2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Context-aware robot control using gesture episodes

Petr Vanc[1] Jan Kristof Behrens[1] Karla Stepanova[1]

***Abstract*— Collaborative robots became a popular tool for increasing productivity in partly automated manufacturing plants. Intuitive robot teaching methods are required to quickly and flexibly adapt the robot programs to new tasks. Gestures have an essential role in human communication. However, in human-robot-interaction scenarios, gesture-based user interfaces are so far used rarely, and if they employ a one-to-one mapping of gestures to robot control variables. In this paper, we propose a method that infers the user's intent based on gesture episodes, the context of the situation, and common sense. The approach is evaluated in a simulated table-top manipulation setting. We conduct deterministic experiments with simulated users and show that the system can even handle the personal preferences of each user.**

I. Introduction

Robots are complex machines with unintuitive kinematics and operational constraints. When a worker wants to instruct a robot to do a set of manipulations, it is more natural to convey the high-level manipulation goal (intent) than to program low-level robot actuation. Gestures are a natural part of human communication and have been used to communicate with robots since the late 1990s when the recognition of six different arm gestures was used to control a wheeled robot [1]. In contrast to the natural use of gestures in social situations, gestures as part of a user interface (UI) are considered *functional* [2]. Most of the current works focus on 1-to-1 mapping between gestures and controlled variables (e.g., [3], [4], [5]), thus heavily restricting the expressiveness of the gestures.

This paper proposes a system that allows the user to control a robot by communicating target actions, objects, and other parameters via gestures. Simultaneously, it leaves the robot the autonomy to execute the high-level intent by a suitable set of low-level robot actions, e.g., generated by a behavior tree [6]. We utilize the Leap Motion sensor [7] to detect and track the user's hand-bone structure. We do not impose any assumptions on hand gestures or their exact meaning, enabling the user to define new gestures via demonstration. The detected gestures (expressing a human intent) are fed to a probabilistic neural network for classification (see Fig.3). The use of the gestures might vary from user to user and based on the scene or manipulated object. For example, a gesture to open a drawer might be a pulling motion, while the opening of a box will rather be a lifting motion. We show the importance of including the context of the situation when inferring the user's intents on a set of artificial datasets.

[1]Czech Technical University in Prague, Czech Institute of Informatics, Robotics, and Cybernetics, petr.vanc@cvut.cz, jan.kristof.behrens@cvut.cz, karla.stepanova@cvut.cz

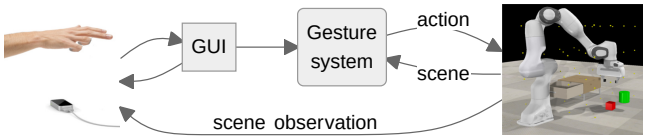

Fig. 1: Proposed Human-Robot interaction and Experimental setup. The gesture system block represents our proposed setup (Fig. 2). On the left side is Leap Motion Sensor [7] for hand tracking. On the right side is a Robot setup with Panda Manipulator with an initialized scene. The yellow dots show the robot scene grid.

We demonstrate our system in a simulated table-top manipulation scenario with a 7DoF robot manipulator (see Fig.5). Let us assume the user intends to tidy up a cup in a drawer. She focuses her eyes on the drawer, then makes a gesture *swipe down* to signal the object's placement. In this way, the intent is assembled as the accumulated desired change in the world state. The system accumulates all gesture detection data over the episode and compresses it into an observation vector. This observation vector and the world state are then used to infer the user's intent.

The contributions of this paper are

- a flexible gesture recognition framework that also supports gesture combinations (for improved expressiveness) and context-aware gesture interpretation,
- an approach to learn such a context-aware gesture-intent mapping from sampled data, and
- a system implementation that grants the robot more autonomy and thereby increases the robustness of the gesture control system (see code and data at github.com/imitrob/context-based-gesture-operation)

II. Related Work

Based on the taxonomy introduced in the survey paper [2], we focus on functional static and dynamic manipulative and control hand gestures. The proposed system enables the definition and usage of both semaphoric gestures associated with a command and manipulative gestures that map the gesture movement to the location and pose of the object.

Many works focus on developing better sensors enabling gesture recognition such as wearable [8] or contact-less sensors [9]. The development of better methods to detect individual gestures [3], [4] and human activities provides important key components for HRI systems. Several papers utilize the Leap Motion controller [10] to detect hand gestures. Methods to learn and detect the gestures include deterministic learning [11], support vector machines (SVM)

with custom combinations of features [12], or recurrent neural networks (RNN) [13]. [14] introduces hand gesture recognition using a Leap Motion sensor and Kinect depth camera. These works typically focus on 1-to-1 mapping between gestures and controlled variable [3]. Instead, we aim for more robot autonomy and a more expressive gesture language through a combination of context-dependent gestures.

