomography-enabled soft robotic gripper," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 8500–8506.

[14] L. Scimeca, P. Maiolino, D. Cardin-Catalan, A. P. d. Pobil, A. Morales, and F. Iida, "Non-destructive robotic assessment of mango ripeness via multi-point soft haptics," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 1821–1826.

[15] G. Sochacki, J. Hughes, and F. Iida, "Sensorized compliant robot gripper for estimating the cooking time of boil-cooked vegetables," in *Proc. Int. Conf. Intell. Auton. Syst.* Cham, Switzerland: Springer, 2022, pp. 227–238.

[16] G. Sochacki, J. Hughes, S. Hauser, and F. Iida, "Closed-loop robotic cooking of scrambled eggs with a salinity-based 'taste' sensor," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 594–600.

[17] Q. Ouyang, J. Zhao, Q. Chen, and H. Lin, "Classification of Rice wine according to different marked ages using a novel artificial olfactory technique based on colorimetric sensor array," *Food Chem.*, vol. 138, nos. 2–3, pp. 1320–1324, Jun. 2013.

[18] X. Tian, J. Wang, Z. Ma, M. Li, and Z. Wei, "Combination of an E-nose and an E-tongue for adulteration detection of minced mutton mixed with pork," *J. Food Qual.*, vol. 2019, pp. 1–10, Apr. 2019.

[19] J. W. Gardner, H. V. Shurmer, and T. T. Tan, "Application of an electronic nose to the discrimination of coffees," *Sens. Actuators B, Chem.*, vol. 6, nos. 1–3, pp. 71–75, Jan. 1992.

[20] N. Valente, A. Rudnitskaya, J. Oliveira, E. Gaspar, and M. Gomes, "Cheeses made from raw and pasteurized cow's milk analysed by an electronic nose and an electronic tongue," *Sensors*, vol. 18, no. 8, p. 2415, Jul. 2018.
[21] L. Gil, J. M. Barat, E. Garcia-Breijo, J. Ibañez, R. Martínez-Máñez, J. Soto, E. Llobet, J. Brezmes, M.-C. Aristoy, and F. Toldrá, "Fish freshness analysis using metallic potentiometric electrodes," *Sens. Actuators B, Chem.*, vol. 131, no. 2, pp. 362–370, May 2008.
[22] D. Danno, S. Hauser, and F. Iida, "Robotic cooking through pose extraction from human natural cooking using openpose," in *Proc. Int. Conf. Intell. Auton. Syst.* Cham, Switzerland: Springer, 2022, pp. 288–298.
[23] G. Sochacki, A. Abdulali, and F. Iida, "Mastication-enhanced taste-based classification of multi-ingredient dishes for robotic cooking," *Frontiers Robot. AI*, vol. 9, p. 108, May 2022.
[24] K. Junge, J. Hughes, T. G. Thuruthel, and F. Iida, "Improving robotic cooking using batch Bayesian optimization," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 760–765, Apr. 2020.
[25] M. S. Sakib, D. Paulius, and Y. Sun, "Approximate task tree retrieval in a knowledge network for robotic cooking," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 11492–11499, Oct. 2022.
[26] M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus, "Interpreting and executing recipes with a cooking robot," in *Experimental Robotics*. Cham, Switzerland: Springer, 2013, pp. 481–495.
[27] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.
[29] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.
[30] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "You only learn one representation: Unified network for multiple tasks," 2021, *arXiv:2105.04206*.
[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Computer Vision ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham, Switzerland: Springer, 2014, pp. 740–755.
[32] Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh, "OpenPose: Realtime multi-person 2D pose estimation using part affinity fields," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 1, pp. 172–186, Jan. 2021.
[33] Z. Liu, H. Chen, R. Feng, S. Wu, S. Ji, B. Yang, and X. Wang, "Deep dual consecutive network for human pose estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 525–534.
[34] I. K. R. A. Guler and N. Neverova, "Densepose: Dense human pose estimation in the wild," Tech. Rep., 2018.
[35] B. Cheng, B. Xiao, J. Wang, H. Shi, T. S. Huang, and L. Zhang, "HigherHRNet: Scale-aware representation learning for bottom-up human pose estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 5385–5394.
[36] X. Rong, "Word2vec parameter learning explained," 2014, *arXiv:1411.2738*.
[37] L. Ma and Y. Zhang, "Using Word2 Vec to process big text data," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 2895–2897.
[38] D. Jatnika, M. A. Bijaksana, and A. A. Suryani, "Word2 Vec model analysis for semantic similarities in english words," *Proc. Comput. Sci.*, vol. 157, pp. 160–167, Jan. 2019.

GRZEGORZ SOCHACKI received the bachelor's and master's degrees in electronics and electrical engineering from The University of Edinburgh, in 2019, and the master's degree in robotics from the University of Lincoln, in 2020. He is currently pursuing the Ph.D. degree in robotics with the University of Cambridge.

His research interests include electronic taste, robotic chefs, machine learning, and signal processing.

ARSEN ABDULALI (Member, IEEE) received the B.S. degree in information security from the Tashkent University of Information Technologies, Tashkent, Uzbekistan, in 2014, and the Ph.D. degree from the Department of Computer Science and Engineering, Kyung Hee University, Seoul, South Korea. During the Ph.D. degree, he developed a unified framework that enables haptic interaction with virtual objects. The framework considers both surface properties by modeling vibrotactile feedback induced by the haptic texture and material properties that they experience through the deformation of an object. In 2021, he joined the Bio-Inspired Robotics Laboratory, Department of Engineering, University of Cambridge, as a Research Associate. His research interests include modeling haptic interaction for the robotics and virtual and augmented realities. In 2022, he was awarded a three-year Marie Skdowska-Curie Fellowship.

NARGES KHADEM HOSSEINI (Member, IEEE) received the B.S. degree in electrical engineering from Technical and Vocational University, Tehran, Iran, in 2014, and the M.S. degree in control signaling from the Iran University of Science and Technology, Tehran, in 2016. She is currently a Researcher in robotics with the Department of Engineering, University of Cambridge, Cambridge, U.K. Her research interests include optimization techniques and applications and machine learning and intelligent systems, such as neural networks and fuzzy systems applied in control systems.

FUMIYA IIDA (Senior Member, IEEE) received the bachelor's and master's degrees in mechanical engineering from the Tokyo University of Science, Japan, in 1999, and the Dr. sc. nat. degree in informatics from the University of Zurich, in 2006. He is currently a Professor in robotics with the Department of Engineering, University of Cambridge. From 2006 to 2009, he was a Postdoctoral Associate with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA. In 2006, he was awarded the Fellowship for Prospective Researchers from the Swiss National Science Foundation, and in 2009, he was appointed as a Swiss National Science Foundation Professor for Bio-Inspired Robotics with ETH Zurich. He has been involved in a number of research projects related to dynamic legged locomotion, navigation of autonomous robots, and human–machine interactions. He has published over 40 publications in major robotics journals and conferences and edited two books. His research interests include biologically inspired robotics, embodied artificial intelligence, and biomechanics. Currently, he serves on the editorial board for the *Soft Robotics* journal and *Frontiers in Robotics and AI* (Bio-Inspired Robotics Section) and a program committee member for international conferences and workshops. In addition, he has organized a few seminal meetings, such as the International Conference of Embodied Intelligence, RoboSoft, and TAROS.