Synthesis Lectures on Computer Vision

SYNTHESIS
COLLECTION OF TECHNOLOGY

Katsushi Ikeuchi · Naoki Wake ·
Jun Takamatsu · Kazuhiro Sasabuchi

Learning-from-Observation 2.0

Automatic Acquisition of Robot Behavior from Human Demonstration

Springer

Synthesis Lectures on Computer Vision

Katsushi Ikeuchi · Naoki Wake ·
Jun Takamatsu · Kazuhiro Sasabuchi

Learning-from-Observation 2.0

Automatic Acquisition of Robot Behavior from Human Demonstration

Springer

Katsushi Ikeuchi
Applied Robotics Research
Microsoft
Redmond, WA, USA

Naoki Wake
Applied Robotics Research
Microsoft
Redmond, WA, USA

Jun Takamatsu
Applied Robotics Research
Microsoft
Redmond, WA, USA

Kazuhiro Sasabuchi
Applied Robotics Research
Microsoft
Redmond, WA, USA

ISSN 2153-1056 ISSN 2153-1064 (electronic)
Synthesis Lectures on Computer Vision
ISBN 978-3-032-03444-1 ISBN 978-3-032-03445-8 (eBook)
https://doi.org/10.1007/978-3-032-03445-8

Prologue

Learning-from-Observation

Learning-from-observation (LfO) is a system designed to mimic human behavior through observation. Unlike methods such as Imitation Learning and Learning from Demonstration [1–4], which directly replicate human joint trajectories, LfO adopts a different, indirect approach. It segments the trajectories into meaningful parts, comprehends the underlying intent of each segment, and subsequently represents this understanding in *task models*, Minsky's frame-like representations [5]. These representations are then mapped to robotic skills, which are pre-trained in advance, for execution. This indirect imitation enables the system to discern which parts of the trajectory are crucial, allowing it to replicate only those essential segments or apply past experiences to new situations.

LfO also distinguishes itself from other methodologies by relying exclusively on visual information for imitation. While force sensations are commonly taught in robot learning—such as in teleoperation—LfO deliberately omits this aspect. According to Piaget's theory of cognitive development [6], infants undergo a developmental stage in which they mimic their parents' behaviors. In this learning process, infants and their parents share a visual space, enabling the infants to acquire actions through observation. However, due to differences in physical characteristics, the force sensations experienced by parents are not directly shared with their infants, requiring the infants to develop force sensations suited to their own bodies.

During this period, infants refine their actions through repeated performance of visually acquired behaviors, likely because force sensation feedback must be optimized for their own bodies. Similarly, in LfO, force sensations are used exclusively in local reinforcement learning to train skill agents for each robot hardware, effectively addressing hardware discrepancies. As a result, LfO operates through a system where, after deriving a general policy from visual information, skill agents—optimized for their specific hardware—execute the actions.

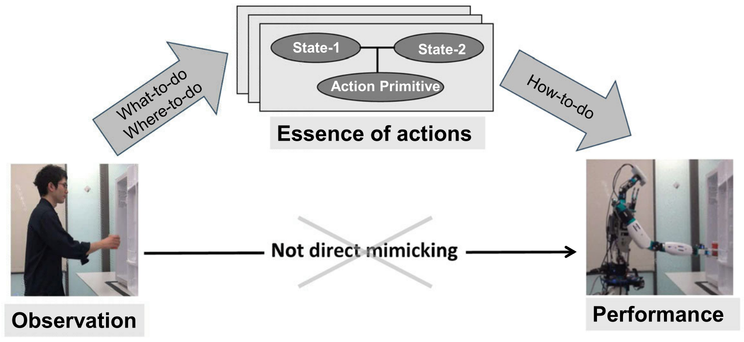

Fig. p.1 Learning-from-observation

The term “Learning-from-Observation” itself dates back to the 1990s with the Ikeuchi/Reddy system [7, 8]. LfO research commenced in the 1990s at CMU with the concept of endowing robots with this ability, analogous to how infants gradually acquire complex behaviors by imitating their mothers. Since then, systems with similar names, such as Learning by Demonstration and Programming by Demonstration, have been developed.

This book defines Learning-from-Observation as an indirect mimicking system where an intermediate framework, referred to as task model, is pre-constructed based on robotics theories. These task models play a central role in both observation and execution; human behavior is observed through the task models, and robot actions are generated based on them (See Fig. p.1).