Designing a good UI is an art that is approached very systematically in [3], where arm gestures detected by an inertial sensor are used to control a robot's functional states (e.g., off, idle, and moving) as well as the specific motion when the robot is moving. While we are not concentrating on making a single system as user-friendly as possible, we introduce concepts that let users operate systems in a personalized manner.

[15] introduces a system that detects gestures in real-time from a Kinect RGBD camera using ad-hoc Hidden Markov Models. Our system works as well in real-time. Our gesture detection using probabilistic neural networks also needs only a few sample demonstrations to learn new gestures. In contrast, we deal with a more complex HRI use case where the robot has more autonomy. [16] use Kinect RGBD camera and OpenNI to extract joint angles as features. A neural network classifier allows real-time gesture detection with a sliding window approach. However, only three joint angles were considered in the feature vector, and gestures have a fixed length of 2 seconds (60 frames). In this work, we represent dynamic gestures as Probabilistic Motion Primitives [17] and compare new data via dynamic time warping against the library of dynamic gestures. We also add context and user-dependent gesture selection on top.

III. Problem formalization

Taking into account the current state of the world, the user expresses his desired intent (the desired change in the world) by hand and eye movements (i.e., gesture and region of interest (ROI) can be observed). The gesture-based system should be able given the observation $\mathbf{o}$ to determine the user's intent $\mathbf{i}$ and transform it to a sequence of robot actions $\mathbf{a}$. In other words, the task is to learn from a set of observations the mapping $\mathcal{M}$ between the observations and the intents and the mapping $\mathcal{A}$ between the intent $\mathbf{i}$ and the robotic action sequence $\mathbf{a}$. Note that individual users might use different gestures in different contexts. Therefore the mapping has to take into account all the observation variables.

Observation tuple $\mathbf{o} = [\mathbf{h}, \mathbf{f}, \mathbf{s}, u]$ contains time series of hand features $\mathbf{h}$ and ROI $\mathbf{f}$, accompanied by the context of the situation – i.e., the scene description $\mathbf{s}$ and the user description u. *Scene* $\mathbf{s}$ expresses the state of the system. It includes information about the position, type, and state of the objects in the scene, the position and state of the end-effector, and possibly other features. *User description* can be just identification of the user or additional parameters describing the user (e.g., emotional state, nationality, etc.).

Given the time series of *hand movement features* $\mathbf{h}$, set of *available gestures* $\mathbf{G}$ and the trained *classifier* $\mathcal{G}$, the probability of individual *gestures* $\mathbf{g}$ can be determined:

$$\mathbf{g} = \mathcal{G}(\mathbf{h}, \mathbf{G}) \tag{1}$$

In this regard, the gesture is defined as a specific pattern performed by a user to affect the behavior of an intelligent system (see [2]) – we distinguish static (pose-based) or dynamic (motion-based) gestures. Each gesture is expressed by a model trained using a set of demonstrations. The gesture set can be arbitrarily extended. Note that each user might train their own set of gestures or use the general gestures available in the system.

We define *intent* in the given context $\mathbf{i}$ as a tuple of three variables: target action ta, target object to, and target metric tm (selected based on their probability).

$$\mathbf{i} = (\underbrace{argmax(\mathbf{i}_{ta}^{probs})}_{ta}, \underbrace{argmax(\mathbf{i}_{to}^{probs})}_{to}, ap). \tag{2}$$

The probabilities of the intended actions $\mathbf{i}_{ta}^{probs}$ and target objects $\mathbf{i}_{to}^{probs}$ in the given context are determined by applying a mapping on observation tuple: $\mathbf{i}_{ta}^{probs} = \mathcal{M}_a(\mathbf{o})$, $\mathbf{i}_{to}^{probs} = \mathcal{M}_o(\mathbf{o})$. Target action ta represents the action that the user wants to perform (e.g., open, push, move into, etc.), and target object to denotes an object instance from the scene to act on. Auxiliary parameters ap represents optional information about user intent. Generally, they are in form of a vector of metric parameters for given action. For example, a user may specify the robot speed rate by the system extracting the hand distance between the thumb and the pointing finger.

