Robotics and Autonomous Systems 170 (2023) 104556

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

A method for understanding and digitizing manipulation activities using programming by demonstration in robotic applications

Pablo Malvido Fresnillo [a,*], Saigopal Vasudevan [a], Wael M. Mohammed [a], Jose L. Martinez Lastra [a], José A. Pérez García [b]

[a] FAST-Lab Faculty of Engineering and Natural Sciences, Tampere University, Tampere, Finland
[b] Design and Fabrication in Industrial Engineering, University of Vigo, Vigo, Spain

ARTICLE INFO

Keywords:
Robot Programming by Demonstration
Motion intention recognition
Markov Models
Knowledge based systems
Automatic process segmentation

ABSTRACT

Robots are flexible machines, where the flexibility is achieved, mainly, by the re-programming of the robotic system. To fully exploit the potential of robotic systems, an easy, fast, and intuitive programming methodology is desired. By applying such methodology, robots will be open to a wider audience of potential users (i.e. SMEs, etc.) since the need for a robotic expert in charge of programming the robot will not be needed anymore. This paper presents a Programming by Demonstration approach dealing with high-level tasks taking advantage of the ROS standard. The system identifies the different processes associated to a single-arm human manipulation activity and generates an action plan for future interpretation by the robot. The system is composed of five modules, all of them containerized and interconnected by ROS. Three of these modules are in charge of processing the manipulation data gathered by the sensors system, and converting it from the lowest level to the highest manipulation processes. In order to do this transformation, a module is used to train the system. This module generates, for each operation, an Optimized Multiorder Multivariate Markov Model, that later will be used for the operations recognition and process segmentation. Finally, the fifth module is used to interface and calibrate the system. The system was implemented and tested using a dataglove and a hand position tracker to capture the operator's data during the manipulation. Four users and five different object types were used to train and test the system both for operations recognition and process segmentation and classification, including also the detection of the locations where the operations are performed.

1. Introduction

Humans have always attempted to find substitutes that would be able to mimic their behavior for performing certain tasks and interacting with the environment. This desire motivated the development of robots. The first industrial robots, which were introduced to the industry in the 1960s, were very bulky, and they were intended for the mass manufacturing of products of the same type [1]. Since then, the robotics field has experienced huge advances, such as better sensors [2], more advanced controllers, improvements in design and reduction of weight and size [1]; and, as a result, nowadays robots are present in almost every industrial sector [3,4]. Thanks to these advances robots have become significantly more versatile, with the ability to perform a wide array of tasks, unlike traditional/dedicated machinery like lathing or packaging machines. Consequently, robots can be repurposed for different processes and product models by simply adjusting their programs, saving on new machinery and allowing for quick adaptation to industry changes. However, to take full advantage of this versatility, some aspects must be considered: The reconfiguration of the robotic system should not require robotic expertise, as in most domains the end-users are not robotic engineers or software developers [5]; and its cost and time must be minimized. As a result, the development of new tools and systems that allow the easy, intuitive, and fast programming and reconfiguration of the robot has become remarkably important.

There are many different techniques for programming robots. Villani et al. classified them in four categories and compared them in terms of intuitiveness and ease of use, as well as thoroughness of possible operations [6]. The study concludes that, as the intuitiveness of the approach increases, the completeness of the performed operations decreases. These categories, from least to most intuitive are: traditional lead through programming, which entails moving the robot online through the desired trajectory using the teach pendant and recording waypoints; offline programming, in which the robot's task is remotely

* Corresponding author.
E-mail address: pablo.malvidofresnillo@tuni.fi (P. Malvido Fresnillo).
URL: https://www.tuni.fi/en/research/fast-lab (P. Malvido Fresnillo).

https://doi.org/10.1016/j.robot.2023.104556
Received 21 December 2021; Received in revised form 13 September 2023; Accepted 3 October 2023
Available online 9 October 2023

programmed and simulated in a 3D model of the work cell; walk-through programming, similar to lead through but, instead of using the pendant, the operator grasps the end effector and moves it manually; and finally Programming by Demonstration (PbD), in which the teacher provides a set of demonstrations of the desired behavior to the robot and this generates a policy based on them. PbD techniques are therefore the most intuitive ones, as once the system is developed, no manual programming is required, which allows for the rapid and easy programming of the robot without the need of a robotic expert.

The PbD technique comprises of many possible approaches. Zhou et al. classified them in two groups: PbD for low-level motion and PbD for high-level task [7]. While the first group focuses on learning robot trajectories or force controllers from human demonstrations, the latter focuses on understanding the goal of the demonstration by extracting the semantics of human operations accurately. Chitnis et al. in [8] and Szynkiewicz in [9] consider that, in the case of manipulation processes, it is more interesting to focus on the tasks to be performed, rather than on the motions. This involves the extraction of a high-level plan of the sequence of steps to be executed, and then, the robot can optimize the trajectories to be followed within these steps by utilizing the information provided about the workbench layout and the manipulated object properties. Additionally, PbD approaches, and in particular, those working with manipulation applications in the task space, can implement different strategies to provide demonstrations to the robot (e.g., using wearable sensors [10,11] or a vision system [12,13]) and to analyze and understand these demonstrations, where Artificial Intelligence (AI) techniques such as Markov Models [11,14–16] or Neural Networks [17] stand out.

The aim of this paper is to develop a PbD system which is able to understand and digitize the manipulation activities performed by an operator, allowing the intuitive and easy reconfiguration of a robotic system by just providing a few demonstrations of the new task. Therefore, instead of merely copying the movements of the operator, the objective is to understand his intention and generate a high-level action plan with the operations to be performed, the objects to be used, and the positions where to execute them. Furthermore, the system must be hardware-agnostic, meaning that it can work, with better or worse performance, with any sensory device (or any combination of them) that captures information about the operator's movements or interactions with the manipulated objects; and that produces a high-level action plan that can be executed by any robot and gripper, if the morphology and structure is not a limitation.

The rest of the document is structured as follows: Section 2 provides a literature review on robot task-level PbD for manipulation applications, as well as methods for recognizing manipulation operations (with a focus on Markov Models) and devices for capturing manipulation information. Section 3 describes the developed system, starting with an introduction to the semantics used in the system that distinguishes between four manipulation levels. It then presents the conceptual approach, which can be implemented with any sensor that captures the user's movements and object interactions, focusing on how the system manages and processes this information to achieve its diverse functionalities. The section also includes a description of the implementation of this approach using a dataglove and a hand motion tracker. Section 4 presents the experiments conducted to evaluate the performance of the system and discusses the obtained results. Finally, Section 5 presents the conclusions drawn from the study and suggests possible future research directions.

2. Literature review

2.1. Task-level Robot Programming by Demonstration

Lozano-Pérez classified programming systems into three categories [18]: guiding systems, in which the robot is moved to each desired position and the joint positions recorded; robot-level systems, specifying motion and sensing in a programming language; and task-level systems, where just the goals to be achieved are specified (e.g., the operations to perform on an object or its final position). According to these definitions, both guiding and task-level systems require intuitive inputs. However, the latter is more suited for building end-user programming systems, as the user just needs to specify the goal and does not need to worry about how this is achieved. Additionally, these systems are more robust against deviations as the trajectory is optimized by the system instead of being fixed by the user.

Task-level systems (also called high-level systems by other authors [19]) allow the generation of complex robot behaviors based on a set of predefined skills. Thanks to this, the programming responsibilities can be separated between the robot programmer (who develops the robot skills), and the task specialist (who combines the existing skills to achieve the goal of the task) [20]. This can be seen in [21], where the user just needs to specify a set of goal conditions to generate the robot's program. This is possible because all the knowledge about the objects in the environment, the robot's hardware, and the robot's capabilities (a set of predefined skills), is organized in an ontology. Each of these skills requires a set of preconditions and produces a set of postconditions (i.e., the expected output state). Thus, the system is able to determine the optimal sequence of skills to reach a certain goal.

Another advantage of these skill-based systems is the easier vertical integration with the production system of the enterprise. An example of this is presented in [22], where the program of all the robots of the factory is updated automatically just by modifying the Standard Operating Procedures (SOPs) issued by the Enterprise Resource Planner (ERP). These SOPs are decomposed into a set of tasks by the Production Manager (which is a ROS package), and it determines which robot/resource has to execute each of them based on their skills. Finally, a ROS-based task-level programming module running inside each robot defines its behavior as a combination of its pre-programmed set of skills. Due to all this, task-level programming systems are, in most cases, intuitive and easy to use. However, in some applications, with a lot of similar skills (e.g., complex manipulation activities that might require different grasp types) or with a lot of labeled tools and locations which might be used to configure the skills, it can become tricky. In these cases, it would be more intuitive for the task specialist to program the robot just by demonstrating how to perform the task, which is known as Programming by Demonstration (PbD).

PbD is an online robot programming method in which a policy for the robot is derived based on the demonstration of its desired behavior [23]. Once this policy is learned, it will tell the robot what to do based on the state captured by its sensors. The main interest of PbD lies in the fact that, once the teaching system is developed, programming a new task is very intuitive and does not require advanced knowledge in robotics. There is a wide variety of PbD methods, ranging from the first approaches using symbolic reasoning and manual control [24] to advanced task-level systems using more user-friendly interfaces and machine learning techniques to generalize across demonstrations [12, 25]. Argall et al. classified PbD methods into four categories based on how and where demonstrations are captured [23]. When the demonstration is recorded directly in the robot or a morphologically identical platform, the method is classified either as teleoperation (when the teacher operates the robot) or as shadowing (when the robot mimics the teacher's motion). On the other hand, when the demonstrations are not captured directly in the robot, they are classified as either sensors on teacher (when the sensors record the teaching data directly, e.g., data gloves or motion trackers [10]), or as external observation (when the information needs to be extracted in a post-processing phase, e.g., vision system recording the operator actions [25]).

Programming systems in the first two categories focus on teaching the robot trajectories and different commands, like opening and closing the gripper [26]. The development of these approaches is simpler as the demonstrations are recorded in the robot hence, they do not need to solve the correspondence mapping between the human and the robot's body. However, the extra complexity in the approaches of the last two categories makes them more intuitive, allowing users to demonstrate tasks by just performing them, regardless of the robot's morphology. Additionally, these approaches have the advantage of being appropriate for both collaborative and industrial robots, as they do not require physical contact. Like the first two categories, sensors on the teacher and external observation systems can be used to teach trajectories to the robot [10]. However, they can also be used to understand and digitize the demonstrations performed by the user, creating a high-level action plan that the robot can execute. This is known as task-level PbD, and it combines the strengths of the PbD and task-level programming paradigms, resulting in a very intuitive, robust, and easily reconfigurable system.

The main application of task-level PbD systems is understanding manipulation processes and subsequently transfer this knowledge to the robot. These systems normally focus either on identifying the performed grasp types or on digitizing the demonstrated processes. For grasp type identification, most approaches analyze and compare the manipulation data captured during the demonstration with the different grasp types of a human grasp taxonomy (an extensive review on human grasp taxonomies can be found in [27]) using techniques like trajectory analysis or HMM [11,15]. Regarding the digitization of manipulation processes, most authors agree on defining complex processes as a sequence of skills or simpler operations [12,28]. Therefore, these approaches have to solve two problems: the segmentation of the process into simpler operations and the recognition of each of these operations among a set of pre-programmed parameterized robot operations.

Temporal process segmentation is required to determine when one operation finishes, so the recognition system can be restarted and start identifying the next operation. Different solutions have been proposed for this problem. Aksoy et al. differentiated three groups of approaches: boundary detection, sliding window and high-level grammars [12]. Boundary detection approaches study the start and end points of the operations based on temporal discontinuities or changes in velocity and acceleration [29]. Sliding window methods look for coincidences between the operation features and the manipulation segment analyzed within the sliding window, like in [17]. Finally, high-level grammars build a large network from individually modeled actions. An example of this is proposed in [28], where gesture segmentation is done by monitoring the operation's model with the maximum probability. When its likelihood drops, it is considered that the action has finished. The methods used to recognize the segmented operations are discussed in detail in the next subsection.

Finally, it can be highlighted that most of the referenced approaches just focus on determining the processes and operations performed by the user, and little attention is paid to other important aspects that can be used to parameterize the predefined skills (e.g., how much to twist, or where to grasp), which has high relevance for the quality of the resulting robot's program.

2.2. Methods to recognize manipulation operations

As stated in the previous subsection, the most popular way of identifying and digitizing manipulation processes is by segmenting them into a sequence of simpler operations. The recognition of these operations is a challenging task. Different solutions have been proposed for addressing this problem, most of them using AI techniques. In [17], Neto et al. used an artificial neural network (ANN) for static, dynamic and composed gesture classification. In a different approach, Ramirez et al. were able to recognize human manipulation activities by using a Support Vector Machine (SVM) algorithm to classify the low-level features extracted from videos by using the Independent Subspace Analysis (ISA) [13]. In [30], Gutzeit et al. were able to recognize pick-and-place and ball-throwing movements using k-Nearest Neighbor (kNN), with one neighbor ($k = 1$), to classify the segmented movements in a list of known operations. There are also many approaches based on discriminative (e.g. Conditional Random Fields (CRF) [31]) and generative (e.g. Hidden Markov Models (HMM) [11,14–16]) temporal-space models. Markov Models are a very powerful mathematical tool to model noisy temporal signals of variable length. In general, they consider the transition probabilities between all the possible discrete states of the analyzed data, thus, the model only depends on the sequence of the states and is independent of their transition speed.

Due to the previous characteristics, Markov Models are an appropriate method for modeling and recognizing manipulation operations, and many authors have used them for this purpose. In [28], Aarno et al. implemented a Layered HMM (LHMM), consisting of two layers: a first level for gestures recognition and a second level, whose input is the output of the first layer, for task recognition. In [16], Ogawara et al. modeled different action primitives by HMM hierarchical clustering, identifying later the demonstrated action by comparing it with all the existing models. A different approach was used by Vicente et al. in [14], where instead of defining a model for each operation, a unique HMM is built with the training set, representing the whole set of actions, so that different paths through the HMM correspond to different actions. Therefore, the most probable path or sequence, considering this model and the observations, determines the executed action.

As can be seen in the previous examples, there are many possibilities and ways to implement Markov Models depending on the input data and the desired functionality. One of the decisions when working with Markov Models is the topology of the models, that will depend on the application and the nature of the modeled data. The two main topologies are the ergodic models, in which every state of the model can be reached from every other state; and the left–right or Bakis model, in which as time increases, the state index increases or remains the same [32]. Another important decision is the order of the model. In some applications, the probability of an observation at time n does not depend just on the previous observation (at time $n - 1$), but also in the observations at $n - 2$, $n - 3$... In this case, we would talk about a second, third... order Markov Model, depending on the number of previous states considered for the transitions [33]. There are also other possibilities with these models that can be interesting in some situations, like the multivariate Markov Models [34], when the input data is composed of multiple dependent variables, or the Continuous Density HMM (CDHMM), in which, instead of quantizing a continuous input signal in discrete symbols, a Gaussian Distribution is associated with each state [35]. Due to all their possibilities, there are plenty of applications for this kind of models, not only for motion pattern recognition, but also in fields like: speech [35], writing [36] and biological sequences recognition, and robot navigation [37,38].