This book is titled Learning-from-Observation 2.0. LfO has been a subject of research since the 1990s. Significant advancements in machine learning, including large language models, vision-langauge models, and reinforcement learning, have led to remarkable progress in both the encoder, which converts inputs into task models, and the decoder, which executes them. These advancements have resulted in performance that markedly differs from classical LfO systems. Nevertheless, the system retains the name LfO due to its structure, which utilizes an indirect imitation based on the traditional task models, characterized by the distinct separation of encoding and decoding processes.

Overview of Part I

This book is divided into two main parts. Part I details the functions of each component of LfO 2.0. Part II discusses the evolution of LfO research and the fundamental concepts underlying it. These parts are nearly independent, so you may start reading from either one. Alternatively, you may choose to begin with the prologue and epilogue, followed by the Part II and then Part I.

Chapter 1 of Part I explores the encoder in LfO 2.0, which has undergone significant improvements compared to its predecessor. In the original LfO, the encoder was developed in an ad hoc manner using classical computer vision techniques. However, with advancements in machine learning—particularly in vision-language models—its performance has dramatically improved in LfO 2.0.

The encoder consists of two key components: the task recognizer and the affordance analyzer. Section 1.1 focuses on the task recognizer, detailing how the encoder leverages vision-language models and other advanced technologies to decompose videos of human actions into sequences of task models—such as grasp task model or pick task model—representing what needs to be done. Each task model typically corresponds to a single verb.

A crucial aspect of this system is that the vision-language model is aware of the predefined set of task models (or verbs) available. As a result, it explains video content in terms of these accessible task models. This process of determining "what-to-do" is referred to as task recognition.

The affordance analyzer, described in Sect. 1.2 gathers the parameters essential for task execution and stores them into task models. Initially, it performs grounding to identify the tasks being performed in each segment of the video. Subsequently, it extracts task specific parameters from the corresponding video segments, referred to as skill parameters. For instance, in a grasping task, skill parameters might include the direction from which to grasp the object, while in a picking task, they might specify the direction in which to lift the object. These parameters indicate the position and direction of operations on the target object during task execution and can be considered as the object words corresponding to specific verbs.

The skill parameters, which convey the "where-to-do" information, vary depending on the task. For each task, they are predefined within the task model using corresponding slots, similar to Minsky's frame [5]. This predefined task model is called an *abstract task model*.

The task recognizer selects the appropriate abstract task model from a library of abstract task models. The affordance analyzer then completes the selected model as an "instantiated task model" by acquiring skill parameters. This process of selection and parameter assignment is called *instantiation*.

In programming terms, the task model functions as an operator, with skill parameters acting as the operands corresponding to each operator. The task recognizer selects the appropriate operator, while the affordance analyzer substitutes the operand values. From an object-oriented perspective on task operations, the encoder selects an object, and the affordance analyzer instantiates it. This instantiation triggers daemons within the object's slots, which retrieve the necessary parameters from the provided video segments, thereby completing the object during affordance analysis.

Chapter 2 discusses the decoder, which plays a crucial role in the robot's task execution. The decoder activates a sequence of skill agents based on instantiated task models and their corresponding skill parameters, ensuring consistency across tasks. Each task model has a one-to-one correspondence with a skill agent, which is responsible for executing the required actions on the robot.

Skill agents are pre-trained through reinforcement learning to develop action-specific policies. Consequently, the program units containing these learned policies are referred to as skill agents. They utilize skill parameters, along with visual and force feedback from the environment, to ensure precise execution tailored to each robot's hardware. Once activated by the decoder, skill agents initiate actions using values from human demonstrations as initial parameters and dynamically adjust execution based on feedback—such as force feedback—to achieve the desired behaviors. All available skill agents are stored in a skill library, which is structured to comprehensively cover the target domain without overlap.

Human actions can be categorized into two types: grasping actions, which occur before making contact with an object, and manipulation actions, which take place after contact. As a result, two distinct skill libraries are established—one for grasping actions (Chap. 3) and another for manipulation actions (Chap. 4). These libraries are designed in advance to prevent skill overlap while ensuring comprehensive task coverage through mathematically backed selection processes. Specifically, the grasp skill library is constructed based on Yoshikawa's closure theory (Chap. 3), whereas the manipulation skill library is formulated using the Kuhn-Tucker theory (Chap. 4).

Overview of Part II

While the Part I discusses the current state of LfO 2.0, Part II conducts a retrospective review of the research trends from the early days.

Chapter 5 in Part II, entitled "Big Bang of LfO," explores the first Learning-from-Observation (LfO) system created by Ikeuchi and Reddy at CMU in 1989 [7]. See also the acompaning YouTube video, *Learning-from-observation: Big bang*.[1] This pioneering system involved a robot replicating human actions to construct identical structures using two blocks. Essentially, it served as an online iteration of the "copy demon" developed by Prof. Mavin Minsky and his students, including Profs. Berthod. K. P. Horn, Patrick H. Winston and Thomas O. Binford, in the 1980s at MIT's AI lab. This system was only capable of handling two blocks, but it introduced the concept of associating state transitions with corresponding manipulation actions—an approach that later became the foundation of LfO's task model design. This chapter further elucidates the parallel structure between this concept and Marr's object recognition framework.