From the intent $\mathbf{i}$, the *robotic action sequence* $\mathbf{a}$ with specified parameters is computed, taking into account the set of actions within the action space A: $\mathbf{a} = \mathcal{A}(\mathbf{i})$. We consider robotic actions in this regard as purposeful manipulation of the world by the robot. A sequence of actions is needed when the target action has unsatisfied preconditions. $\mathcal{A}$ can be implemented in general using task planning, but we opted for a solution based on a behavior tree that handles the execution reactively, including error recovery strategies. We consider the following three cases. The intent can either correspond to 1) a single robot action (e.g., pour water from the cup); 2) a single robot action with specific parameters (e.g., rotating the cup by 5 degrees); or 3) a sequence of robotic actions (e.g., put the cup to the drawer corresponds to a sequence of actions: get close to the drawer, open gripper, move gripper away).

IV. Materials and Methods

The proposed system can be divided into three parts. The first handles the real-time gesture recognition task $\mathcal{G}$ (Fig.: $2^{\text{upper}}_{\text{part}}$) from hand observations $\mathbf{h}$, the second uses the gesture classification vector $\mathbf{g}$ and infers the user's intent $\mathbf{i}$ (using mapping $\mathcal{M}$), and finally the third part realizes robotic action sequence $\mathbf{a}$ for the given intent.

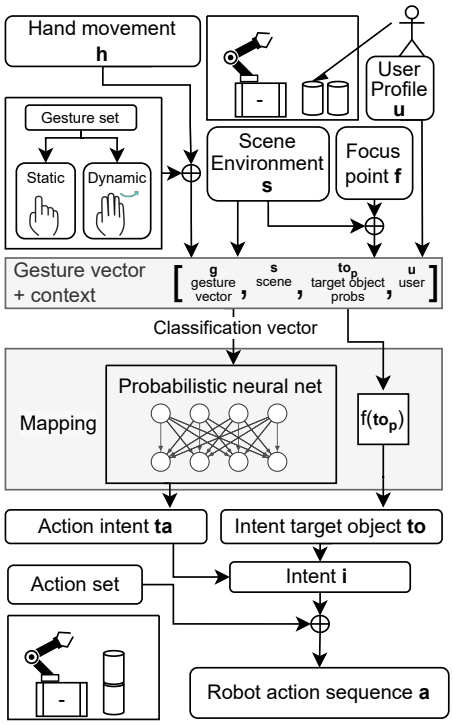

Fig. 2: The diagram of the proposed system shows the whole pipeline from hand observations to robotic actions.

A. Gesture classification

We distinguish static and dynamic gestures in the proposed setup, classified by separate classifiers. A set of features describing the hand movement are computed from the data acquired from the Leap Motion sensor [7] (i.e., hand bone structure and hand position).

For the gesture classification task (see Eq. 1) we utilize our teleoperation gesture toolbox (see `github.com/imitrob/teleop_gesture_toolbox`, [5]). The gesture sets (static and dynamic variants) can be adapted to the use case and user preferences, including learning new gestures directly from the user or adjusting the mapping between the existing gestures and the desired action. The system enables one to set the mapping between gestures and actions directly in GUI or learn it from observations.

1) Static gestures: Static gestures have a constant stroke phase. The following hand-crafted feature set is analyzed in every frame: 1) distances between fingertip positions $(\mathbf{x}_{f1}, \ldots, \mathbf{x}_{f5})$ (including the distances to the palm center $(\mathbf{x}_{palm})$), and 2) joint angles between hand bones $(\alpha_1, \ldots, \alpha_N)$ that are computed as angles between direction vectors (obtained from differences between individual bone positions). A total amount of 57 features are used:

$$\mathbf{h} = [\alpha_1, ..., \alpha_{42}, ||\mathbf{x}_{f1}, \mathbf{x}_{palm}||, ||\mathbf{x}_{f1}, \mathbf{x}_{f2}||, \ldots, ||\mathbf{x}_{f4}, \mathbf{x}_{f5}||] \tag{3}$$

The most accurate detection method for static gestures has proven to be the probabilistic neural network model (see Sec. IV-C). Other tested methods included deterministic neural networks and a hand-crafted approach based on hand-picked thresholds. We made empirical and classification tests for various combinations of the features. In our setup, up to 10 static gestures were considered.

2) Dynamic gestures: Compared to static gestures, dynamic gestures are characterized by a movement, i.e., the stroke phase is a trajectory. For simplicity, we represent in our experiments dynamic gestures only by the Cartesian position of the palm center in each time step i ($i \in \{1, \ldots, N\}$):

$$\mathbf{h} = [x_1, y_1, z_1, \ldots, x_N, y_N, z_N] \tag{4}$$

The used Leap Motion sensor records with up to 100 Hz. According to [18] it is possible to reduce measurement frequency to $f = 20\,\text{Hz}$ without any significant loss.