Additionally, in most cases, the states of the modeled data are not directly observable, but there is a relation between them and the observable events that are captured by the sensors. To deal with these situations, Hidden Markov Models (HMM) are used, which not only consider the transition probabilities between states (A) and the probability of starting in a certain state (π), but also the probability of seeing each of the observable events when the data is in a certain state (B). These models ($\lambda = (A, B, \pi)$) can be used to solve three main problems [32]: What is the probability of certain observation sequence, given the model? Given the model and the observation sequence, which is the most probable sequence of states? How can we adjust the model parameters to best account for the observed sequence of states?

In this paper, one of the objectives is to develop a hardware-agnostic system, meaning that it can work with any sensor (or any combination of them) that provides information about the manipulation activity. These sensors will provide dependent data to the system (e.g. the hand position, the fingers' bending, hand contact...), thus multivariate Markov Models could be used to integrate all this information. Additionally, the optimal order for each of these variables will be analyzed, resulting in an Optimized Multivariate Multiorder Markov Model.

2.3. *Techniques to capture manipulation information*

Capturing human motion and being able to continuously track it, is an important aspect to consider for implementing PbD applications in robotics. In general, the prevalence of tracking technology has increased in recent days due to advances in computing power; with improved tracking algorithms and enhanced post processors which minimize measurement errors [39]. The spatio-temporal information of the measured entity or body part (its continuous deviation from an initial point of reference) is determined by tracking its real-time position and orientation with high sampling rates. Moreover, different aspects of human physiology, motion, and gestures (which is essentially a combination of several basic motions) can be monitored to get useful information by dynamically measuring the following metrics: change in position, speed, joint angles, heartbeat rate, contact pressure, muscle contraction/ relaxation, eye tracking, etc.

With the current advances in technology, there are several different techniques to track human physiological and motion metrics, by using various sensing techniques and mechanisms [40]. These human motion sensing techniques could be widely classified as (i) Vision-based sensing (ii) wearable sensing and (iii) multimodal sensing. This section briefly expands on these categories and highlights the devices used, their principle of operation and respective applications.

Vision-Based Sensing utilizes optical cameras (marker-less sensing systems) or Infrared (IR) cameras (marker-based sensing systems) to track the elements of the human body and are most suitable for determining the position/location information than recognizing gestures. Cameras utilized for motion capture include RGB cameras and RGB-Depth or RGB-D cameras, and they are predominantly suitable for capturing human motions at a low cost with stability [41,42]. RGB cameras (without depth perception) face challenges to effectively capture the dynamics of human motion as occlusions can take place due to perspective projection, and it is difficult to extract accurate information from a single camera setup. This can be overcome by using a multi-camera setup at different angles to observe the motion in 3D [43]. However, this limits the sensing area, requires the cameras to be pre-calibrated, and is expensive to set up. RGB-D overcomes the above-mentioned issues of the RGB camera, as it captures the depth information for each pixel in the frame along with the RGB data. This allows improved recognition of human body motions, hand gestures, and facial features [44], enabling their applications in gaming, augmented reality, human motion analysis etc. These belong to the sub-category of marker-less sensing systems, where the human body is tracked without wearing any distinct-trackable markers; and the location, orientation and change in posture is identified through image segmentation and processing which can be matched to a human template [45]. Whereas, in marker-based sensing systems, special IR cameras are only used to track the specialized visual markers placed on the human body/limb instead of capturing and processing RGB or RGB-D images. Since the markers are specifically made to be tracked by the IR camera, these systems could provide accurate position/location data in real-time; and are unaffected by varying light conditions, object occlusions, and disturbances in the scene [46]. These advantages overcome the requirement of specialized cameras and inconvenient markers to be utilized for specialized applications (i.e., gait analysis, animation, robot programming and interaction, etc. [47]).

The previously described vision-based sensing systems are more suited for capturing information pertaining to the joint angles of human body/limb or their location and orientation, however, they are not suited to capture the minute gestures and movements of the human's fingers nor can they sense the forces exhibited by the human while performing the motions. This is made possible by utilizing wearable sensors, to effectively capture these types of information. Wearable sensors can be easily integrated with regular apparels such as clothes, gloves, watches, neck tags, eyeglasses, etc. There are numerous wearable sensor technologies available in the market [48–50], but for the relevance of this paper, this section would briefly elaborate on electromyographic sensors and datagloves. Electromyography (EMG) is the method of tracing and capturing the minute electrical signals generated by the muscles of the human body when they are expanding or contracting. An EMG sensor is the transducer which is used to measure these signals, and they have gained a lot of interest in recent times for examining human manipulation and movement. There are a few approaches where the EMG sensors are placed at different layers of the human body, however, the most popular approach is the surface Electromyography. Its popularity is due to the fact that this approach is non-invasive, as the sensor arrays are placed on the skin overlaying the muscle to be studied. This is the most suitable approach for capturing and analyzing human motion and actions [51]. The other approach is intramuscular Electromyography, and it involves the placement of electrodes into the required muscle fibers (invasively), for acquiring accurate signal data without too much noise and crosstalk, and is predominantly used in the clinical neurophysiology field for the proportional control of prosthetic devices [52]. On the other hand, datagloves have emerged as one of the most important devices for tracking and studying human hand configuration and finger movements. The advances in sensor technologies have led to the integration of inertial sensors, accelerometers, bend sensors, touch sensors, etc. to determine the angle and position of the hands and fingers, the finger abductions and adductions, pressure applied, and to provide haptic feedback to the user [53]. This makes the datagloves suitable for providing a wide range of information that could be input to the system, which could be processed to extract various commands to create an interface with a wide range of applications in gaming, controlling mobile and industrial robotics, etc.

All of the above-mentioned devices are suitable to serve their functions as standalone setups, but for more comprehensible and versatile applications, where reliable information about human motion is required, simultaneously implementing two or more of these technologies and combining their outcomes yield significantly improved results. This is multimodal sensing, in which the shortcomings of a particular type of sensor could be supplemented by the addition of a secondary sensor system. An illustration for this is a teaching application for an industrial robot by using gestures and visual hand movements [10], where the teacher provides hand gestures (for determining the operation mode) as signals through a data glove and uses an RGB-depth camera for providing trajectory data to the manipulator. Though multimodal sensing systems capture more versatile information as opposed to standalone sensor systems, they have complex system architectures and require high computational capabilities to synchronize data and extract useful information. This manuscript deals with a multimodal sensor system implementation, where it combines the functionality of a data glove with a marker-based visual motion capture system.

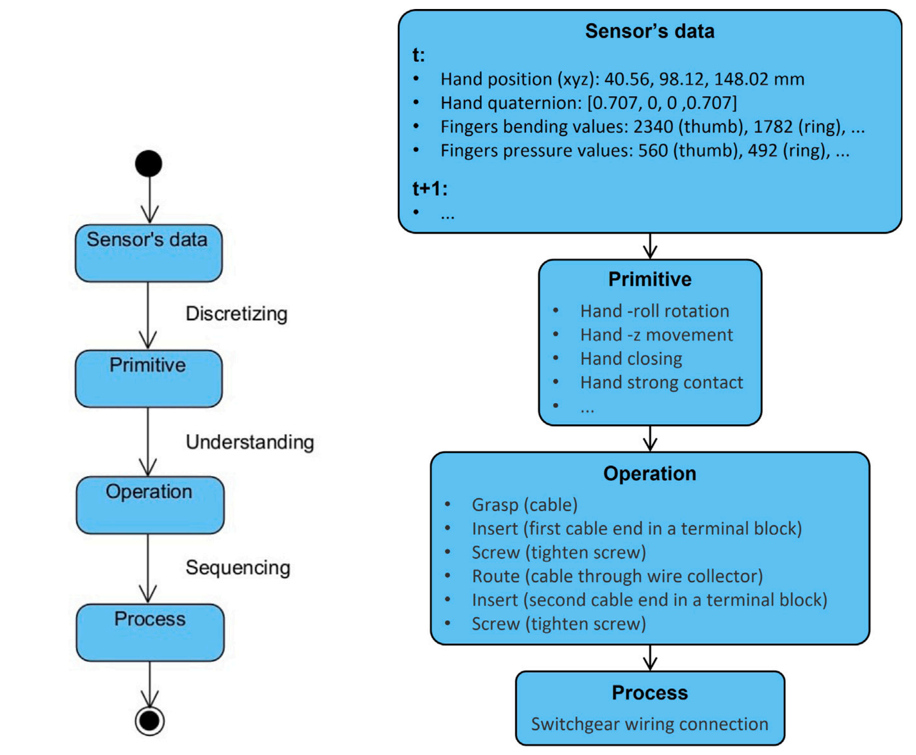
3. Proposed approach

This section provides a detailed examination of the approach proposed in this paper. To enhance its clarity and comprehension, an accompanying video[1] has been prepared. This video visually explains all the steps and techniques of the approach and demonstrates how it works in practice. We encourage readers to consider watching this video before beginning this section, as it can serve as a helpful introduction and preparation for the more technical content that follows.

3.1. *Levels of the manipulation activities and associated semantics*

The goal of this paper is understanding and digitizing single-arm manipulation processes performed by humans, based on the data captured by different kind of sensors. The presented approach does not

[1] https://www.youtube.com/watch?v=nPZHHYW00rE


Sensor's data
Discretizing
Primitive
Understanding
Operation
Sequencing
Process
Sensor's data
t:
Hand position (xyz): 40.56, 98.12, 148.02 mm
Hand quaternion: [0.707, 0, 0 ,0.707]
Fingers bending values: 2340 (thumb), 1782 (ring), ...
Fingers pressure values: 560 (thumb), 492 (ring), ...
t+1:
...
Primitive
Hand -roll rotation
Hand -z movement
Hand closing
Hand strong contact
...
Operation
Grasp (cable)
Insert (first cable end in a terminal block)
Screw (tighten screw)
Route (cable through wire collector)
Insert (second cable end in a terminal block)
Screw (tighten screw)
Process
Switchgear wiring connection

(a) UML State Machine Diagram of the Manipulation levels. (b) Example of the data obtained in the different manipulation levels for a switchgear wiring process.

Fig. 1. Levels of the manipulation activities.

depend on any specific type of sensor, but all of them must provide raw information about the user's movements and/or interactions with the manipulated objects. The data obtained from these sensors is simple and does not depend on other sensors' readings, nor on time, nor on its sequence. Therefore, the sensor data must be analyzed, processed, and integrated, increasing its complexity level until it can be compared with manipulation processes.

In order to do this, the first step is to understand what a manipulation process is and what it is composed of. In [12], Aksoy et al. suggest a three level hierarchical manipulation structure. Manipulation sequences or activities (e.g. making a sandwich) are composed of atomic manipulations (e.g. taking a slide of bread or cutting a cucumber), and these, in turn, are composed of manipulation primitives, that are the smallest basic components of a manipulation (e.g., cutting or lifting). Szynkiewicz uses a similar approach in [9], decomposing manipulation tasks (e.g. solving a Rubik's cube) into manipulation skills (e.g. turning a single face of the Rubik's cube), and these into basic skills (e.g. concrete grasping and releasing strategies). Aarno et al. [28] also proposes three manipulation levels. The simplest level consists of the motion data captured by the sensors; the next level are the gestemes, that are any arbitrary motion in 2D or 3D; and finally the tasks, such as: move, align, insert... In this paper, a new single-arm manipulation structure is proposed, differentiating four levels of complexity (Fig. 1). These levels are described below, from simplest to most complex.

- *Sensor's data* - Raw data captured by the sensors. It depends on the type of sensors utilized and their resolution. Some examples of this manipulation level could be the bending angle of a finger, the pressure of a contact point or the position and orientation of the hand.
- *Primitive* - Discretized sensor's data with a semantic meaning. The number of discrete levels into which the sensor's data is classified is critical. Having many levels could reduce the robustness of the approach, resulting in more deviations and less consistency between demonstrations of the same operation, thus hindering its recognition. However, having few could lead into confusions between similar operations due to the low resolution of the primitive. Hence, a balanced optimization must be made. Additionally, primitives can be classified into different primitive's variables depending on the type of information they provide. Examples of primitive's variables are: the hand motions in the world axes (e.g., +X hand motion, -Z hand motion ...), derived from the sequence of hand positions, or the hand gestures (e.g., open, half-closed, closed), calculated by discretizing and combining the fingers' bending angles captured by the sensors.
- *Operation* - Simplest manipulation action that can be performed on an object. An operation can be described and identified as the sequence and combination of multiple primitives' variables. Examples of operations could be grasp, place, or insert.
- *Process* - Combination of discrete operations performed with/on an object, in sequence, to achieve the desired manipulation goal. An example of process could be grasping a bottle, pouring water in a glass, and placing it again, whose common goal is serving water.

The life cycle of the data, since its acquisition until its transition to a process manipulation level, is managed by three of the system modules. The first one is *Discretizing*, that is in charge of the acquisition of the sensor data, its transition to the primitives level and its storage. The next stage is *Understanding*, that manages the transition of the primitives to the operations level, as well as the segmentation of a full manipulation process in individual operations. After this step, the manipulation is understood and can be digitized as a sequence of operations. However, the robustness can be increased substantially with a third module, *Sequencing*, that manages the transition of the data to the process level, identifying the joint goal of the manipulation, based on the recognized sequence of operations.

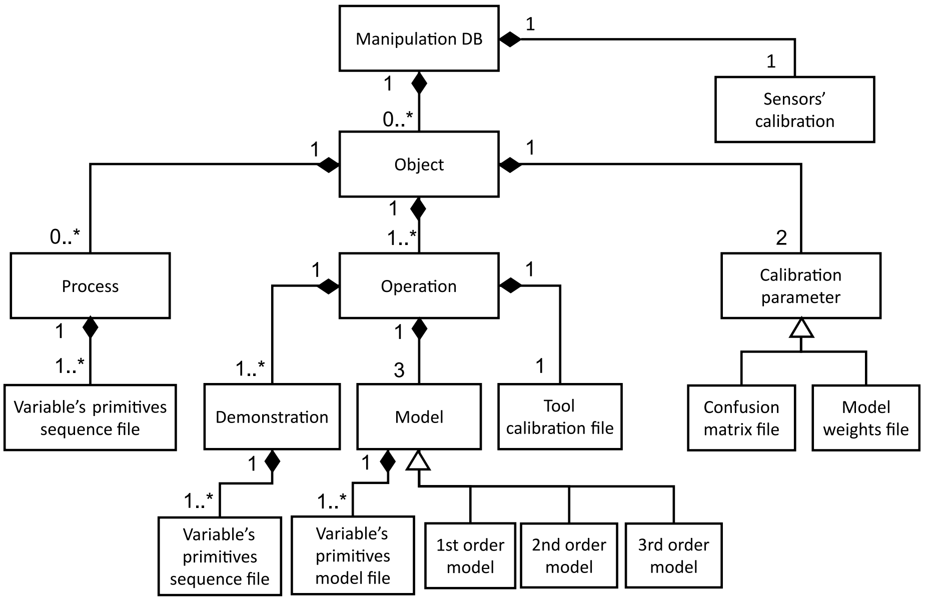


Fig. 2. UML class diagram of the Information system's model.