Chapter 6 delves into the concept of LfO in the realm of polyhedral operations. The system introduced in the previous chapter presented the concept of a task model, associated with state transitions, but only dealt with limited transition relations between two cubes.

[1] https://youtu.be/UkvDUmRHg4c.

Ikeuchi and Suehiro [9, 10] defined the state of a manipulated object as the range of possible movements due to contact with the environment and defined manipulation actions (i.e., tasks) as causing transitions in this range of movement. This range of movement can be represented as the solution space of a set of simultaneous linear inequalities defined by the surface orientations of the contact points. By examining the rank of the simultaneous linear inequalities using Kuhn-Tucker theory, this solution space can be classified into ten patterns, which consequently defines ten patterns of contact states. Since transitions in contact states occur between these ten states, the upper bound is 100. By sequentially investigating which of these 100 transitions actually occur, they demonstrated that, for polyhedral assembly, only 13 transitions—thus 13 tasks—need to be considered. They prepared 13 abstract task models and constructed LfO in the polyhedral world accordingly.

Chapter 7 discusses LfO in the context of manipulating flexible objects, with a focus on knot tying. Beyond rigid bodies, robots often handle a variety of non-rigid objects. Examples of flexible object manipulation include knot tying and folding clothes.

As an attempt to apply a similar framework in the world of flexible objects, Takamatsu et al. adopted a data format known as P-data, developed in the field of genetic analysis within string theory, to represent the state of ropes [11]. It is known that P-data and rope projections have a one-to-one correspondence, enabling the derivation of unique P-data from projections and the reconstruction of the original projections from P-data. Additionally, transitions in P-data correspond to possible rope manipulations known as Reidemeister moves. By utilizing P-data transitions as task models for rope manipulation, akin to face contact state transitions, they demonstrated the applicability of the LfO framework to flexible objects.

Chapter 8 explores LfO in the context of non-contact tasks, with a particular focus on dance. In polyhedra and ropes, the approach relied on inferring actions based on contact transitions between objects, emphasizing the imitation of outcomes. However, in certain cases, accurately mimicking the movement itself is more important. Dance and martial arts—such as karate and judo—are prime examples.

In these scenarios, exact replication of every movement is impossible due to physical differences between the teacher and the student. Instead, only specific parts of the movements—or the features represented by those parts—are imitated. In other words, within a sequence of continuous movements, only the essential actions that align with the overall purpose, such performing dance or martial arts, are expressed.

In dance, only postures that viewers perceive as identical are imitated. The dance community has developed and used a notation system called Labanotation to describe sequences of these postures [12]. Each Labanotation represents a state, while transitions between them are defined as tasks. Ikeuchi described the upper body using Labanotation [13], while Nakaoka et al. proposed a method for describing the lower body [14, 15]. By integrating these approaches, they developed an LfO system for learning the Aizu Bandaisan dance.

About Epilogue

In the epilogue, we conclude this book with a discussion on the design philosophy underlying LfO and its distinctions from other methods, approached from an advanced perspective. First, it is argued that LfO has been designed along Piaget's theory of cognitive development, thereby utilizing only visual information and restricting the use of force sensing to local reinforcement learning. Subsequently, the discussion explores why the method of direct imitation, as represented by imitation learning, was not adopted, and what advantages this approach yielded. Finally, the epilogue examines the relevance of Reddy's 90% AI, emphasizing that the design of embodied AI should maintain the fundamental stance of supporting humans, augmenting—rather than replacing—human cognitive and physical abilities.

Competing Interests The work presented herein was primarily conducted by the authors in collaboration with their students and colleagues affiliated with Carnegie Mellon University, the University of Tokyo, and Microsoft. The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the institutions. The latest software demonstrated in the videos has been released by Microsoft as open-source under the MIT license.

Contents

Part I

Learning-from-Observation 2.0

1 Encoder and Its Operation

This section elaborates on a vision-language model-powered encoder designed to translate human demonstrations into task models. We refer to this computation as an encoder because it converts human actions into a sequence of symbolic representations—i.e., task models.

