Using The Concept Hierarchy for Household Action Recognition*

Andrei Costinescu, Luis Figueredo and Darius Burschka[1]

***Abstract*—We propose a method to systematically represent both the static and the dynamic components of environments, i.e. objects and agents, as well as the changes that are happening in the environment, i.e. the actions and skills performed by agents. Our approach, the Concept Hierarchy, provides the necessary information for autonomous systems to represent environment states, perform action modeling and recognition, and plan the execution of tasks. Additionally, the hierarchical structure supports generalization and knowledge transfer to environments. We rigorously define tasks, actions, skills, and affordances that enable human-understandable action and skill recognition.**

***Index Terms*—knowledge modeling, and perceptual reasoning.**

I. Introduction

Making sense of the world in which we, humans, live is a difficult problem. It is even more difficult for a robotic system. Acting intelligently and interacting with objects for purposeful changes in the environment requires both knowledge of objects, agents, actions, skills, and tasks as well as algorithms that leverage this knowledge and take into account the task, the abilities of the performing agent(s), and the environment in which the task is to be performed.

It is the goal of this paper to present a comprehensive framework for knowledge representation, including objects, agents, actions, skills, and tasks, that aids in the understanding of and participation in the environment dynamics, i.e. the environment changes and their causes. Thus, the framework can be used in, but is not limited to, household service robotics as in Figure 1. Applications of the framework include task planning, environment modeling, and action recognition.

Intelligent agents reason about perceived data in an environment and decide to act in the environment to fulfill a goal [1]. Structuring and representing knowledge for artificial agents is an important domain of artificial intelligence. One such technique is a **Semantic network**, also known as an ontology, which defines knowledge as concepts and semantic relations between them. OWL, the web ontology language, is a standard for writing ontologies [2]. In OWL, concepts, instances, properties, and relations between concepts can be defined. Properties have associated value types and one can define custom value types, such as vectors or matrices. However, inference with custom data types is not supported, and defining functions to modify the values of custom types is not easy. Composing functions to create new ones is not supported.

Knowrob [3] extends the ontology in [4] and aims to represent knowledge for executing robotic skills. Based on OWL, it is thus quite cumbersome to define functions in the knowledge representation itself that check if an action or a

*This work was supported by the Lighthouse Initiative Geriatronics by StMWi Bayern (Project X, grant no. 5140951).

[1]All authors are with the School Of Computation, Information, and Technology at the Technical University of Munich. {andrei.costinescu, luis.figueredo, burschka}@tum.de

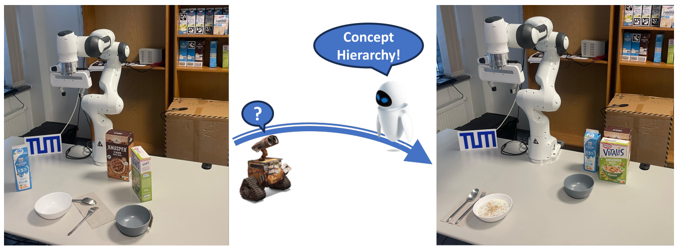


Fig. 1: "How to transform the left environment into the right one?" The knowledge in the Concept Hierarchy enables household robots to represent environments and to create a plan to execute tasks.

skill is active in the environment. Furthermore, there is no clear distinction between a task, an action, and a skill. Also, knowledge of verifying if skills are active is not represented in Knowrob; only knowledge about task execution.

The authors of [5] define a robot task planning ontology in which tasks are represented as linear action sequences. Their action definition as "atomic actions" is circular and resembles the definition of a motion primitive, implying that actions have different meanings for different robots. We clarify this in II-C and II-D by providing a concrete definition for actions.

The authors of [6] represent knowledge as a conceptual hierarchy that can specify default values and exceptions, thus being a non-monotonic knowledge base. We also employ the Principle of Specificity described in [7] to model default values and exceptions for concept properties, e.g. for skill properties.

Probabilistic learning methods have become popular in various applications, including action recognition [8], [9]. Their advantage is a fast generation of possible action hypotheses and a human-like description or segmentation of the recognized actions in a demonstration. Such approaches lack, however, an understanding of **why** motions "look like" actions.

A model-based approach enables richer uses of knowledge, such as recognizing failures and pinpointing the reason for an unsuccessful skill execution thanks to a model of the physical world. Furthermore, a model-based approach can help identify the missing steps or needed circumstances to successfully execute a skill. Having a model of an action enables verification of whether it is truly happening in the scene and not just if it "looks like" the action is executed. Finally, compared to neural networks, a model-based approach **can** model the states of the environment and make long-term temporal decisions based on the properties of objects present in the environment, so we have decided not to use a learning approach for action recognition.