Each dynamic gesture is learned from a set of observations and represented as a probabilistic motion primitive [17]. For the classification task, the Dynamic Time Warping (DTW) [19] [20] is utilized, as it can deal with different motion speeds and time deformations and is very computational efficient (see [5] for the details of the method).

B. Episode

We consider an *episode* as the time window in which the user specifies the intent using one or more gestures. In this paper, we assume that an episode begins with the detection of hands and ends when they disappear. To improve the ergonomy and stability of the system, we run the gesture recognizers on the movements (so-called *stroke*) that started in the center of the detection area. This allows the user to always return to the center for the next gesture without accidentally triggering other gestures and avoid undesired *preparation* and *retraction* moves. The notion of episode enables us to collect observations over time until we accumulate enough evidence to infer an actionable intent.

The data are continuously fed to the circular buffer for the gesture recognition task. The parallel recognition of static and dynamic gestures is running (see Sec. IV-A), accumulating the evidence for individual gestures. Therefore, a set of gestures passing the evidence threshold E_t might be recognized within one episode (see sample episode evaluation Fig.: 6, where gestures "point" and "swipe down" were recognized). Anytime one of the gestures passes the evidence threshold, the probability vector for all gestures is saved. The set of the saved probability vectors within one episode is combined into one gesture vector $\mathbf{g}$, that gives information about the maximum probabilities of each gesture in the given episode:

$$\mathbf{g} = max_i \left(\begin{bmatrix} g_{1,1} \\ g_{2,1} \\ \ldots \\ g_{G,1} \end{bmatrix}, \ldots, \begin{bmatrix} g_{1,i} \\ g_{2,i} \\ \ldots \\ g_{G,i} \end{bmatrix}, \ldots, \begin{bmatrix} g_{1,C} \\ g_{2,C} \\ \ldots \\ g_{G,C} \end{bmatrix} \right) \tag{5}$$

where G is the number of considered gestures, and C is the number of triggered gesture observations in the given episode. The gesture vector $\mathbf{g}$ is one of the inputs to the intent recognition system.

C. Mapping represented by Probabilistic Neural Network

We approach the problem of inferring user intent **i** for a given set of gestures **g** as a classification problem. We train a probabilistic neural network (1 to 3 fully connected hidden layers) in a supervised manner using data from sampled scenes with generated intents and gestures. To reflect the context-dependent usage of the gestures, we take as an input to the network a combination of gesture vector, user id, ROI, and scene object state and type (see Fig. 3). The output is a categorical distribution with probability density function $f(i|\mathbf{p}) = p_i, \forall$ intents i, $\mathbf{p}$ is a vector of probabilities of individual intents ($\sum_i p_i = 1$, $p_i > 0 \forall i$).

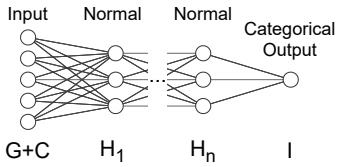

Fig. 3: Graphical model of Probabilistic feed-forward neural network for the mapping task $\mathcal{M}$. (Gestures + Context $\rightarrow$ Intent) The input is comprised of a Gesture vector with size G and context features with size C (e.g. objects, object states, user, etc.), Output is user intent probabilities with size I. *Normal* and *Categorical* tags represent type of probability distribution. *Categorical* type represents discrete probabilistic distribution (see Sec.: IV-C). H_1 to H_n is number of nodes in the i-th hidden layer.

The forward loop of the network, used in experiments, can be written as:

$$Y = sigmoid(\tanh(\mathbf{X} \cdot \mathbf{w}_1) \cdot \mathbf{w}_2), \tag{6}$$

where $\mathbf{X}$ is input observations vector, $\mathbf{w}_1$ and $\mathbf{w}_2$ are classification weights of the length $O \times H_1$ and $H_2 \times I$, respectively. We picked hidden layer sizes as $H_1 = 25, H_2 = 25$. O is the length of observation vector (gesture vector + context) $O = G + C$, I is the amount of possible gestures/intents. The $\mathbf{Y}$ is a vector of I intent/gesture probabilities. For the classification result, the selected class is chosen as the one with the maximum probability $c = \text{argmax}_I \mathbf{Y}$.

The automatic differentiation variational inference (ADVI) is used as a fitting algorithm [21]. Kullback-Leibler divergence is used as a loss function. The function callback is called every 2000th epoch and makes a few samples drawn on the test data. In this way, we can estimate the local maximum and avoid overfitting. The network is built on PyMC framework [22].