3.2. Conceptual definition of the approach

The developed system is independent of the sensor technology utilized, with the only requirement being that they must provide information about the operator's movements or interactions with the manipulated objects. This subsection, therefore, presents the approach from a conceptual point of view, without considering its implementation with any sensor in particular.

3.2.1. Information system

In order to recognize manipulation operations, the recorded manipulation data have to be compared with all the possible operations, thus, a model is needed for each of them. The developed system should be able to work with multiple objects, and each object needs to have models for all its operations. Training all these models to be generic and robust requires a large amount of training data. Additionally, all the files required for the system and sensor's calibration and configuration have to be stored somewhere. Therefore, to manage all this data and make the system scalable, it is important to have a structured and well-organized information system.

The adopted information model can be seen in Fig. 2. The first level organizes the data by object. This initial division by object is because the actions for executing the same operation vary depending on the manipulated object, due to its differences in weight, size, shape, surface friction or deformability, as reported in [54]. This can be clearly observed in a grasp operation. When this operation is performed with a bottle, normally a cylindrical grasp is done, with all the fingers surrounding the bottle, however, when the grasped object is a cable, a tip pinch or palmar pinch grasp [55] are the most common ones. The pattern of a grasp operation for each of these objects is completely different and, therefore, every object requires their own operation's models.

In the second level of this information system, every object contains a directory for each of its possible operations, a directory for the processes recorded using that object, and another directory for storing two configuration files, required for the operations recognition and process classification with that object. One level deeper, each operation contains a set of demonstrations recorded to train its models, a set of models that compose the Multiorder Multivariate Markov Model of the operation, and a configuration file used to calibrate the system for the object dimensions.

Each demonstration is stored as a set of files, each of them containing the detected sequence of primitives of a certain variable. For instance, one file contains the sequence of primitives of cartesian hand movements (e.g., +z movement, -x movement...), another file contains the sequence of fingers positions primitives (e.g., 5 fingers open 0 closed, 3 fingers open 2 closed...), etc. Additionally, the sequence of all primitive's variables is also stored in a single file (e.g., -z movement, roll hand rotation, hand closing...) to consider the chronological dependency between primitives of different variables. The reason for this is that in most manipulation operations, only the sequences of primitives of some variables follow a pattern, while others do not provide useful information. Therefore, storing the sequence of all the primitives captured during a demonstration in a single file could hinder the recognition of the operation.

Regarding the models, an Optimized Multiorder Multivariate Markov Model is created for each operation. To define these models, the information system must store the first, second, and third order Markov Models of all the primitive variables. This is possible because, as explained previously, the demonstrations store the sequence of primitives of each variable in different files, which allows to train the model of each variable individually. Then, all these models have to be integrated. To do this, an order is selected for the model of each variable (first, second, or third, in this case) and a weight is assigned to it according to their importance for recognizing the operation pattern (similar to [11]). These weights are optimized by an iterative algorithm that maximizes the recognition accuracy. Regarding the order of the models, when it increases, the similarity between the recorded operation and its model normally decreases, as it becomes more sensitive to deviations. However, this also produces an increment in the similarity ratio between the most similar operation and the rest of the operations, helping in differentiating between similar operations. As a consequence, for some variables it would be more interesting to use a higher order model, while for others a lower order model would give better results. Therefore, the order of each variable's model is tuned experimentally to maximize the system performance.

Finally, each process is stored in a folder whose name indicates the sequence of operations and their locations, providing ground truth for testing the system. As for the operations, each process is stored in different files, each of them containing the sequence of primitives of different a variable, but now including also the time at which every primitive was observed, so the primitives of all variables can be considered in a

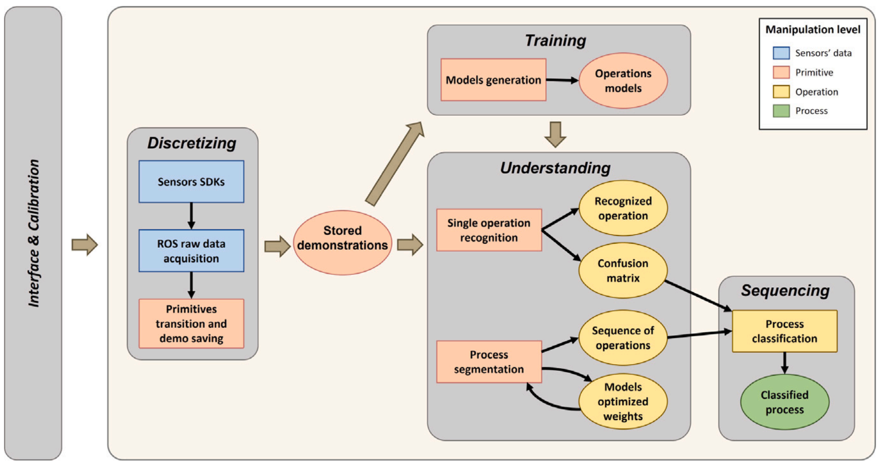


Fig. 3. System overview. The square shapes represent the actions performed by the modules, and the oval shapes represent the results of these actions. The arrows show the intra and inter module communication, and the colors represent the manipulation level of the information.

common timeline for the operations segmentation. Additionally, each process folder contains a file with the cartesian coordinates of the hand at each timestep, which is used for the recognition of the location of each segmented operation.

3.2.2. *System functionalities*

The developed system is composed of five main modules, integrated in a ROS-based architecture. Three of these modules were introduced in Section 3.1: *Discretizing*, *Understanding*, *Sequencing*. These are in charge of the transition of the data to higher complexity manipulation levels. The fourth module is called *Training* and, as its name indicates, it is responsible for generating the operations models. Finally, the last module, called *Interface & Calibration*, manages the communication of the user with the system, for recording new demonstrations and calibrating the sensors. Fig. 3 shows, with a schematic representation, the main functionalities of these modules and the communication between them. Intra and inter module communication takes place through txt files and ROS topics.

All the system functionalities are initialized from the Interface & Calibration module. This module allows the user to calibrate the sensors and to start the desired system functionality: Record a new demonstration, train the system, calibrate the system or recognize an operation or a process.

In order to be able to use the rest of the system functionalities, first it is necessary to populate the information system. Therefore, the first functionality of the system is recording demonstrations (Fig. 4). This is the only functionality that requires the use of the Discretizing module, as it is the only one in which user's data has to be captured. This module establishes a socket communication with the SDKs of all the sensors, introducing the captured raw sensor's data in the ROS system. Then, a calibration of the sensors is performed before starting the recording. The sensor's data is then sent to different ROS nodes, tailored for the specific sensors utilized, that process and discretize it, converting the data to the primitive manipulation level. Finally, the recorded sequence of primitives of each variable is saved in a different txt file, according to the information structure presented in Fig. 2. Both the calibration of the sensors and the discretization of the captured data depend on the sensors used. Therefore, this is explained for the specific implementation of the system in Section 3.3.

Once the information system is populated with the users' demonstrations, the system can be trained (Fig. 5). For this functionality, the Training module is called, generating or updating the models of each operation, generalizing across all the recorded demonstrations. First, second and third order Markov models of every primitive's variable are generated for every operation.

After training the system, it is necessary to the perform system calibration before it can be used to recognize processes (Fig. 6). Two calibration files are needed for the processes recognition, one with the confusion matrix of the object's operations recognition and the other with the weight assigned to each variable's model, and both are calculated using the Understanding module. For computing the confusion matrix, a cross-validation with all the demonstrations of operations of an object is made. This means that for every demonstration, the models of that operation are recalculated excluding the evaluated demo, then the demo is compared with the models of all the operations, selecting the one with the highest similarity. This is repeated for all the demos, populating a confusion matrix, which indicates how many operations were recognized correctly and how many were recognized as other operations.

Regarding the calculation of the optimal weight of the variables' models, an iterative optimization algorithm is used. In every iteration, all the recorded processes of an object are segmented, and the results are compared with the ground truth calculating the overall performance with a metric called *performance*, described in Section 4. This is automated, as the name of the folder in which each process demonstration is stored indicates the sequence of operations and locations (i.e., the ground truth). After each iteration, the weight of the model of one variable is modified in one direction (increasing or decreasing a certain step value). If the performance improves, the weight is modified in the same direction in the next iteration. If not, the opposite direction is evaluated. When this weight of the variable is adjusted, the same process is repeated for the next variable until all the weights are determined.

Finally, the last and main functionality of the system is the operation and process recognition (Fig. 7), for which the Understanding and Sequencing modules are required, as well as the models and calibration files generated during the system training and calibration. The Understanding module is used to convert the demonstrations, stored as sequences of primitives, to the operation level. This can be an operation or, in the case of processes recognition, a sequence of operations, obtained by performing a segmentation. In order to do this, the analyzed demonstration is compared with all the possible operation models of that object, using the optimized variables' weights. After this, the Sequencing module increases the data complexity an additional level, reaching the process level. This is done by comparing the segmented sequence of operations with all the possible processes for that object, using the confusion matrix results to consider the probabilities of wrong operations identification. Finally, the selected process is the one that presents the highest similarity.

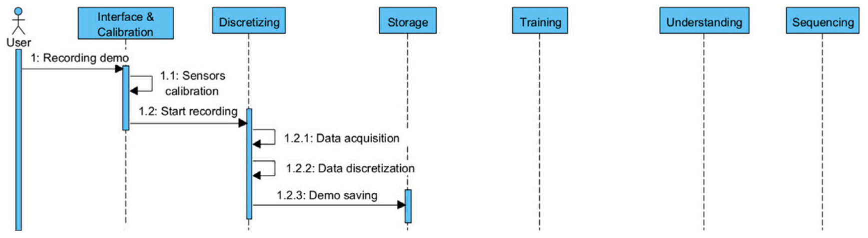


Fig. 4. UML sequence diagram of the recording functionality.

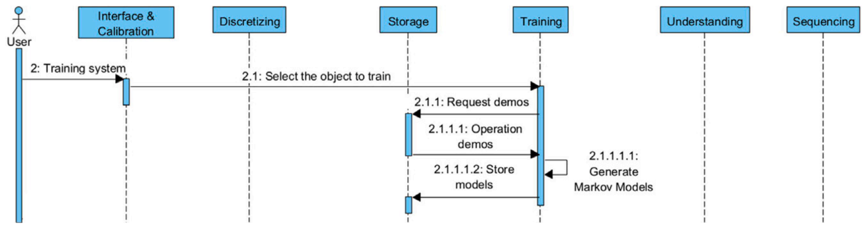


Fig. 5. UML sequence diagram of the training functionality.

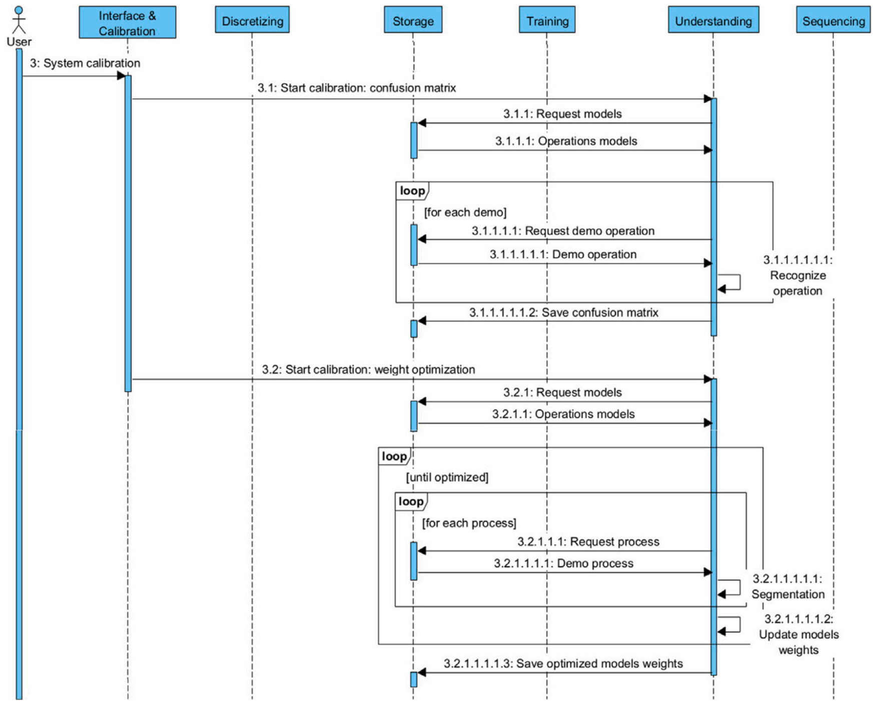

User
Interface & Calibration
Discretizing
Storage
Training
Understanding
Sequencing
3: System calibration
3.1: Start calibration: confusion matrix
3.1.1: Request models
3.1.1.1: Operations models
loop
[for each demo]
3.1.1.1.1: Request demo operation
3.1.1.1.1.1: Demo operation
3.1.1.1.1.1.1: Recognize operation
3.1.1.1.1.1.2: Save confusion matrix
3.2: Start calibration: weight optimization
3.2.1: Request models
3.2.1.1: Operations models
loop
[until optimized]
loop
[for each process]
3.2.1.1.1: Request process
3.2.1.1.1.1: Demo process
3.2.1.1.1.1.1: Segmentation
3.2.1.1.1.1.2: Update models weights
3.2.1.1.1.1.3: Save optimized models weights

Fig. 6. UML sequence diagram of the calibration functionality.

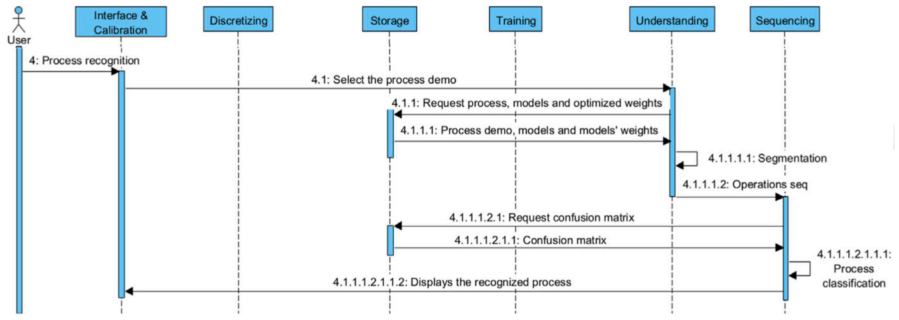

User
Interface & Calibration
Discretizing
Storage
Training
Understanding
Sequencing
4: Process recognition
4.1: Select the process demo
4.1.1: Request process, models and optimized weights
4.1.1.1: Process demo, models and models' weights
4.1.1.1.1: Segmentation
4.1.1.1.2: Operations seq
4.1.1.1.2.1: Request confusion matrix
4.1.1.1.2.1.1: Confusion matrix
4.1.1.1.2.1.1.1: Process classification
4.1.1.1.2.1.1.2: Displays the recognized process

Fig. 7. UML sequence diagram of the operation and process recognition functionality.