Within the framework of Learning-from-Observation (LfO), human demonstrations, when observed visually, are interpreted with the granularity of task models, defined in a top-down manner. These task models outline high-level actions and are decoupled from the sequences of motor commands specific to individual hardware. Task models solely encapsulate the "what-to-do" along with the corresponding "where-to-do" parameters, forming representations akin to Minsky's frame [5]. The robot actions necessary to execute these task models—i.e., the "how-to-do"—are provided by a subsequent decoder system equipped with skill libraries.

In this context, the "what-to-do" describes abstract actions that correspond to single verbs.[1] This structure is well suited for general vision-language models (VLM), which function as encoders that describe input scenes using a standardized set of verbs. The VLM-empowered encoder consists of two sequentially connected modules: a task recognizer and an affordance analyzer [16].

The first module, the task recognizer, processes demonstration videos and generates a sequence of task models representing the actions to be executed by a robot. Specifically, this process begins with VLM (VLM) (i.e., GPT-4V) analyzing the video to convert environmental details and actions into text, followed by large language model (LLM) (i.e., GPT-4) generating the corresponding sequence of task models.

[1] More precise definitions are given in Chaps. 3 and 4.

K. Ikeuchi et al., *Learning-from-Observation 2.0*, Synthesis Lectures on Computer Vision,
https://doi.org/10.1007/978-3-032-03445-8_1

The second module, the affordance analyzer, determines the "where-to-do" aspect by identifying the timing, locations, and methods of tasks observed in human demonstration videos. It extracts the spatial information necessary for effective robot execution.

Task models can be categorized into two types: abstract task models and instantiated task models, both of which share a structure similar to Minsky's frames. An abstract task model serves as a structural framework with its slots remaining unfilled. First, the encoder identifies an appropriate abstract task model from a set of possible models that correspond to the input action. Subsequently, the affordance analyzer derives the necessary parameters based on the attributes of the slots within the abstract task models, completing its formulation. Once the slots are populated with specific values, the model is referred to as an instantiated task model.

The affordance analyzer reanalyzes specific sections of the video, focusing on the time frames identified by the task model. Through spatiotemporal grounding, it collects affordance data—such as grasp type, way points, and body posture—and combines this data with each task as skill parameters, completing the instantiated task model. This approach can be understood through the lens of sentence analysis, where the affordance analyzer recognizes objective words corresponding to each verb.

It is assumed that input videos for this system are designed for a teaching framework in which the robot replicates human actions. Consequently, there is no self-occlusion or intentional concealment within the demonstrations, ensuring clear and effective task recognition.

1.1 Task Recognizer

The task recognizer is further decomposed into three sub-modules: (1) video analyzer, (2) scene analyzer, and (3) task-cohesion generator. See Fig. 1.1.

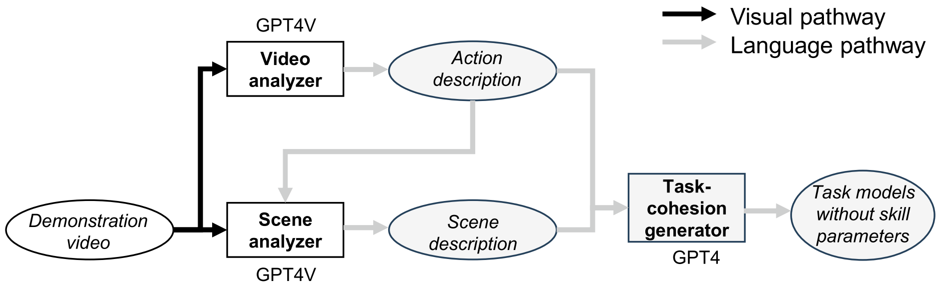

Fig. 1.1 Task Recognizer [16]. The input video is captioned using actions that the robot can perform, as identified by the video analyzer. Subsequently, related objects are extracted by the scene analyzer. Finally, the task cohesion generator creates a sequence of task models based on the output of the scene analyzer. The gray components and lines are associated with language/text processing, while the white components and black lines are related to image processing

1.1.1 Video Analyzer

When a demo video is input, a VLM recognizes the actions performed by humans in the video and outputs them as text. An example of a VLM is OpenAI's GPT-4V or GPT-4o. To ensure that the subsequent Task Cohesion Generator can effectively process human instruction sentences, the Video Analyzer transcribes the video in a style commonly used for human-to-human communication, essentially functioning as video captioning using a predefined pool of verbs.

Given the token limitations and latency of the VLM, not all frames are analyzed. Instead, video frames are extracted at regular intervals and input into the VLM. By outputting descriptions, human verification and correction become easier.

As an example, we will use a demonstration of a human moving a can of Spam on a table to explain the process. This setup is divided into two parts: an observation station and an execution station,[2] analogous to how a processing system is divided into an encoder and a decoder.