D. Robotic action sequence generation

The generation of robotic motions for intent in the form of a *target action*, a *target objects*, and (optionally) a set of auxiliary parameters utilizes a behavior tree [6]. The behavior tree encodes a high-level policy and is parameterized by the intent as a goal. An intent *(put into, drawer)* while holding a *cup* requires opening the drawer (if it is closed) and placing the cup inside the drawer. The behavior tree automatically triggers the opening action that the users rarely command. Hence, it is necessary to deal with incomplete user specifications. Fig. 4 shows a visual representation of a behavior tree for our example domain.

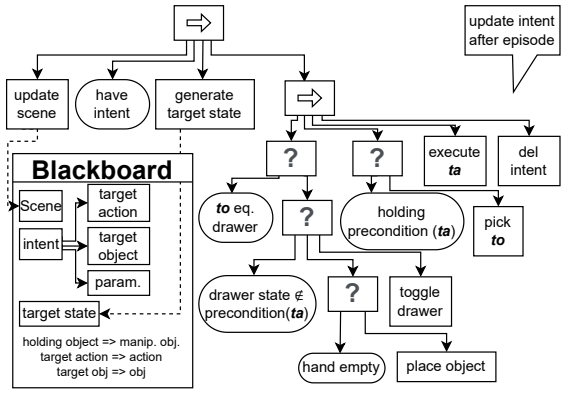

Fig. 4: Construction of action sequence **a** from intent **i** using a Behavior tree approach. See Sec. IV-D.

V. Experimental setup and dataset generation

A. Robotic environment and gesture classification

We evaluate the proposed method using a simulated setup in the simulator Coppelia Sim [23] and a build tool PyRep [24]. The scenes consist of a Franka Emika Panda with 7 DOF, cups, drawers, and cubes. The robot can open and close the drawer and manipulate the other objects. Furthermore, cups and cubes can be stacked on cubes and placed inside drawers. Cups also can contain liquid that can be poured into other containers. See the experimental setup in Fig.: 1.

B. Gesture classification framework

To classify the raw hand observations from users in the actual HRI case as gestures and compress them into the gesture vector, we utilize our *Toolbox for gesture operation of the robot* [5]. The toolbox segments the episodes based on hand visibility. The classification is a signal per tracked gesture over time proportional to the probability that the gesture was presented at that point in time (see Fig. 6).

C. Artificial datasets generation

We create artificial datasets with different levels of context-dependency of the gestures. These well-controlled datasets that simulate various users interacting with the system in different conditions enable us to evaluate which input data are necessary to consider to learn the mapping between the gestures and the intents.

Each dataset comprises a set of observation points $\mathbf{o}_i$:

$$\begin{aligned} \mathbf{D} &= \{\mathbf{o}_i\}, \\ \mathbf{o}_i &= \left[\mathbf{g}_j, \mathbf{s}_i, \mathbf{f}_i, u_i\right], \end{aligned} \tag{7}$$

where $\mathbf{g}$ is a gesture observations vector, $\mathbf{s}$ is a scene description (object states, positions), $\mathbf{f}$ is a ROI, and u is the identification of a user.

Each observation is generated as follows: 1. a random scene generation, 2. one of the possible intent actions and target objects are selected, 3. a focus point is generated, 4. the user is selected, and 5. a gesture vector is generated. The individual steps are described in detail in the following subsections.

1) Scene generation: To sample random valid scenes, we first randomly select the number n of objects in the scene. To achieve useful combinations of scene objects, we instantiate the n leading objects in the vector

$$\mathbf{T} = [\text{cup, drawer, box, cup, drawer, box, cup}].$$

Then we place each object randomly in a $4 \times 4 \times 4$ grid in front of the robot (all grid positions are reachable for the robot). Real-world behavior is simulated by adding a random number to the default grid position. First, drawers are placed, then cubes, and finally cups. Objects landing in the same vertical column are stacked, if possible, or replaced. Then each object pose is slightly perturbed. Finally, we randomly select the value of each object attribute, e.g., open/close drawer, full/empty cup, gripper full/empty. Any state is not binary but given as a unit interval number to simulate the real world. In the case of a full gripper, the pose of the assigned object is changed to the gripper location. Three sample-generated random scenes are visualized in Fig.: 5.