Algorithm 1 Automatic process segmentation

```
1: procedure GetModels(object)
2:     $operations \leftarrow$ get the operations of the $object$
3:     for $op$ in $operations$ do
4:         for $file$ in $model\_files$ do
5:             $f1 \leftarrow$ open $object/op/models1/file$.txt
6:             $models1[op][file] \leftarrow$ read and store $f1$
7:             Same to populate $models2$ and $models3$
8: procedure Compare($demo$)
9:     for $file$ in $model\_files$ do
10:        $f \leftarrow$ open $demo/file$.txt file
11:        for $line$ in $f$ do
12:            $time[file] \leftarrow$ append time from $line$
13:            for $op$ in $operations$ do
14:                $i\_prob \leftarrow$ compare $line$ with $models[op][file]$ starting probability
15:                $tr\_prob \leftarrow$ compare $line$ with $models[op][file]$ transition probability
16:    $t \leftarrow$ join and sort $time[file]$ lists
17:    for $op$ in $operations$ do
18:        for $file$ in $model\_files$ do
19:            $i\_prob \leftarrow$ interpolate $i\_prob$ for time in $t$
20:            $i\_prob[file] \leftarrow i\_prob[file] \cdot weight[file]$
21:            $tr\_prob \leftarrow$interpolate $tr\_prob$ for time in $t$
22:            $tr\_prob[file] \leftarrow tr\_prob[file] \cdot weight[file]$
23:        $init\_prob[op] \leftarrow sum(i\_prob[op])$
24:        $trans\_prob[op] \leftarrow sum(tr\_prob[op])$
25:    return $t, init\_prob, trans\_prob$
26: procedure Segment($t, init\_prob, trans\_prob$)
27:    $order \leftarrow$ get optimal polynomial regression order
28:    for $op$ in $operations$ do
29:        $init\_fit[op] \leftarrow fit(t, init\_prob, order)$
30:        $trans\_fit[op] \leftarrow fit(t, trans\_prob, order)$
31:        $start\_index[op] \leftarrow$ get $init\_fit[op]$ max values
32:    for $op$ in $operations$ do
33:        for $i$ in $start\_index[op]$ do
34:            if $trans\_fit[op][i] \geqslant trans\_fit[i]$ and $trans\_fit[op][i] > 30\%$ then
35:                $coord \leftarrow$ get coordinates at $t[i]$
36:                $loc \leftarrow$ get location of $coord$
37:                $segm \leftarrow$ append $\{t[i] : [op, loc]\}$
38:    $op\_seq \leftarrow$ sort $segm$ by time
39:    return $op\_seq$
40: procedure Main($object, demo$)
41:    call $GetModels(object)$
42:    $t$, $init\_prob$, $trans\_prob \leftarrow$ call $Compare(demo)$
43:    $op\_seq \leftarrow$ call $Segment(t, init\_prob, trans\_prob)$
44:    $plot(t, init\_prob, trans\_prob)$
```

3.2.3. Understanding module

The name of this module comes from its ability to understand a manipulation process, determining which operations the operator is doing, where he/she is doing them and when; presenting it as a decomposed high-level action plan. This module is composed of two main nodes, that allow the recognition of operations, isolated or in a sequence. Additionally, these nodes can be used to calculate two system parameters: the operation confusion matrix and the variables' models optimized weights (Fig. 6).

For the single operation recognition, the probability of the observed sequences of primitives (obs_j) is calculated with all the models of all the operations, and the operation with the highest likelihood is selected (op). Each operation is defined by a Multiorder Multivariate Markov Model (λ_i), composed of several models ($\lambda_{i,j}$), one for each primitive's variable, and each one with an associated weight (W_j) (where i represents the index of the operation and j the index of the variable). Additionally, each variable's model can be a 1st, 2nd, or 3rd order Markov Model, and the order used for each of them is determined empirically. Considering all this, the following formula is applied for the single operation recognition is done:

$$op = \underset{i}{argmax}\left(\sum_{j=0}^{len(files)} W_j \cdot P(obs_j \mid \lambda_{i,j})\right)$$

However, the complexity increases when recognizing operations within a process, as there is no information regarding when each operation starts and finishes, and the models cannot be applied directly. The procedure followed for the process segmentation is presented in Algorithm 1 and it was inspired by [28], that monitors the most probable model seeking a sharp drop in its probability, and [17], where a sliding window approach is adopted to evaluate and segment a sequence of observations.

The procedure followed starts by calculating the starting (or initial state) and transition probabilities of all the operations in the timeline of the observation. For this, the probabilities of every variable's sequence of primitives are merged in the same timeline. Then, these probability timelines are fitted with a polynomial regression, which smooths the curve and removes the outliers. Next, the resultant regression curves are analyzed. For every operation, when a maximum is detected in its initial state probability curve, a window composed of the values of the transition probability curve during the next four timesteps is calculated for all the operations. If the maximum average value of this window corresponds to the operation with a maximum in its initial state function and it is higher than 30%, the operation is added to the segmented sequence of operations. Additionally, the coordinates of the hand during this timestep are compared with all the known locations and, if there is any coincidence, this location is added to the segmentation too.

3.2.4. Sequencing module

After processing the process observation with the Understanding module, the operator actions are understood and an action plan to perform the demonstrated manipulation is produced, specifying the sequence of operations that have to be executed and in which locations. However, in some cases, not all the operations of the generated action plan are contributing to reach the final goal of the manipulation. This is in most of the cases due to segmentation errors (normally because of confusion between similar operations), but can be also caused by the sensors' inaccuracy, lack of data, and/or by the inefficient execution of the process (i.e.,performing unnecessary operations). A clear example of this last point is when the object grasping is not correct and a regrasping is needed, resulting in the detection of two consecutive grasp operations in the segmentation. Therefore, a final Sequencing processing module can increase considerably the performance of the system, making it more robust against disturbances.

The Sequencing module, as it name indicates, recognizes the performed process based on the sequence of operations segmented from the observation. The procedure followed for this is presented in Algorithm 2. In this procedure, all the possible combinations of the segmented sequence of operations (without altering their order) are compared with the sequences of operations of all the possible processes for the manipulated object. Finally, the process with the highest similarity is selected.

This procedure is explained with an example. Let us suppose that the performed process was pouring water with a bottle, that is composed of the operations: Grasp (G) - PourWater (PW) - Place (P); but the segmentation result was the sequence: GraspTop (GT) - Place (P) - PourWater (PW) - Place (P), where GraspTop refers to grasping the bottle from the top instead of in the cylindrical face. In order to classify this observation, the segmented sequence is compared with all the possible processes; as an example, this is shown for the case of the $G - PW - P$ process. In order to compare the process and the segmentation, both must have the same length, the one of the process. Thus, all the three-operation sequences that can be extracted from the segmentation ($GT - P - PW - P$), with a maximum of two positions

Algorithm 2 Process classification

```
1: procedure CLASSIFY($segm, process\_list$)
2:     $segm\_len \leftarrow length(segm)$
3:     for $proc$ in $process\_list$ do
4:         $seq\_len \leftarrow length(proc)$
5:         $index\_check\_list \leftarrow$ get all the possible index combinations for $segm$, with a length of $seq\_len$ and without changing their order
6:         for $segm\_index$ in $index\_check\_list$ do
7:             for $i \leftarrow 0$ to $seq\_len$ do
8:                 if $segm\_index[i] < 0$ then
9:                     $prob\_calc \leftarrow$ append 0
10:                if $segm\_index[i] >= segm\_len$ then
11:                    $prob\_calc \leftarrow$ append 0
12:                else
13:                    $op\_seq \leftarrow [proc[i]]$
14:                    $op\_segm \leftarrow [segm[segm\_index[i]]$
15:                    $prob\_calc \leftarrow$ append $conf[op\_seq][op\_segm]/max(conf[op\_seq])$
16:            for $i \leftarrow 0$ to $segm\_len$ do
17:                if $i$ not in $segm\_index$ then
18:                    $prob\_calc \leftarrow$ append 0
19:            $prob\_avg \leftarrow average(prob\_calc)$
20:            if $prob\_avg > max\_prob$ then
21:                $max\_seq[proc] \leftarrow segm\_index$
22:                $max\_seq[proc] \leftarrow prob\_avg$
23:     $proc, max\_seq, max\_prob \leftarrow max(max\_seq)$
24:     if $max\_prob < 30\%$ then
25:         $proc \leftarrow$ None
26:     $loc \leftarrow segm[max\_seq][loc]$
27:     return $proc, loc$
28: procedure MAIN($object, op\_segm$)
29:     $conf \leftarrow$ read and store confusion matrix of $object$
30:     $process\_list \leftarrow$ possible processes with $object$
31:     $process, loc \leftarrow Classify(op\_segm, process\_list)$
```

between consecutive operations, are compared with the process. Some examples of this are: $GT - P - PW$, $GT - PW - P$, $GT - P - P$ or $P - PW - P$, but also others, considering the case when the first ($X - GT - P$) or the last ($PW - P - X$) operations are not detected (the X represents no operation, because it is before the first operation of the segmentation or after the last one). As an example, this comparison is shown for the most similar three-operation sequence: $GT - PW - P$. The sequences are compared operation by operation, using the formula in Line 15. First, GT is compared with G, these are similar operations, so the result based on the confusion matrix values, is, for instance, 0.3. Then, PW is compared with PW and P with P, resulting in 1.0 probability in both cases. Additionally, a 0.0 is also considered for the calculation of the similarity according to Line 18, as the second operation of the segmentation (P) was excluded from this comparison. Finally, the average of all these values is calculated, obtaining the similarity between the sequences, 57.5%. After checking all the sequences and all the processes, no higher similarities are found, and therefore, the observation is classified as a pouring water process. Thus, even with a wrong segmentation, the overall goal of the manipulation (i.e., the process) is recognized, thanks to this last module.

This classification is also applied to the recognition of the locations where the operations were performed. Therefore, the final locations considered, are the ones corresponding to the operations sequence with the highest similarity with the process.

Additionally, to increase even more the robustness and the accuracy of system, this module offers the possibility of performing several demonstrations of the same process. In this case, initially every demonstration is processed individually as already presented in this section, calculating the maximum similarity with each process. Then, when all the demonstrations are processed, the system takes all the individual results and calculates with them the similarity average for each process.

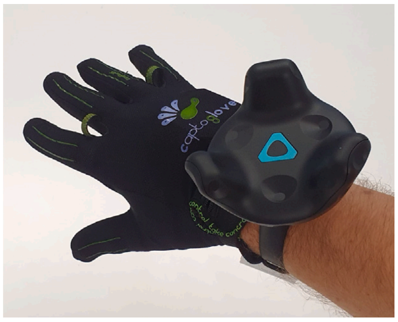


Fig. 8. Sensors used for the implementation of the system: CaptoGlove and HTC VIVE Tracker 2.0.

Finally, the selected process will be the one with the highest similarity average with the demonstrations. This is also applied for the locations recognition following the same principle. The advantage of this method is that, even if one or two demonstrations were not recognized correctly, if most of them are classified precisely, the outliers effect will be reduced and the process will be successfully identified. Thus, the more demonstrations, the more accurate the result should be, but the more time would be spent recording them.

3.3. Implementation

The system was implemented using two wearable devices to record the user's demonstrations, a dataglove and a motion tracker, as can be seen in Fig. 8. The dataglove is a CaptoGlove, equipped with: five finger sensors able to perceive finger movement on the bending axis, five pressure sensors for fingertips able to perceive pressure from 100 g to 10 kg, a triple-axis gyro, a triple-axis accelerometer, a triple-axis magnetometer, and a barometer. The motion tracker system consists of a HTC VIVE Tracker 2.0 attached to the user's wrist, and two HTC VIVE Base stations 1.0 that detect the tracker position in an area of approximately 3.5 m × 3.5 m.

Both devices communicate with a Windows computer using Bluetooth. In order to communicate with the ROS system, running on a Linux computer, the SDK of both sensors is modified, creating a socket client in each of them, to send the captured raw data to a socket server integrated in the Discretizing module of the ROS system, with a period of 0.15 s. Then, this node distributes the sensor's data to a net of nodes (see Fig. 9) that discretize it into primitives and store their transitions. This net is composed of three nodes specialized for different kinds of information: fingers' data, hand orientation, and hand position.

The *Fingers Analysis* node receives the raw glove values, including fingers' bending (fp) and fingertips pressure (fp), and discretizes them into a set of primitives' variables. In particular, fingers' bending data is converted into two different primitive variables (hand gesture and fingers' bending) and fingers' pressure data is converted into another two (hand contact and fingers contact). Threshold values were empirically defined for the discretization of these primitives (see Table 1), differentiating between three states for fingers' bending (open, half-closed, closed) and three states for fingers' contact (no contact, contact, strong contact). Regarding hand gesture, it just considers gesture transition (opening/closing) according to the fingers' bending values, and hand contact considers an overall contact for the hand, which can be: contact, in case any finger detects a simple or strong contact, or no contact otherwise.

The *Hand Orientation Analysis* node receives the raw orientation of the hand from the glove and it calculates the hand rotation and orientation primitives. For the hand rotation analysis, it calculates the rotation matrix between two consecutive timesteps and, if any of the Euler angles is above the empirically defined threshold (φ_r), a rotation in one of the hand's axes is detected. Additionally, higher threshold

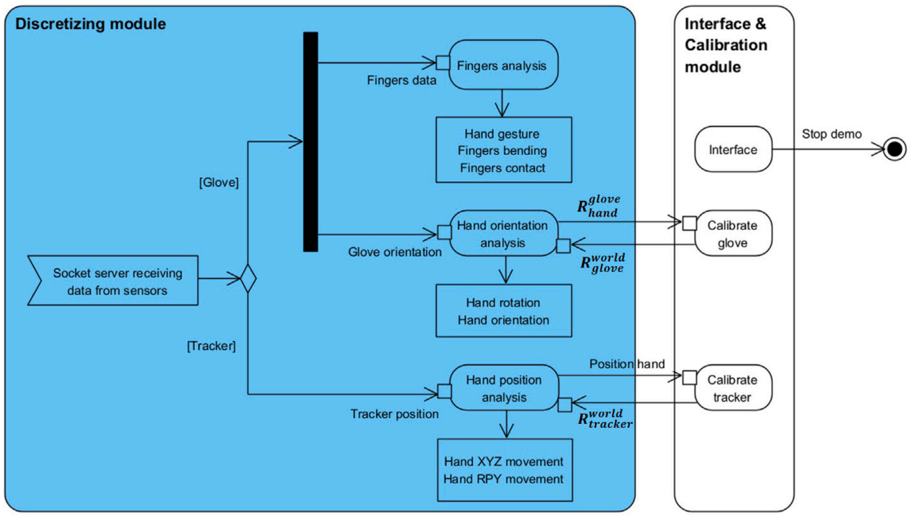


Fig. 9. UML activity diagram of the Discretizing module and its interaction with the Interface & Calibration module.

values are also used to identify fast rotations. Regarding the hand orientation, the rotation matrix of the glove is analyzed, identifying which hand axis (raw, pitch or yaw) is aligned with the gravity axis (i.e. the z world axis), if any.