At the observation station, human actions are observed and input into the encoder. Specifically, as shown in Fig. 1.2, a demonstration is conducted in front of the Azure Kinect. AR markers are installed at both stations to establish the rough relationship between their coordinate systems. These markers define a general positional relationship and are used to re-utilize the vector coordinate system of affordance parameters during execution.

It should be noted that during execution, the object's position does not need to be strictly identical at both stations for the execution station to recognize the object's position anew. It is also assumed that the arrangement of obstacles around the target object is almost the same.

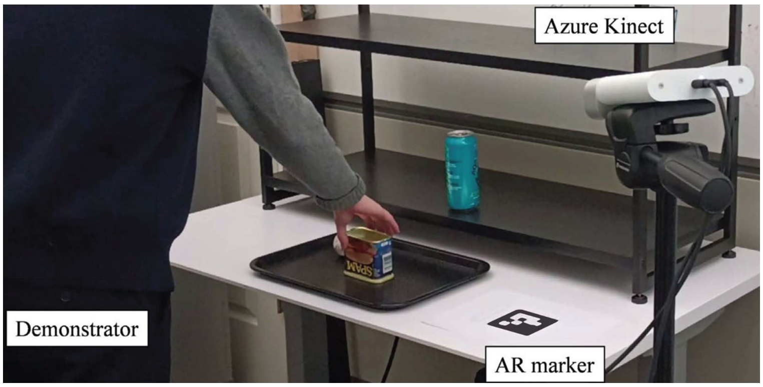

Fig. 1.2 Observation station

[2] Section 2.3 provides a detailed explanation of the execution station, but in summary, the site where the robot executes tasks based on the generated task model is referred to as the execution station.

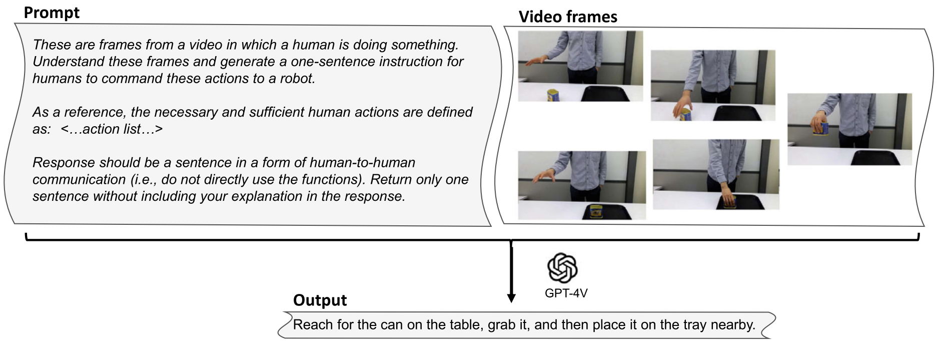
Prompt
These are frames from a video in which a human is doing something. Understand these frames and generate a one-sentence instruction for humans to command these actions to a robot.
As a reference, the necessary and sufficient human actions are defined as: <...action list...>
Response should be a sentence in a form of human-to-human communication (i.e., do not directly use the functions). Return only one sentence without including your explanation in the response.
Video frames
GPT-4V
Output
Reach for the can on the table, grab it, and then place it on the tray nearby.

Fig. 1.3 Video analyzer. The top plane shows the prompt for the GPT-4V, while the bottom plane presents examples of its output. Five frames, including the first and the last frames of the video, are extracted at regular intervals and fed into GPT-4V

As illustrated in Fig. 1.3, the video sequence and prompt are input into the VLM. In front of this observation system, a can of Spam is moved on the tray. The VLM then generates the sentence, “The person reaches for the can on the table, grab it, and then place it on the tray nearby.” The role of the video analyzer is to generate this sentence. Notably, unlike general captioning, the description of actions uses only verbs limited by the prompt, rather than common verbs.

1.1.2 Scene Analyzer

The scene analyzer converts the work environment into text information based on the text output by the video analyzer and the first frame (or an appropriate frame) of the video data. Specifically, the output includes a list of object names being operated on and the spatial relationships between these objects and related objects.

The object names related to the tasks are provided in an open vocabulary format based on GPT-4V’s understanding. These names are then used by the affordance analyzer in the next stage to match with video segments.

Figure 1.4 shows an example of the scene analyzer in action. As illustrated, the scene analyzer identifies objects related to the operation such as “can,” “table,” and “tray.” In this case, since a person moves a Spam can to the tray, the tray is included in the output.

1.1.3 Task-Cohesion Generator

The task-cohesion generator uses GPT-4 to generate the necessary sequence of task models from the given action description, environment description, and available abstract task mod-