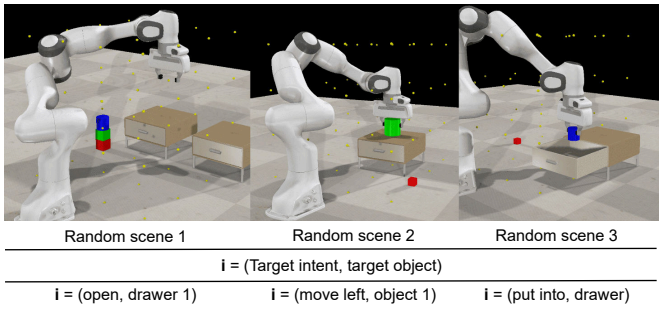

Fig. 5: Random scene example generation.

2) Generating Intents and Action sets: For every scene, we generate a set of suitable intents. The set of valid intents is a subset of the Cartesian product of the scene objects $\times$ the target actions. $\text{Pre}(t_A)$ denotes the set of preconditions of the target action t_A that must hold for execution. For invalid intents holds

$$\bigcap \text{Pre}(t_A) \in \text{scene} \neq true.$$

Such intents are ignored. For example, the target action *put into* is only applicable in scenes with an object in the gripper and an open drawer as the target object.

$$\begin{aligned} \text{Pre}(t_A) = \{&[\text{type}(t_O) = \textit{drawer}], \\ &[t_O.\text{open} = \textit{true}], \\ &[\text{gripper.holding} = \textit{true}]\} \end{aligned} \tag{8}$$

For *pick up* only *cubes* and *cups* can be selected as target objects and the gripper has to be empty. In general, intents denote a desired delta to the current scene state. Note that we do not require that all robotic motions to reach the intent are specified. We assume that the robot will have a sufficient level of autonomy to work toward these tiny goals.

The available intent action set in our setup includes *[put into, put on target, place, pour, pick up, open, close, ⟨ moves ⟩]*., moves:*[right, left, up, down]*. One or more parameters refine each intent. For example, *pour* is refined by an angle α that determines the final rotation of the object (default $\alpha = 90\,\text{deg}$).

3) Generation of the ROI: Given the selected target object, the ROI (f) is generated artifically using a normal distribution with the center in the target object.

4) Generation of the user: User u is randomly selected.

5) Generation of the gesture vector: The base for generating the gesture observations is a valid scene-intent-user combination. Simplified user models are used (inspired by a real human demonstrator performance within the given environment and the set of gestures). A multi-dimensional decision table represents the model. Each dimension represents the dependency on some context variable, such as the user, the scene, the intent, etc. For comparative studies, we prepared four datasets with a growing degree of differentiation in gesture generation. The four models (inspired by the selection of the gestures of a real demonstrator, see our website: github.com/imitrob/context-based-gesture-operation for exact values) are of the following dimensions:

- Dataset 1 $D1 : [I \times G]$,
- Dataset 2 $D2 : [T \times I \times G]$,
- Dataset 3 $D3 : [U \times T \times I \times G]$,
- Dataset 4 $D4 : [S \times U \times T \times I \times G]$,

where G is the number of gestures in the sample set (in our case 9), I is the number of intents in the sample set (in our case 11), T is the number of object types (in our case 3), U is the number of predefined users (in our case 2), and S is the number of object states (in our case 2).

Higher-dimensional models capture that humans use gestures in a context-dependent way. For example, one entry in Dataset 4 encodes which gesture user 1 would use in a situation when the target intent *put in* should be applied to the target object *drawer* in the state *open*. In this way, a set of different fake users was created.

D. Compared models

To show how the mapping between the context-dependent gestures and the intent can be learned from observations, we compare the following five models of probabilistic neural networks (all stemming from the general scheme shown in Fig. 3). Model M1 has one hidden layer. Other models have two hidden layers. The models differ in which input data they consider: *M1* (gesture vector), *M2* (gesture vector + dist.objects to ROI), *M3* (gesture vector + user id), *M4* (gesture vector + dist.objects to ROI + user id), *M5* (gesture vector + dist.objects to ROI + user id + objects' states).

The models M1-M5 were trained on the datasets D1-D4 (see Sec. V-C.5). Each dataset was split to training ($D_{train} = 4000$ samples) and testing ($D_{test} = 500$ samples) part. These were kept the same for all the evaluated models.

VI. Experimental Results

This section describes an evaluation of each part of the process in the pipeline.

1) Gesture classification results: When the set of gestures is chosen correctly, the gesture feature space is not overlaying. Detection is unambiguous. For the static gesture set consisting of 8 gestures (grab, pinch, point, two, three, four, five, thumbs up), we reached 99.1% balanced accuracy on the well-performed test data. The training dataset has around 12000 samples and 4000 for testing. The dynamic gesture detection approach based on the DTW was tested on a set of 5 directional swipe gestures and reached an overall balanced accuracy of 83%. Static and dynamic forms of gestures produce two independent streams of data. From one episode, we get a list of the few triggered gestures which fulfill the requirements. The reader can see single episode detection in Fig.: 6.