Finally, the *Hand Position Analysis* node receives the raw position of the hand from the tracker (P) and it calculates the hand discrete movements in the world (XYZ) and hand (RPY) axis. In both cases, first, the distance in the different axes between timesteps is calculated. In the case of XYZ motions, these distances are calculated just as the position difference in the different axes. However, this is a bit more complicated for the RPY motions, as the hand positions have to be transformed to the hand reference frame. Therefore, this distance is defined as the translation vector of the homogeneous transformation matrix between two timesteps. Then, the calculated distances are analyzed to determine if there was a displacement in any of the axes. However, the analysis is more complex now than for the previous two nodes, as slow movements have to be detected but, at the same time, it must be robust against hand vibrations. Therefore, an initial threshold (φ_m) is used for the $t-1$ to t timesteps transition, if no movement is detected, it is checked again from $t-2$ to t timesteps using $1.5\,\varphi_m$, and this is repeated again for a third level using $1.75\,\varphi_m$ (this can be seen Table 1). This way, small isolated movements are discarded but small and continued movements are detected.

Additionally, the Discretizing module communicates with the Interface & Calibration module for calibrating the sensing devices. This calibration is composed of an online calibration, performed every time the teaching system is launched, and an offline calibration, which just needs to be done once and it is saved. In the online calibration, the devices' readings are transformed and scaled into the world's frame. For this, the user has to follow some steps, which are indicated in Fig. 10. The first step is the alignment of the gloves and tracker axes with the world ones. For this, it is necessary to obtain the rotation matrices between the world and the axes of the devices (i.e., R^{world}_{glove} and $R^{world}_{tracker}$), so the sensor's readings (i.e., R^{glove}_{hand} and $R^{tracker}_{hand}$) can be transformed to the world frame (i.e., R^{world}_{hand}). In the case of the tracker, the gravity axis (i.e., Z) is already aligned with the world's one, so just one rotation angle is needed. This angle is calculated by analyzing the initial and final X and Y coordinates captured by the tracker after moving the hand along the X and Y axes. Regarding the glove, it is necessary to calculate the 3D rotation matrix. For this, the user just needs to align the hand with the world axes (i.e., Roll with X, Pitch with Y, and Yaw with Z), ideally using a calibration platform. Then, with the

Table 1
Sensor's data discretization conditions. Subindexes: i = finger index, j = hand axis (RPY), k = world axis (XYZ), t = current timestep (by default). Other notation: H^A_B = homogeneous transformation matrix from A to B, composed of R (rotation matrix), and p (translation vector), P_t = position of the hand at timestep t, φ = threshold (its subindex specifies the sensor's data for which it is applied).

Primitive variable	Discretization conditions
Fingers bending (FB)	$\forall i, FB_i = \begin{cases} \text{Open}, & \text{if } fb_i \geq \varphi_{b_{i,1}}; \\ \text{Half closed}, & \text{if } \varphi_{b_{i,1}} > fb_i \geq \varphi_{b_{i,2}}; \\ \text{Closed}, & \text{otherwise} \end{cases}$
Fingers pressure (FP)	$\forall i, FP_i = \begin{cases} \text{Strong contact}, & \text{if } fp_i \geq \varphi_{p_i}; \\ \text{Contact}, & \text{if } \varphi_{p_{i,1}} > fp_i > 0; \\ \text{No contact}, & \text{otherwise} \end{cases}$
Hand gesture (HG)	$FB_i = \begin{cases} \text{Opening}, & \text{if } \frac{\sum_{i=1}^{5}((fb_i)_t-(fb_i)_{t-3})}{5} > th_{HR}; \\ \text{Closing}, & \text{if } \frac{\sum_{i=1}^{5}((fb_i)_t-(fb_i)_{t-3})}{5} < -th_{HR}; \\ \text{No change}, & \text{otherwise} \end{cases}$
Hand rotation (HR)	$R^{t-1}_t = (R^{world}_{hand})^{-1}_{t-1} \cdot (H^{world}_{hand})_t$ $\forall j, HR_j = \begin{cases} +\text{Rotation}, & \text{if R\_to\_Euler}(R^{t-1}_t)[j] \geq \varphi_r; \\ -\text{Rotation}, & \text{if R\_to\_Euler}(R^{t-1}_t)[j] \leq -\varphi_r; \\ \text{No rotation}, & \text{otherwise} \end{cases}$
Hand orientation (HO)	$R = (R^{world}_{hand})_t$ $Al_j = \underset{j}{argmax}(\lvert R[j,Z]\rvert)$ $HO = \begin{cases} +Al_j \text{ axis aligned with } Z, & \text{if } R[Z,Al_j] \geq \varphi_o; \\ \text{-}\ Al_j \text{ axis aligned with } Z, & \text{if } R[Z,Al_j] \leq -\varphi_o; \\ \text{Random hand orientation}, & \text{otherwise} \end{cases}$
Hand XYZ motion (WM)	$P^{t-n}_t = \lvert P_t - P_{t-n}\rvert$ $\forall k, WM_k = \begin{cases} \text{Fast motion}, & \text{if } P^{t-1}_t[k] \geq 0.5\,\varphi_m; \\ \text{Motion}, & \text{if } 0.5\,\varphi_m \geq P^{t-1}_t[k] \geq \varphi_m \\ & \quad \text{or } P^{t-2}_t[k] \geq 1.5\,\varphi_m \\ & \quad \text{or } P^{t-3}_t[k] \geq 1.75\,\varphi_m; \\ \text{No motion}, & \text{otherwise} \end{cases}$
Hand RPY motion (HM)	Equivalent equations than for WM, with: $P^{t-n}_t = ((H^{world}_{hand})^{-1}_{t-n} \cdot (H^{world}_{hand})_t)[\text{p}]$

sensor readings (R^{glove}_{hand}) and the known hand orientation ($R^{world}_{hand} = \mathbb{I}$), the conversion matrix can be obtained. These matrices are finally sent to the Discretizing module that can use them to correct the sensor

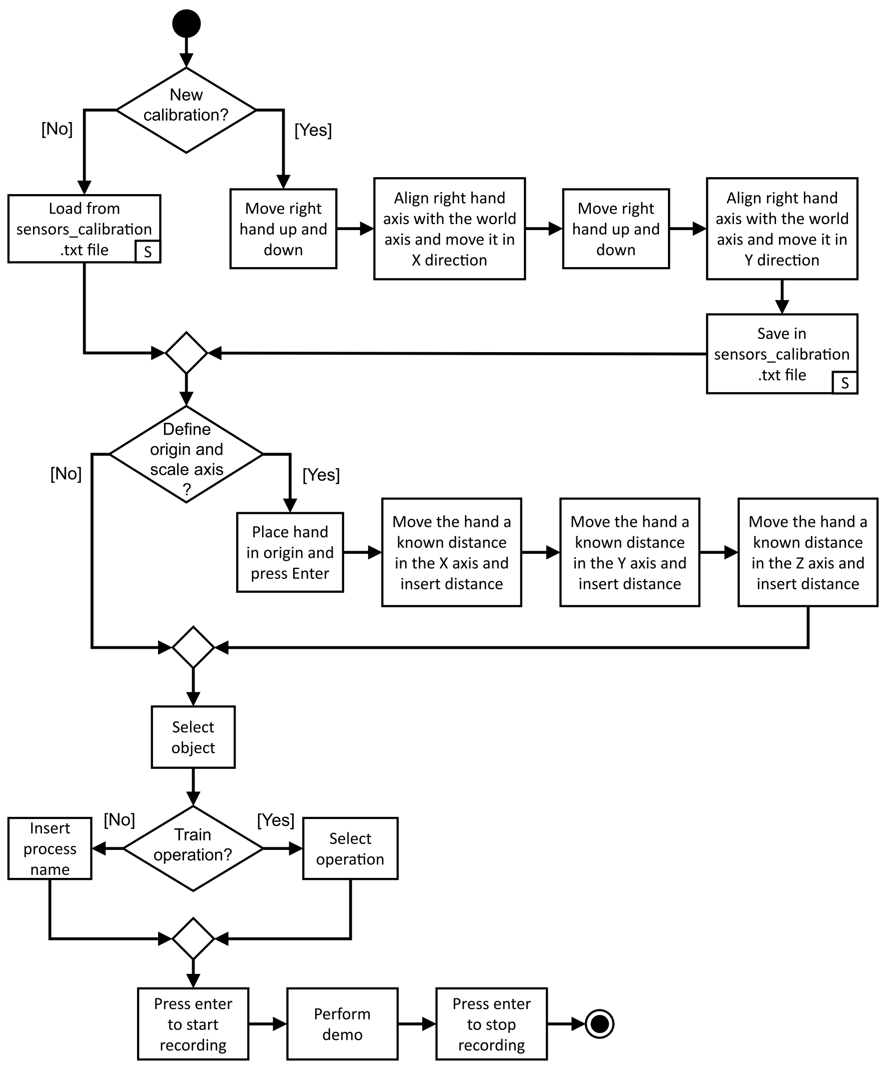

New calibration?
[No]
[Yes]
Load from sensors_calibration .txt file
S
Move right hand up and down
Align right hand axis with the world axis and move it in X direction
Move right hand up and down
Align right hand axis with the world axis and move it in Y direction
Save in sensors_calibration .txt file
S
Define origin and scale axis ?
[No]
[Yes]
Place hand in origin and press Enter
Move the hand a known distance in the X axis and insert distance
Move the hand a known distance in the Y axis and insert distance
Move the hand a known distance in the Z axis and insert distance
Select object
Train operation?
[No]
[Yes]
Insert process name
Select operation
Press enter to start recording
Perform demo
Press enter to stop recording

Fig. 10. UML activity diagram of the user steps required to calibrate the sensors and record a demonstration. S: the activity is performed automatically by the system.

readings.

$$R^{world}_{glove} = R^{world}_{hand} \cdot R^{glove^{-1}}_{hand}$$

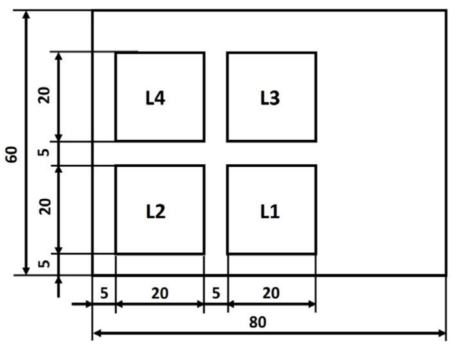
The second step is aligning the origin of the coordinate frames of the devices with the origin of the working platform. For this, the user needs to place the hand in the platform's origin, and this position is defined as the new zero. Finally, the tracker readings in the three axes have to be scaled to correct small deviations. This is done by moving the hand a known distance along each axis and calculating the ratio between this distance and the one obtained with the tracker. The placement of the hand in this step does not need to be extremely precise, as the goal of the system is not to record trajectories but to recognize patterns of motion.

Besides calibrating the sensors, it is necessary to determine the translation vectors from the tracker (placed on the user's wrist) to the objects' actuation frames (i.e., the part of the object that executes the operation), so the system can detect in which locations the operations are performed. This is known as offline calibration, as it just needs to be done once for each operation of each object (as objects can be grasped with a different orientation in each operation). The calibration translation vector of each operation is defined as the tracker's position (with reversed signs) when the actuation frame of the object is placed at the origin of the working platform. However, to make the calibration independent of the object's size, this vector is normalized. Hence, to calibrate an object's position, the calibration vector of the recognized operation has to be multiplied by the object's length.

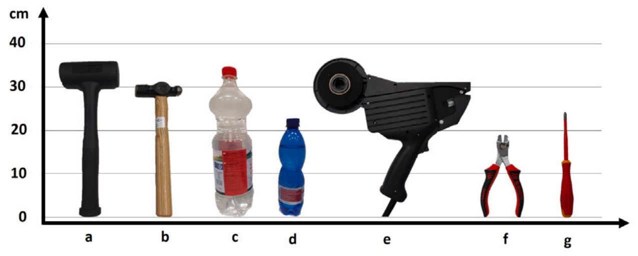
4. Experimental evaluation

The system was tested in a workbench with four location areas for five different objects: two bottles of different sizes, two hammers of different sizes, a screwdriver, a taping gun and cutting pliers (see Fig. 11). The system was trained with 40 demonstrations for each operation, performed by four users, three men and a woman, with different hand and arm sizes. After calibrating the sensing devices, the average time to record an operation demonstration is approximately fifteen seconds. Thus, to achieve the results presented in this section, every operation was trained in less than fifteen minutes. The following subsections present and analyze the results obtained when recognizing operations, processes and the operations' locations (Table 2 shows all the objects, operations and processes used for testing the system).

Regarding the models used in the experiments, these were generated using the following primitive variables: *hand_xyz*, the sequence of hand movements in the world axis; *hand_z*, the sequence of hand movements in the gravity world axis (positive or negative, fast or slow, long or short); *hand_rotation*, hand rotations sequence in the hand axis considering also their speed; *hand_orientation*, the sequence of the hand axis aligned with the world Z axis; *finger_bending*, sequence of fingers bending discrete levels; *finger_pressure*, sequence of fingers pressure discrete levels; *demo_long*, the sequence of the primitives of all the previous files; *demo_short*, a simplified sequence of the primitives of all the previous files (e.g., hand motion instead of hand motion in +Z



(a) Testing workbench drawing with 4 locations. Dimensions in cm.



(b) Testing objects; a: big hammer, b: medium hammer, c: big bottle, d: small bottle, e: taping gun, f: cutting pliers, g: screwdriver.

Fig. 11. Material used for the experiments.

Table 2
Objects, operations and processes used for testing. Additionally, each individual operation was also considered as a process for testing the system.

Object	Operations	Processes
Bottle	Grasp (G), Place (P) Grasp top (GT), Place top (PT), Pour water (PW)	G-PW-P G-P GT-PT
Screwdriver	Grasp (G), Place (P), Screw strong (SS), Screw light (SL)	G-SL-SS-P G-SS-P G-SL-P G-P
Hammer	Grasp (G), Place (P), Hammer (H)	G-K-P G-P
Taping gun	Grasp (G), Place (P), Tape (T)	G-T-P G-P
Cutting pliers	Grasp (G), Place (P), Cut (C)	G-C-P G-P

axis). Fig. 12 shows an instance of the information model, populated for the experimental evaluation.

4.1. Recognition of individual operations

The operation recognition was tested for each object by cross validation, using all the demonstrations of all its operations. Fig. 13 shows the results for a bottle using the 1st, 2nd and 3rd order Markov models. The same weight was assigned to all the variables' models used for each of these models. These results show an improvement in the operations recognition considering the two previous states for the transitions (2nd order models), instead of the conventional 1st order Markov Models. However, the performance drops when considering the three previous states (3rd order). Therefore, it can be concluded that increasing the order of the models (i.e., the number of previous states considered for the transitions), the differences between similar operations increases, but it can also make the recognition less robust against deviations from the normal motion pattern, needing many training demonstrations to have robust and generic models. Due to this, depending on the variable, it will be better to consider a higher or a lower order to be robust but at the same time detect the differences between operations. Additionally, it can be observed that with the three models, the *Place top* operation is frequently confused with the *Grasp top* operation, this was an expected result as these two operations have the same motion pattern, being the only difference between them the hand opening or closing.

Fig. 14 shows the confusion matrix of all the tested objects. In this case, the order of the models used and the variables' weights were already optimized to maximize the process segmentation performance. The single operation recognition performance does not have to necessarily improve with these models, as the optimization was made for the processes. However, this confusion matrix shows a more realistic representation of how the operations will be recognized when performed in a sequence. In general, the results show a good recognition of individual operations with a 76.4% correct recognition average. The poorest recognition is for the screwdriver operations, where *Grasp* and *Place* are confused with *Screw light*, however, both screwing operations (*Screw strong*, with a wrist movement, and *Screw light*, with the fingers movement) are successfully differentiated. Additionally, as was also seen in Fig. 13, the *Place top* operation of the bottle object is sometimes confused with *Grasp top*. The same happens with the *Grasp* and *Place* operations for the hammer because, again, these are operations with a very similar motion pattern that just differ on the hand opening or closing.

4.2. Segmentation and recognition of processes

An average of 40 processes were recorded for each object for testing the process segmentation and classification, as well as the operations location recognition. The processes performed are indicated in Table 2, including also each of the individual operations as additional processes, to test the ability of the system to detect when just one operation was executed. In order to test the performance of the system, several metrics were defined: $op\_perfect$, the percentage of processes perfectly segmented; op_100, the percentage of processes for which all the operations were recognized, independently of their order and the wrong detections; op_50, the percentage of process for which at least a 50% of its operations were recognized; $wrong\_op$, the percentage of incorrect segmented operations; $loc\_perfect$, the percentage of processes in which the sequences of locations is correctly recognized; loc_100, the percentage of processes for which all the locations were recognized, independently of their order and the wrong detections; $proc\_classification$, the percentage of processes correctly classified; $proc\_perfect$, the percentage of processes perfectly recognized (i.e., when both operations and locations are correct); and $proc\_classification5$ and $proc\_perfect5$, that are the same than the two previous metrics when providing five demonstrations of the process, instead of just one. Additionally, a metric was defined for the optimization of the weight assigned to the model of each variable. This metric is called *performance* (P) and it is calculated as follows:

$$P = \frac{3 \cdot op\_perfect + 2 \cdot op\_100 + op\_50 - 2 \cdot wrong\_op}{6}$$

Thus, the weights of the variables' models are calculated using an iterative optimization algorithm that maximizes P. The adjusted weights are shown in Fig. 15. As can be seen, not all the variables have the same relevance in the motion patterns of each object. For instance, the hand orientation has a high weight for the bottle, as it helps to distinguish between G and GT, and P and PT operations, depending on whether the Z world axis is aligned with the hand pitch or yaw axis, respectively. The same happens for the screwdriver object, where the hand orientation can be critical to differentiate the SS operation,

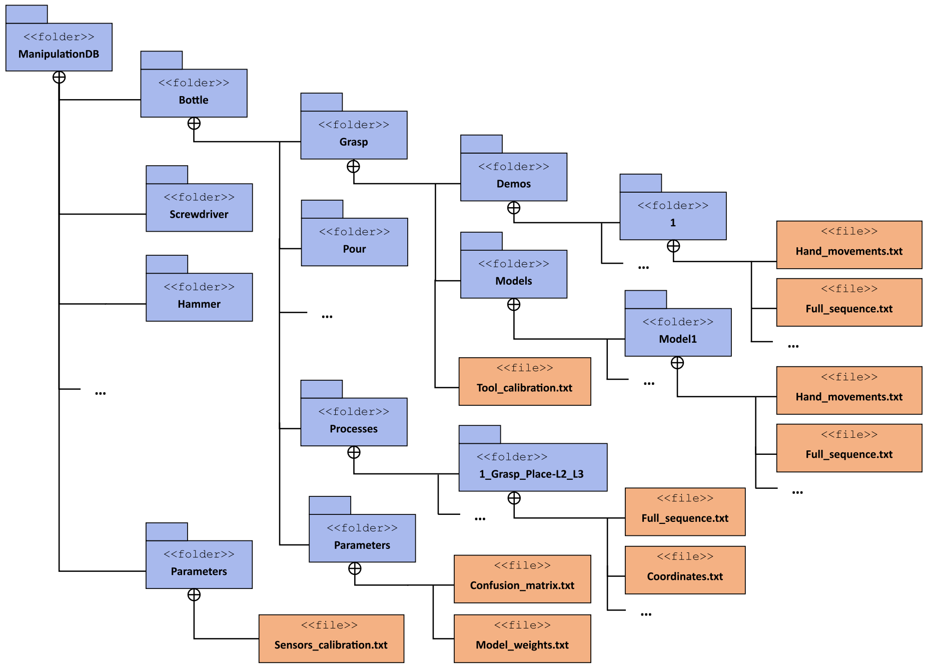
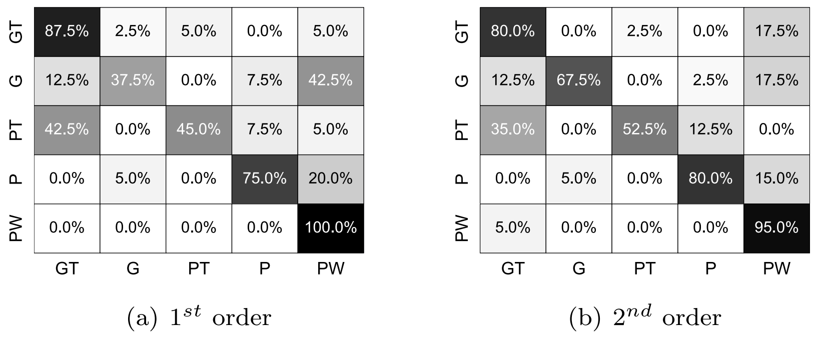

<<folder>> ManipulationDB
<<folder>> Bottle
<<folder>> Screwdriver
<<folder>> Hammer
...
<<folder>> Parameters
<<folder>> Grasp
<<folder>> Pour
...
<<folder>> Processes
<<folder>> Parameters
<<folder>> Demos
<<folder>> Models
<<file>> Tool_calibration.txt
<<folder>> 1_Grasp_Place-L2_L3
<<file>> Confusion_matrix.txt
<<file>> Model_weights.txt
<<file>> Sensors_calibration.txt
<<folder>> 1
<<folder>> Model1
<<file>> Hand_movements.txt
<<file>> Full_sequence.txt
<<file>> Full_sequence.txt
<<file>> Coordinates.txt

Fig. 12. Information model instance.

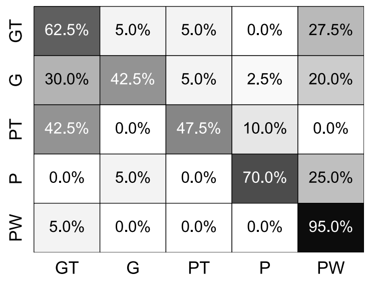


(a) 1st order

	GT	G	PT	P	PW
GT	87.5%	2.5%	5.0%	0.0%	5.0%
G	12.5%	37.5%	0.0%	7.5%	42.5%
PT	42.5%	0.0%	45.0%	7.5%	5.0%
P	0.0%	5.0%	0.0%	75.0%	20.0%
PW	0.0%	0.0%	0.0%	0.0%	100.0%

(b) 2nd order

	GT	G	PT	P	PW
GT	80.0%	0.0%	2.5%	0.0%	17.5%
G	12.5%	67.5%	0.0%	2.5%	17.5%
PT	35.0%	0.0%	52.5%	12.5%	0.0%
P	0.0%	5.0%	0.0%	80.0%	15.0%
PW	5.0%	0.0%	0.0%	0.0%	95.0%



(c) 3rd order

	GT	G	PT	P	PW
GT	62.5%	5.0%	5.0%	0.0%	27.5%
G	30.0%	42.5%	5.0%	2.5%	20.0%
PT	42.5%	0.0%	47.5%	10.0%	0.0%
P	0.0%	5.0%	0.0%	70.0%	25.0%
PW	5.0%	0.0%	0.0%	0.0%	95.0%

Fig. 13. Confusion matrix of the bottle operations recognition using Multivariate Markov Models of different orders.

exerted by the wrist, and SL were the orientation is constant and the fingers move. However, the weight of this model is much lower for the rest of objects. It can also be seen that the demo_long model, which provides information of all the variables in the same sequence, has almost always the highest weight, indicating that this is probably the most important model. Finally, in the legend of the figure we can see that, in almost all cases, second order models were selected. This was expected according to the recognition of individual operations tests, that showed the best results when considering the two previous states for the transitions (see Fig. 13).

Using these weights, the starting and transition probabilities for each operation of the manipulated object are calculated along the demonstration timeline, following Algorithm 1. Based on these probabilities, the demonstration can be segmented in a sequence of operations, determining also the exact moment when each operation starts. This can be seen in Fig. 16 for a $Grasp$–$Hammer$–$Place$ process.

Fig. 17 summarizes the performance of the system, showing the performance metrics obtained for each object. As expected, the screwdriver presents the worst results, as its individual operation recognition was not very good (see Fig. 14). However, without considering the

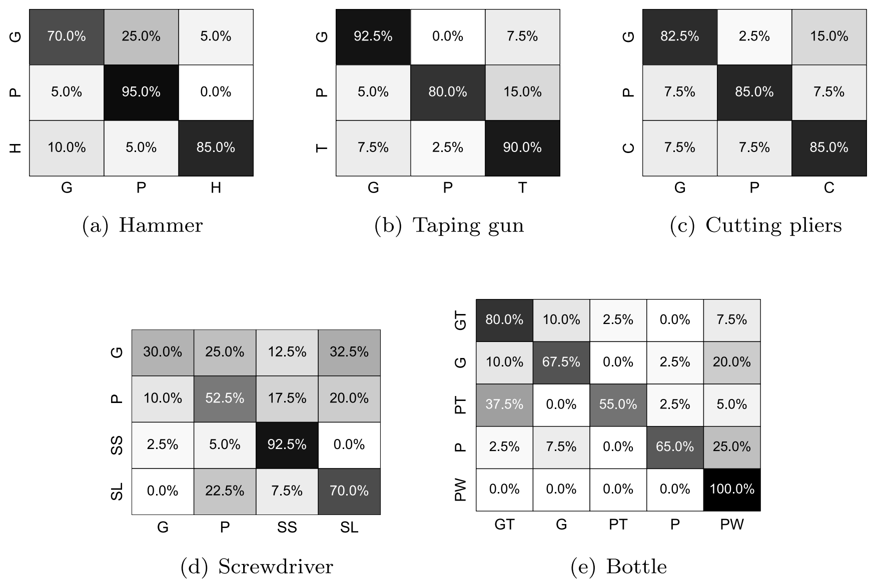


Fig. 14. Confusion matrix of the operations recognition of the tested objects.

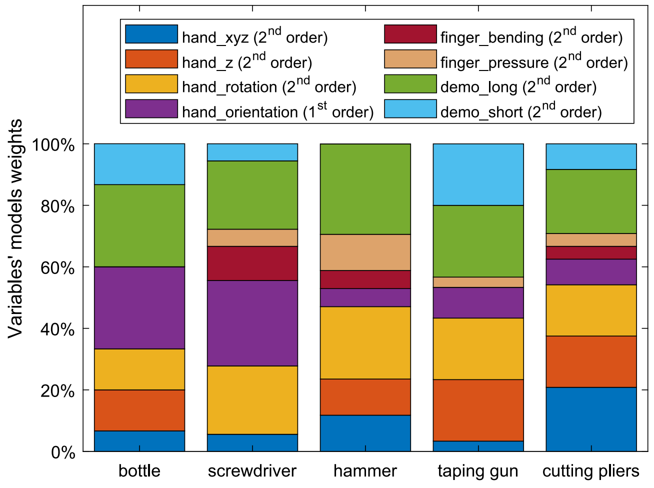


Fig. 15. Weight of the models of each variable optimized for each object. These weights were determined using an iterative optimization algorithm. In the legend, in brackets, the order of each model is specified.

screwdriver, the average of processes that were perfectly segmented is almost 40% and, without taking into account the sequence order, all the operations were detected in an average of almost 70% of the processes. This average increases to over 95% for the processes where at least a 50% of the operations are detected.

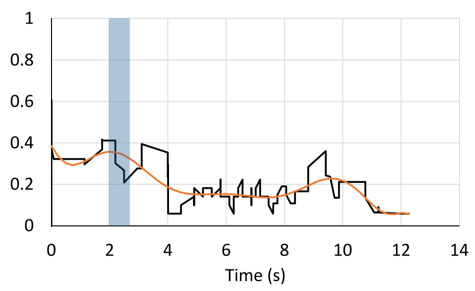
Regarding the location recognition, the calculated metrics depend on how good the segmentation is. Thus, if the operations are not detected in the correct time (or not detected at all), the coordinates of the hand during that time step will not match with the location where they were actually performed. Additionally, if an operation starts with the hand in one location and finish in other, the recognized location will be the one at the beginning, which in most of the cases is incorrect. This is why, the perfect recognition of the locations of a process should not be higher that the perfect recognition of its operations, however, this can be the case if the timeline segmentation is good but the detected operations are confused. The average number of processes with a perfect recognition of locations is around 35%, a bit lower than the perfectly segmented processes. Fig. 18 shows how the wrist and hand frame are visualized in RVIZ at the beginning of the *Grasp* operation of a bottle.

Finally, it can be observed that the system performance improves with the *Sequencing* module, which classifies the segmentations as the most similar processes. The screwdriver results are clearly the worst again, which is derived from its bad segmentation. However, there is also an improvement with respect to the perfectly segmented processes, meaning that even with an incorrect segmentation the processes can be identified. Excluding the screwdriver from the analysis, we can see that the average of correctly classified processes increases over 70%. Regarding the processes correctly classified with the correct locations, a precision of almost 50% is achieved in average, without considering the screwdriver. This last metric indicates the precision of the system for perfectly understanding the recorded manipulation. Additionally, the results show that these two metrics improve even more when performing five demonstrations of the same process. In the case of the screwdriver there is not improvement, as even if the user performs many demonstrations, if most of them are incorrect, the average will not necessarily improve the results. However, in the rest of the objects, the process recognition improves on average to almost 90% and to 75% when considering also location recognition.