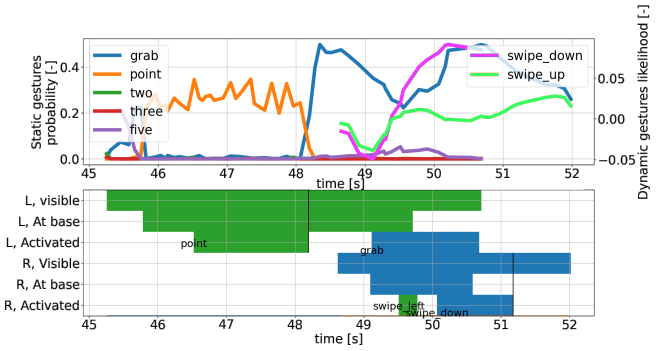

Fig. 6: The gesture detection output of a single episode. The upper plot shows the likelihoods for static (left legend) and dynamic (right legend) gestures. The bottom plot shows the visibility of the hands and the activation of the gestures. First was detected the static gesture point followed by grab and dynamic gesture swipe down.

2) Mapping results: We evaluated the models $M1$-$M5$ (see Sec. V-D) on datasets $D1$-$D4$ with gradually increasing context-dependency of the gestures (see Fig.: 7) using the independent test dataset. The simple model considering only gesture vectors can learn the one-to-one mapping well in the context-independent dataset D_1. For datasets where gestures are used based on the context, models that consider more information about the scene and the user excel. If a sufficiently complex model is used, accuracies over 97% are achieved on the testing set even for the most complex dataset. As a final experiment, the accuracy was measured concerning the number of samples needed for the network to learn the mapping for the dataset D4 well (see Fig.: 8). 2000-4000 samples are needed to reach convergence. After 300 samples, the accuracy improves significantly. In this case, we used the most context-dependent dataset $D4$, meaning that the system has to observe various situations to fully understand the gestures' context dependency.

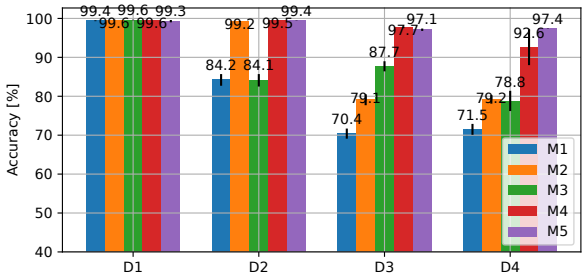

Fig. 7: Model comparison trained on the generated datasets. Independent samples on the test set obtained the measured, balanced accuracies. Models are defined in Sec.: V-D and datasets in Sec.: V-C.5.

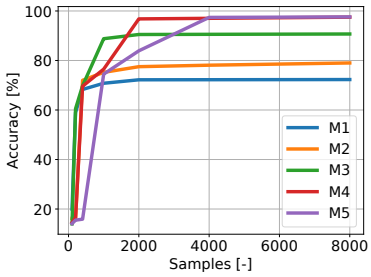

Fig. 8: Accuracy of models trained on the dataset $D4$.

VII. Conclusion and Discussion

In this paper, we proposed a system that uses functional gestures to operate robots. We show what the robot's intuitive, context-aware, and robust system for gesture operation might look like. This includes real-time detection of the gestures (modeled by probabilistic motion primitives) and the ability to extend the gesture set on the fly (using the Leap Motion sensor). Furthermore, such a system should be able to understand the human intent from the available observations of the hand movements (accumulating observations of the detected gestures over time) and the observed scene (e.g., objects on the scene and their state, state of the robot, user identification, etc.). Finally, we showed how to make the system robust and autonomous during the execution of the individual intents by utilizing behavior trees.

To simulate the interaction of various users with the system, we prepared artificial datasets in a simulated robotic environment with different levels of context-dependency of the gestures. We show that incorporating the data about the context into the probabilistic neural network enables the models to learn the mapping between the gestures and the intents even when starting from zero knowledge about the mapping.

VIII. Acknowledgment

This work was supported by the European Regional Development Fund under project Robotics for Industry 4.0 (reg. no. $CZ.02.1.01/0.0/0.0/15\_003/0000470$), MPO TRIO project num. FV40319, and by the Czech Science Foundation (project no. GA21-31000S). P.V. by CTU Student Grant Agency (reg. no. SGS23/138/OHK3-027/23).