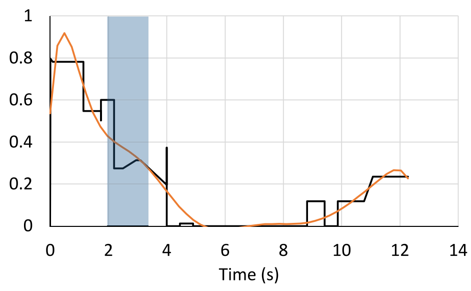
5. Conclusions and future directions

Robots are flexible machines that can perform multiple processes just by changing their program. To exploit this capability, we need tools and systems that allow the fast and easy programming and reconfiguration of the robot. With this aim, in this paper a ROS-based high-level PbD system is proposed, that allows the understanding and digitizing of single-arm human manipulation processes, creating an action plan that later could be interpreted by a robot. The system is composed of five modules intercommunicated, three of them in charge of processing and converting the information from the low-level sensor data to a high-level manipulation process; one for training the system, creating Multiorder Multivariate Markov Models for each operation; and another for interfacing and calibrating it.

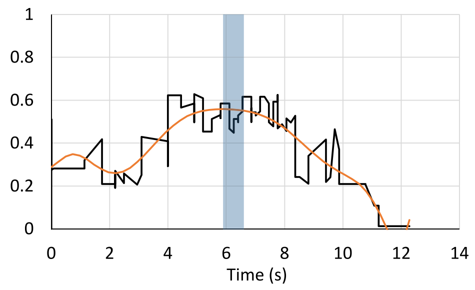
The system was implemented and tested using a hand position tracker and dataglove with a one degree of freedom bending sensor in each finger, a pressure sensor in each fingertip and several sensors to determine the hand orientation. The experiments were run by four users with five different types of objects, checking the performance of the system for the individual operations recognition as well as the process segmentation and classification. The obtained results show a very good performance recognizing individual operations, with an average



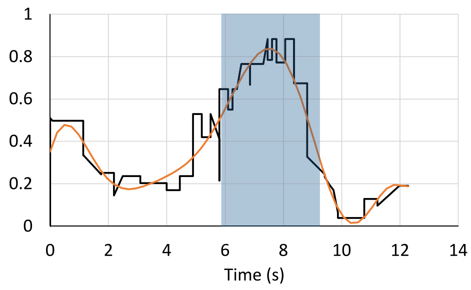
(a) Initial state probability timeline of the Grasp operation



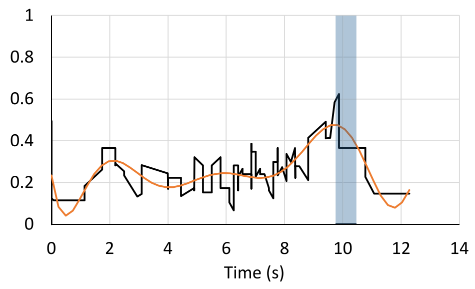
(b) Transition probability timeline of the Grasp operation



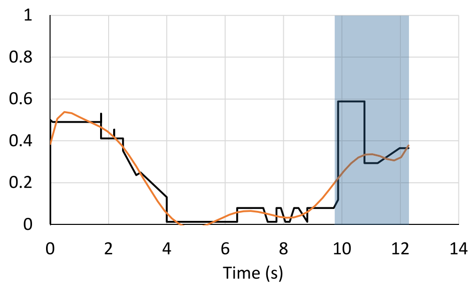
(c) Initial state probability timeline of the Hammer operation



(d) Transition probability timeline of the Hammer operation



(e) Initial state probability timeline of the Place operation



(f) Transition probability timeline of the Place operation

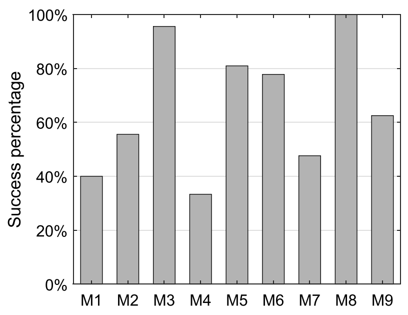
Fig. 16. Initial state and transition Markov probability timelines of the different hammer's operations during a *Grasp–Hammer–Place* process. The black lines are obtained directly from the Markov probabilities computation, whereas the orange ones are a polynomial regression of the black ones, and are used for the segmentation computation. The blue background areas indicate when the beginning of the operation is detected (in the maximums of the regression curves of the initial state probability graphs), and when the execution of the operation is detected (in the transition probability graphs).

recognition accuracy of around 80%. Regarding process segmentation, the accuracy decreases a bit, as it depends strongly on a good operation identification. One of the objects, the screwdriver, clearly shows worse results than the others, however, for the rest, the segmentation is perfect in around 40% of the times, increasing to around 70% if the order of the detected operations is not taken into account. This means that the segmentation provides good information about the process but it is not always perfect. Therefore, the performance and robustness of the system improves considerably classifying the segmentation as the most similar process, from a list of possible processes with that object. With this, the accuracy of the system is doubled for every object, and almost tripled when five demonstrations are provided for every process.

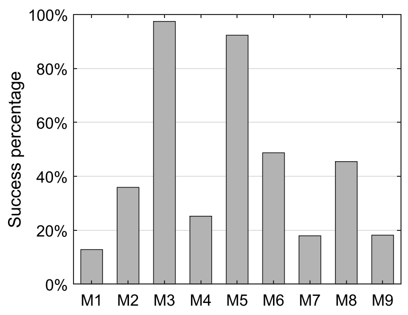
Additionally, the system identifies the locations in which the operations are performed. This depends a lot on how good the segmentation is, as the location picked is the one where the hand is placed at the moment of the segmentation of each operation. Therefore, the results obtained are a bit lower than the ones of the segmentation. However, these results are improved after the process classification and the use of several demonstrations, reaching a perfect process and locations recognition of around 75% for most of the objects when providing five demonstrations.

The system is scalable and can be used to identify any manipulation process performed with any object, provided it has been trained for its constituent operations. To define a new object or operation in the system, the user just needs to create a new folder with the object or operation's name according to the information system's model and populate it with new demonstrations. The time required to train the system for a new operation depends on the number of demonstrations provided, typically yielding improved performance with a greater number of demonstrations. As a reference, the results presented in this manuscript were achieved after training every operation with 40 demonstrations, which took between 10 and 15 min per operation. Once the system is trained, the time required to generate a new high-level robot program is just the time required to demonstrate the new process.

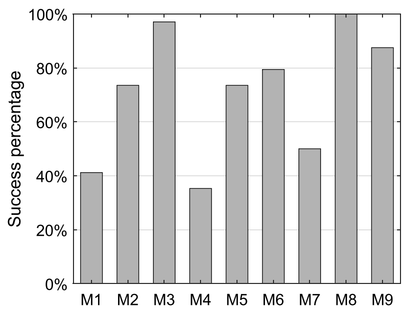
The promising results obtained after this initial implementation of the system open many future lines of research, not just for enhancing and extending it, but also for applying its outcomes for new applications. Regarding the extension of the system, an interesting line of



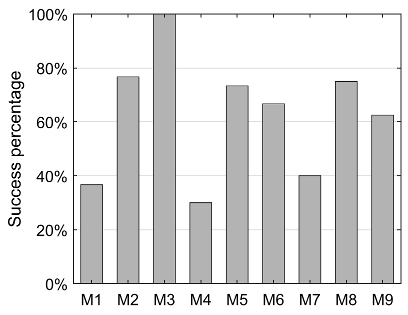
(a) Bottle



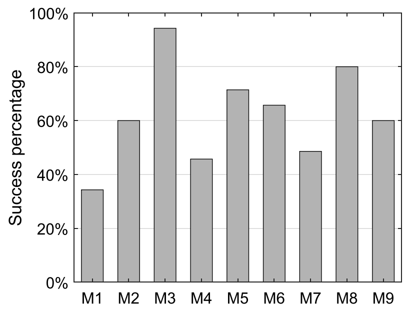
(b) Screwdriver



(c) Hammer



(d) Taping gun



(e) Cutting pliers

Fig. 17. System performance metrics for the different objects. M1: $op\_perfect$, M2: op_100, M3: op_50, M4: $loc\_perfect$, M5: loc_100, M6: $proc\_classification$, M7: $proc\_perfect$, M8: $proc\_classification\_5$, M9: $proc\_perfect\_5$. These metrics are described in Section 4.2.

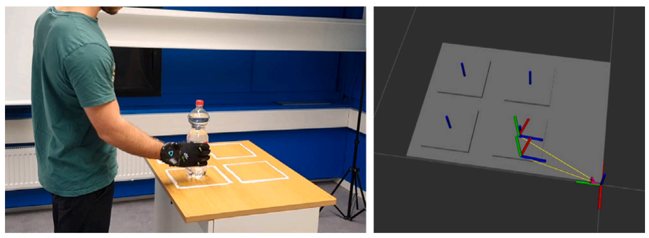


Fig. 18. User grasping a bottle in location L1 of the workbench and RVIZ visualization of the wrist and hand frames.

research could be the inclusion of additional sensors that could capture important manipulation information that currently is missing, such as the hand contact points, the abduction of the fingers or the status of the manipulated object. Additionally, the system could be extended to recognize also dual-arm operations and processes. As to the system applications, the main future line is transferring the digitized operator knowledge to a robotic manipulator. A possible approach for this was already presented by the authors in [56] as a proof of concept. This approach consists of preprogramming all the possible operations as parameterized functions, which are denominated skills. The input of this system must be a high-level action plan that specifies the sequence in which these skills have to be called, and a set of parameters for each of them, which configure them to generate the optimal trajectories and end effectors' actions. Thus, this approach could be combined with the proposed PbD system, using the recognized sequence of operations to determine which skills to execute, and the recognized locations as the parameters sent to configure these skills. This is a simple case, but the same approach could be scaled to very complex applications. Therefore, the goal is to keep working on this line, extending the PbD system to extract more parameters from the recognized operations (e.g., how much force to apply, how many degrees to rotate, etc.), which will allow the definition of more complex and generic skills.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Pablo Malvido Fresnillo reports financial support was provided by Horizon 2020. Saigopal Vasudevan reports financial support was provided by Horizon 2020.

Data availability

Data will be made available on request.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement n° 870133, correspondent to the project entitled REMODEL, Robotic tEchnologies for the Manipulation of cOmplex DeformablE Linear objects.

References

[1] B. Singh, K. P, Evolution of industrial robots and their applications, Int. J. Emerg. Technol. Adv. Eng. 3 (2013) 763–768.

[2] P.M. Fresnillo, S. Vasudevan, W.M. Mohammed, J.L. Martinez Lastra, G. Laudante, S. Pirozzi, K. Galassi, G. Palli, Deformable objects grasping and shape detection with tactile fingers and industrial grippers, in: 2021 IEEE Conference on Industrial Cyberphysical Systems, ICPS, 2021, pp. 525–530, http://dx.doi.org/10.1109/ICPS49255.2021.9468151.

[3] G.Q. Zhang, X. Li, R. Boca, J. Newkirk, B. Zhang, T.A. Fuhlbrigge, H.K. Feng, N.J. Hunt, Use of industrial robots in additive manufacturing - a survey and feasibility study, in: ISR/Robotik 2014; 41st International Symposium on Robotics, 2014, pp. 1–6.

[4] M. Kyrarini, F. Lygerakis, A. Rajavenkatanarayanan, C. Sevastopoulos, H.R. Nambiappan, K.K. Chaitanya, A.R. Babu, J. Mathew, F. Makedon, A survey of robots in healthcare, Technologies 9 (1) (2021) 8, http://dx.doi.org/10.3390/technologies9010008, Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[5] G. Ajaykumar, M. Steele, C.-M. Huang, A survey on end-user robot programming, ACM Comput. Surv. 54 (8) (2021) 164:1–164:36, http://dx.doi.org/10.1145/3466819.

[6] V. Villani, F. Pini, F. Leali, C. Secchi, C. Fantuzzi, Survey on human-robot interaction for robot programming in industrial applications, IFAC-PapersOnLine 51 (11) (2018) 66–71, http://dx.doi.org/10.1016/j.ifacol.2018.08.236.

[7] Z. Zhou, R. Xiong, Y. Wang, J. Zhang, Advanced robot programming: a review, Curr. Robot. Rep. 1 (4) (2020) 251–258, http://dx.doi.org/10.1007/s43154-020-00023-4.

[8] R. Chitnis, S. Tulsiani, S. Gupta, A. Gupta, Efficient bimanual manipulation using learned task schemas, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, (ISSN: 2577-087X) 2020, pp. 1149–1155, http://dx.doi.org/10.1109/ICRA40945.2020.9196958.

[9] W. Szynkiewicz, Skill-based bimanual manipulation planning, J. Telecommun. Inf. Technol. nr 4 (2012) 54–62.

[10] A. Sylari, B.R. Ferrer, J.L.M. Lastra, Hand gesture-based on-line programming of industrial robot manipulators, in: 2019 IEEE 17th International Conference on Industrial Informatics, Vol. 1, INDIN, (ISSN: 2378-363X) 2019, pp. 827–834, http://dx.doi.org/10.1109/INDIN41052.2019.8972301.

[11] S. Ekvall, D. Kragic, Grasp recognition for programming by demonstration, in: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, (ISSN: 1050-4729) 2005, pp. 748–753, http://dx.doi.org/10.1109/ROBOT.2005.1570207.

[12] E.E. Aksoy, A. Orhan, F. Wörgötter, Semantic decomposition and recognition of long and complex manipulation action sequences, Int. J. Comput. Vis. 122 (1) (2017) 84–115, http://dx.doi.org/10.1007/s11263-016-0956-8.

[13] K. Ramirez-Amaro, E.-S. Kim, J. Kim, B.-T. Zhang, M. Beetz, G. Cheng, Enhancing human action recognition through spatio-temporal feature learning and semantic rules, in: 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids), (ISSN: 2164-0580) 2013, pp. 456–461, http://dx.doi.org/10.1109/HUMANOIDS.2013.7030014.

[14] I.S. Vicente, V. Kyrki, D. Kragic, M. Larsson, Action recognition and understanding through motor primitives, Adv. Robot. 21 (15) (2007) 1687–1707, http://dx.doi.org/10.1163/156855307782506156, Publisher: Taylor & Francis _eprint.

[15] K. Bernardin, K. Ogawara, K. Ikeuchi, R. Dillmann, A hidden markov model based sensor fusion approach for recognizing continuous human grasping sequences, in: Proc. 3rd IEEE International Conference on Humanoid Robots, 2003, pp. 1–13.

[16] K. Ogawara, J. Takamatsu, H. Kimura, K. Ikeuchi, Modeling manipulation interactions by hidden Markov models, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 2, 2002, pp. 1096–1101 vol.2, http://dx.doi.org/10.1109/IRDS.2002.1043877.

[17] P. Neto, M. Simão, N. Mendes, M. Safeea, Gesture-based human-robot interaction for human assistance in manufacturing, Int. J. Adv. Manuf. Technol. 101 (1) (2019) 119–135, http://dx.doi.org/10.1007/s00170-018-2788-x.