REFERENCES

[1] D. Kortenkamp, E. Huber, R. P. Bonasso, *et al.*, "Recognizing and interpreting gestures on a mobile robot," in *Proceedings of the National Conference on Artificial Intelligence*, 1996, pp. 915–921.

[2] A. Carfì and F. Mastrogiovanni, "Gesture-based human-machine interaction: Taxonomy, problem definition, and analysis," *IEEE Transactions on Cybernetics*, 2021.

[3] E. Coronado, J. Villalobos, B. Bruno, and F. Mastrogiovanni, "Gesture-based robot control: Design challenges and evaluation with humans," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2761–2767.

[4] A. Carfi, C. Motolese, B. Bruno, and F. Mastrogiovanni, "Online human gesture recognition using recurrent neural networks and wearable sensors," in *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2018, pp. 188–195.

[5] P. Vanc, K. Stepanova, and J. K. Behrens, "Controlling robotic manipulations via bimanual gesture sequences," p. 2, 2022.

[6] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*, Jul 2018, arXiv:1709.00084 [cs]. [Online]. Available: http://arxiv.org/abs/1709.00084

[7] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller," *Sensors*, vol. 13, no. 55, p. 6380–6393, May 2013.

[8] M. Králik and M. Šuppa, "Waveglove: Transformer-based hand gesture recognition using multiple inertial sensors," in *2021 29th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 1576–1580.

[9] C. Stetco, S. Mühlbacher-Karrer, M. Lucchi, M. Weyrer, L.-M. Faller, and H. Zangl, "Gesture-based contactless control of mobile manipulators using capacitive sensing," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, 2020, pp. 1–6.

[10] [Online]. Available: https://www.ultraleap.com/product/leap-motion-controller/

[11] W. Zeng, C. Wang, and Q. Wang, "Hand gesture recognition using leap motion via deterministic learning," *Multimedia Tools and Applications*, vol. 77, no. 21, p. 28185–28206, Nov 2018.

[12] Y. Du, S. Liu, L. Feng, M. Chen, and J. Wu, "Hand gesture recognition with leap motion," no. arXiv:1711.04293, Nov 2017, arXiv:1711.04293 [cs]. [Online]. Available: http://arxiv.org/abs/1711.04293

[13] D. Avola, M. Bernardi, L. Cinque, G. L. Foresti, and C. Massaroni, "Exploiting recurrent neural networks and leap motion controller for the recognition of sign language and semaphoric hand gestures," *IEEE Transactions on Multimedia*, vol. 21, no. 1, p. 234–245, Jan 2019.

[14] G. Marin, "Hand gesture recognition with jointly calibrated leap motion and depth sensor," *Multimed Tools Appl*, p. 25, 2016.

[15] S. Iengo, S. Rossi, M. Staffa, and A. Finzi, "Continuous gesture recognition for flexible human-robot interaction," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, p. 4863–4868.

[16] G. Cicirelli, C. Attolico, C. Guaragnella, and T. D'Orazio, "A kinect-based gesture recognition approach for a natural human robot interface," *International Journal of Advanced Robotic Systems*, vol. 12, no. 3, p. 22, Mar 2015.

[17] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/hash/e53a0a2978c28872a4505bdb51db06dc-Abstract.html

[18] K. Forbes and E. Fiume, "An efficient search algorithm for motion data using weighted PCA," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05*. Los Angeles, California: ACM Press, 2005, p. 67. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1073368.1073377

[19] "Dynamic Time Warping," in *Information Retrieval for Music and Motion*, M. Müller, Ed. Berlin, Heidelberg: Springer, 2007, pp. 69–84. [Online]. Available: https://doi.org/10.1007/978-3-540-74048-3_4

[20] S. Salvador and P. Chan, "Fastdtw: Toward accurate dynamic time warping in linear time and space," p. 11.

[21] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei, "Automatic Differentiation Variational Inference," Mar. 2016, arXiv:1603.00788 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1603.00788

[22] D. Emaasit, "Pymc-learn: Practical Probabilistic Machine Learning in Python," Oct. 2018, arXiv:1811.00542 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1811.00542

[23] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 1321–1326, iSSN: 2153-0866.

[24] S. James, M. Freese, and A. J. Davison, "PyRep: Bringing V-REP to Deep Robot Learning," June 2019, arXiv:1906.11176 [cs]. [Online]. Available: http://arxiv.org/abs/1906.11176