[18] T. Lozano-Perez, Robot programming, Proc. IEEE 71 (7) (1983) 821–841, http://dx.doi.org/10.1109/PROC.1983.12681, Conference Name: Proceedings of the IEEE.

[19] P. Tsarouchi, A. Athanasatos, S. Makris, X. Chatzigeorgiou, G. Chryssolouris, High level robot programming using body and hand gestures, Procedia CIRP 55 (2016) 1–5, http://dx.doi.org/10.1016/j.procir.2016.09.020.

[20] C. Archibald, E. Petriu, Model for skills-oriented robot programming (SKORP), in: Applications of Artificial Intelligence 1993: Machine Vision and Robotics, Vol. 1964, SPIE, 1993, pp. 392–402, http://dx.doi.org/10.1117/12.141787.

[21] F. Rovida, M. Crosby, D. Holz, A.S. Polydoros, B. Großmann, R.P.A. Petrick, V. Krüger, Skiros—A skill-based robot control platform on top of ROS, in: A. Koubaa (Ed.), Robot Operating System (ROS): the Complete Reference (Volume 2), in: Studies in Computational Intelligence, Springer International Publishing, Cham, 2017, pp. 121–160, http://dx.doi.org/10.1007/978-3-319-54927-9_4.

[22] R. Arrais, G. Veiga, T.T. Ribeiro, D. Oliveira, R. Fernandes, A.G.S. Conceição, P.C.M.A. Farias, Application of the open scalable production system to machine tending of additive manufacturing operations by a mobile manipulator, in: Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3–6, 2019, Proceedings, Part II, Springer-Verlag, Berlin, Heidelberg, 2019, pp. 345–356, http://dx.doi.org/10.1007/978-3-030-30244-3_29.

[23] B.D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (5) (2009) 469–483, http://dx.doi.org/10.1016/j.robot.2008.10.024.

[24] B. Dufay, J.-C. Latombe, An approach to automatic robot programming based on inductive learning, Int. J. Robot. Res. 3 (4) (1984) 3–20, http://dx.doi.org/10.1177/027836498400300401, Publisher: SAGE Publications Ltd STM.

[25] M. Ogino, H. Toichi, Y. Yoshikawa, M. Asada, Interaction rule learning with a human partner based on an imitation faculty with a simple visuo-motor mapping, Robot. Auton. Syst. 54 (5) (2006) 414–418, http://dx.doi.org/10.1016/j.robot.2006.01.005.

[26] B. Akgun, M. Cakmak, J.W. Yoo, A.L. Thomaz, Trajectories and keyframes for kinesthetic teaching: a human-robot interaction perspective, in: Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 391–398, http://dx.doi.org/10.1145/2157689.2157815.

[27] W.M. Mohammed, P.M. Fresnillo, S. Vasudevan, Z. Gosar, J.L.M. Lastra, An approach for modeling grasping configuration using ontology-based taxonomy, in: 2020 IEEE Conference on Industrial Cyberphysical Systems, Vol. 1, ICPS, 2020, pp. 507–513, http://dx.doi.org/10.1109/ICPS48405.2020.9274760.

[28] D. Aarno, D. Kragic, Motion intention recognition in robot assisted applications, Robot. Auton. Syst. 56 (8) (2008) 692–705, http://dx.doi.org/10.1016/j.robot.2007.11.005.

[29] K. Kahol, P. Tripathi, S. Panchanathan, Automated gesture segmentation from dance sequences, in: Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings, 2004, pp. 883–888, http://dx.doi.org/10.1109/AFGR.2004.1301645.

[30] L. Gutzeit, E.A. Kirchner, Automatic detection and recognition of human movement patterns in manipulation tasks:, in: Proceedings of the 3rd International Conference on Physiological Computing Systems, SCITEPRESS - Science and Technology Publications, Lisbon, Portugal, 2016, pp. 54–63, http://dx.doi.org/10.5220/0005946500540063.

[31] H. Kjellström, J. Romero, D. Kragić, Visual object-action recognition: Inferring object affordances from human demonstration, Comput. Vis. Image Underst. 115 (1) (2011) 81–90, http://dx.doi.org/10.1016/j.cviu.2010.08.002.

[32] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286, http://dx.doi.org/10.1109/5.18626, Conference Name: Proceedings of the IEEE.

[33] E. Fosler-lussier, Markov Models and Hidden Markov Models: A Brief Tutorial, Technical Report TR-98-041, International Computer Science Institute, University of California, Berkeley, 1998.

[34] J. Bulla, F. Lagona, A. Maruotti, M. Picone, A multivariate hidden Markov model for the identification of sea regimes from incomplete skewed and circular time series, J. Agric. Biol. Environ. Stat. 17 (4) (2012) 544–567, http://dx.doi.org/10.1007/s13253-012-0110-1.

[35] J. Picone, Continuous speech recognition using hidden Markov models, IEEE ASSP Mag. 7 (3) (1990) 26–41, http://dx.doi.org/10.1109/53.54527, Conference Name: IEEE ASSP Magazine.

[36] W. Cho, S.-W. Lee, J.H. Kim, Modeling and recognition of cursive words with hidden Markov models, Pattern Recognit. 28 (12) (1995) 1941–1953, http://dx.doi.org/10.1016/0031-3203(95)00041-0.

[37] O. Aycard, F. Charpillet, D. Fohr, J.-F. Mari, Place learning and recognition using hidden Markov models, in: Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications, Vol. 3, IROS '97, 1997, pp. 1741–1747 vol.3, http://dx.doi.org/10.1109/IROS.1997.656595.

[38] A. Bayoumi, P. Karkowski, M. Bennewitz, Speeding up person finding using hidden Markov models, Robot. Auton. Syst. 115 (2019) 40–48, http://dx.doi.org/10.1016/j.robot.2019.02.001.

[39] M. Field, Z. Pan, D. Stirling, F. Naghdy, Human motion capture sensors and analysis in robotics, Industrial Robot: An International Journal 38 (2) (2011) 163–171, http://dx.doi.org/10.1108/01439911111106372, Publisher: Emerald Group Publishing Limited.

[40] H. Liu, Z. Ju, X. Ji, C.S. Chan, M. Khoury, Human Motion Sensing and Recognition, Studies in Computational Intelligence, vol. 675, Springer Berlin Heidelberg, Berlin, Heidelberg, 2017, http://dx.doi.org/10.1007/978-3-662-53692-6.
[41] M. Hassan, T. Ahmad, A. Farooq, S.A. Ali, S.R. hassan, N. Liaqat, A review on human actions recognition using vision based techniques, J Image Graph. (2014) 28–32, http://dx.doi.org/10.12720/joig.2.1.28-32.
[42] L. Chen, H. Wei, J. Ferryman, A survey of human motion analysis using depth imagery, Pattern Recognit. Lett. 34 (15) (2013) 1995–2006, http://dx.doi.org/10.1016/j.patrec.2013.02.006.
[43] S. Han, S. Lee, A vision-based motion capture and recognition framework for behavior-based safety management, Autom. Constr. 35 (2013) 131–141, http://dx.doi.org/10.1016/j.autcon.2013.05.001.
[44] R. Lun, W. Zhao, A survey of applications and human motion recognition with microsoft kinect, Int. J. Pattern Recognit. Artif. Intell. 29 (05) (2015) 1555008, http://dx.doi.org/10.1142/S0218001415550083, Publisher: World Scientific Publishing Co.
[45] L. Sigal, A.O. Balan, M.J. Black, HumanEva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion, Int. J. Comput. Vis. 87 (1–2) (2010) 4–27, http://dx.doi.org/10.1007/s11263-009-0273-6.
[46] C. Ionescu, D. Papava, V. Olaru, C. Sminchisescu, Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments, IEEE Trans. Pattern Anal. Mach. Intell. 36 (7) (2014) 1325–1339, http://dx.doi.org/10.1109/TPAMI.2013.248, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.
[47] D.K. Noh, N.G. Lee, J.H. You, A novel spinal kinematic analysis using X-ray imaging and vicon motion analysis: A case study, Bio-Med. Mater. Eng. 24 (1) (2014) 593–598, http://dx.doi.org/10.3233/BME-130846, Publisher: IOS Press.
[48] H. Zhou, H. Hu, Human motion tracking for rehabilitation—A survey, Biomed. Signal Process. Control 3 (1) (2008) 1–18, http://dx.doi.org/10.1016/j.bspc.2007.09.001.
[49] Y. Qi, C.B. Soh, E. Gunawan, K.-S. Low, R. Thomas, Estimation of spatial-temporal gait parameters using a low-cost ultrasonic motion analysis system, Sensors 14 (8) (2014) 15434–15457, http://dx.doi.org/10.3390/s140815434, Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.
[50] S. Cho, J. Ku, Y.K. Cho, I.Y. Kim, Y.J. Kang, D.P. Jang, S.I. Kim, Development of virtual reality proprioceptive rehabilitation system for stroke patients, Comput. Methods Programs Biomed. 113 (1) (2014) 258–265, http://dx.doi.org/10.1016/j.cmpb.2013.09.006.
[51] R. Meattini, D. Chiaravalli, L. Biagiotti, G. Palli, C. Melchiorri, Combining unsupervised muscle co-contraction estimation with bio-feedback allows augmented kinesthetic teaching, IEEE Robot. Autom. Lett. 6 (4) (2021) 6180–6187, http://dx.doi.org/10.1109/LRA.2021.3092269, Conference Name: IEEE Robotics and Automation Letters.
[52] E.N. Kamavuako, D. Farina, K. Yoshida, W. Jensen, Relationship between grasping force and features of single-channel intramuscular EMG signals, J. Neurosci. Methods 185 (1) (2009) 143–150, http://dx.doi.org/10.1016/j.jneumeth.2009.09.006.
[53] L. Dipietro, A.M. Sabatini, P. Dario, A survey of glove-based systems and their applications, IEEE Trans. Syst. Man Cybern C 38 (4) (2008) 461–482, http://dx.doi.org/10.1109/TSMCC.2008.923862, Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews).
[54] J. Liu, F. Feng, Y.C. Nakamura, N.S. Pollard, A taxonomy of everyday grasps in action, in: 2014 IEEE-RAS International Conference on Humanoid Robots, (ISSN: 2164-0580) 2014, pp. 573–580, http://dx.doi.org/10.1109/HUMANOIDS.2014.7041420.
[55] T. Feix, J. Romero, H.-B. Schmiedmayer, A.M. Dollar, D. Kragic, The GRASP taxonomy of human grasp types, IEEE Trans. Hum.-Mach. Syst. 46 (1) (2016) 66–77, http://dx.doi.org/10.1109/THMS.2015.2470657, Conference Name: IEEE Transactions on Human-Machine Systems.
[56] P.M. Fresnillo, S. Vasudevan, W.M. Mohammed, An approach for the bimanual manipulation of a deformable linear object using a dual-arm industrial robot: cable routing use case, in: 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems, ICPS, Conventry, United Kingdom, 2022, pp. 1–8, http://dx.doi.org/10.1109/ICPS51978.2022.9816981.

Pablo Malvido Fresnillo is a Doctoral Researcher at Tampere University (Finland). He received a M.Sc. in Electronics and Automation Engineering from University of Vigo (Spain) in 2019 and a B.Sc. in Industrial Engineering from University of Vigo (Spain) in 2017. As he is pursuing a Doctoral Degree in Engineering Sciences, Mr. Malvido's research interests include Robotics, Programming by Demonstration, Robot Bimanual Manipulation, Knowledge Based Reasoning Engines and Factory Automation. In addition, Mr. Malvido is part of the FAST-Lab research group and, since 2020, he is working in the project REMODEL, part of the EU H2020 funding program, whose aim is the development and implementation of hardware and software technologies to manipulate complex Deformable Linear Objects (DLOs) using robotic manipulators.

Saigopal Vasudevan is a Project Researcher for the FAST-Lab research group, at Tampere University. He received a M.Sc. in Factory Automation and Industrial Engineering from Tampere University, Finland in 2019 and a B.E (bachelor's degree) in Mechanical Engineering from Anna University, India in 2016. Mr. Vasudevan has worked in the automation industry as a Roboticist for a year, between 2016 and 2017. He is currently working in the project REMODEL from the EU H2020 funding program, since 2019; where the aim is to develop hardware and software technologies to manipulate complex Deformable Linear Objects (DLOs) using robotic manipulators. And since 2020, he is also working in the project AISOLA of the INSO profiling action, which is funded by the academy of Finland; whose objective is to assess social isolation and to prevent the onset of perceived loneliness in older adults, with a focus on developing and utilizing dedicated AI systems. In addition to working as a researcher, he also involved in collaborating for proposals, representing Tampere University, for the Horizon funding program of the EU with considerable success. Mr. Vasudevan's interests include Robotics, Bimanual Robot Manipulation, AI systems and Industrial Informatics & Automation.

Wael M. Mohammed is a Doctoral Researcher at Tampere University. He received a M.Sc. in Automation Engineering from Tampere University of Technology in 2017 and a B.Sc. in Mechatronics Engineering from university of Jordan in 2010. As he is pursuing a Doctoral Degree in Engineering Sciences, Mr. Mohammed's research interests include Robotics, Digital Twins, Knowledge Based Reasoning Engines and Factory Automation. In addition, Mr. Mohammed has been involved in writing proposals for research and innovation project funded by the EU commission. In 2010, he worked as a research assistant in the Production Engineering Department at Tampere University of Technology. Then in 2011, Mr. Mohammed worked as a head of the technical department in the Traffic Management System project at Etihad Alafandi L.L.C. in Saudi Arabia.

Prof. Lastra joined Tampere University of Technology in 1997 and became University Full Professor in 2006. His research interest is in applying Information and Communication Technologies to the field of Automation. He leads the FAST-Lab. with the ultimate goal of seamlessly integrating the knowledge of humans and machines in order to create smart environments. Prof. Lastra has co/authored over 300 articles and holds a number of patents in the field of Industrial Informatics and Automation. Prof. Lastra is an active member within the international research scene with a track record of 21 funded European Research projects. He serves as Associate Editor of the IEEE Transactions on Industrial Informatics, and He was a Technical Editor of the IEEE/ASME Transactions on Mechatronics in 2015–2016. Prof. Lastra served as Guest Editor for the MDPI Sensors Journal, and is an editorial member of the MDPI Machines journal. Furthermore, Prof. Lastra served as Technical Secretary for the IEEE Technical Committee on Industrial Cyber–Physical Systems in 2017–2018. For the period 2019–2022, He served as a Co-Chair for the IEEE Technical Committee on Industrial CyberPhysical systems.

Dr. José A. Pérez García worked as a production engineer for 6 years before he joined University of Vigo (Spain) in 1998, where he became Associate Professor in 2009. His research interest is on CAD/CAM and manufacturing technologies, fields in which he has developed a number of collaborations with several Spanish companies. Dr. Perez has taught CAM courses in several Universities, both in Europe and America, and has co-authored around 40 scientific articles and conference papers. He holds a M.Sc. in Industrial Engineering, a M.Sc. in Labor Risk Prevention and a Ph.D. in Design & Manufacturing, all of them from University of Vigo.