A. Contributions

Our new knowledge representation allows us to represent both the "things" in an environment, i.e., objects and agents, and the changes happening in the environment. We provide structured definitions of the represented knowledge and fill the

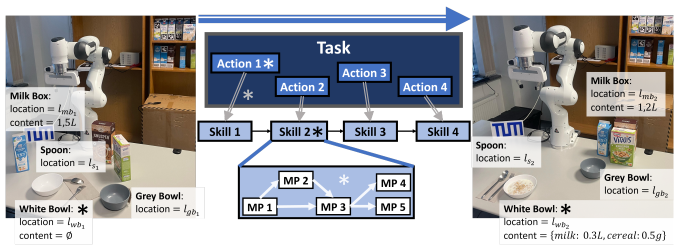


Fig. 2: The task of transforming the left environment to the desired right one is divided into actions that are correlated to skills, which are composed of Motion Primitives (MP). * marks knowledge from the **CH**: action and skill definitions, skill-to-action association, sequencing of motion primitives for skill execution, and entity properties.

Concept Hierarchy (**CH**) with concepts and instances that can be used to execute skills on a robotic agent and to recognize and understand the changes in the environment. We rigorously define the difference between tasks, actions, and skills (terms that usually do not have a clear meaning or differentiation in robotics literature) and show how affordances are represented in the **CH**. Our implementation is written in C++ and supports dynamic type changes of instances, i.e., instances adding or removing concepts at runtime. It also enables data updating and programmatic restructuring of the existing knowledge.

B. *Structure*

Section II presents our modeling of concepts, instances, objects, agents, actions, skills, tasks, and contexts in the **CH**. The **CH**'s use for action and skill recognition is described in Section III, followed by the conclusion and future work.

II. The Content of the Concept Hierarchy

The Concept Hierarchy (**CH**) is a knowledge modeling framework to represent the information needed for a particular application or within an application domain. Figure 2 shows an application domain where the **CH**'s data is useful.

To turn the left environment into the right one, one must first describe the state of both environments. Then, one determines the differences between the two environments, which lastly results in a sequence of changes for an agent to execute.

For describing the environment state, it is not enough to just segment the environment's geometry into clusters. The clusters must have semantic information attached to them and properties that differentiate the clusters from others of similar type or shape. We thus introduce *Concepts* that define the properties of different cluster types, i.e. *Container*, *MilkBox*, *Bowl*, *CerealBox*, *Spoon*, *Human*, *Robot*, etc. The actual properties values are not set by *Concepts*, but by *Instances*, the knowledge containers. The environment state is the collection of all properties of entities inside it, i.e. *Objects* and *Agents*.

The differences between the entity properties in the start and end environment are formalized and represented as *Actions* in the **CH**. One *Action* represents one change in an entity's property. In Figure 2, the location property of the *Milk Box* is different. This is represented by the *ChangeLocation Action*. For performing this change (physically) in the environment, we represent *Skills* in the hierarchy, which are agent-dependent. For executing the *ChangeLocation* in the environment, a *Human* can choose from a large number of *Skills*, such as *Transporting*, *Throwing*, *Rolling*, *Pushing*, etc. *Actions* and

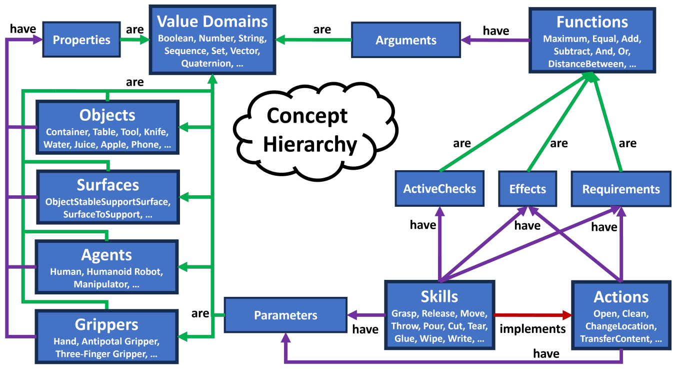

have
Properties
are
Value Domains
Boolean, Number, String, Sequence, Set, Vector, Quaternion, ...
are
Arguments
have
Functions
Maximum, Equal, Add, Subtract, And, Or, DistanceBetween, ...
are
Concept Hierarchy
Objects
Container, Table, Tool, Knife, Water, Juice, Apple, Phone, ...
Surfaces
ObjectStableSupportSurface, SurfaceToSupport, ...
Agents
Human, Humanoid Robot, Manipulator, ...
Grippers
Hand, Antipotal Gripper, Three-Finger Gripper, ...
are
are
are
ActiveChecks
Effects
Requirements
have
have
have
are
Parameters
have
Skills
Grasp, Release, Move, Throw, Pour, Cut, Tear, Glue, Wipe, Write, ...
implements
Actions
Open, Clean, ChangeLocation, TransferContent, ...
have

Fig. 3: The **CH** contains *Objects*, *Surfaces*, *Agents*, and *Grippers*. Their specific properties are modeled as *ValueDomains*. *Skills* and *Actions* contain *ValueDomain* parameters, including *Objects* and *Agents*. *Skills* perform *Actions* in an environment. They have requirements from and effects on their parameters and can check whether a skill is active. These are modeled as *Functions* with *ValueDomain* arguments.

Skills are also *Concepts* and have parameters that describe the change and the way of performing it in the environment.

After determining the necessary changes, a *Task* planner sequences, prioritizes and optimizes their execution in the environment. This final *Task* execution plan is then distributed to the agent(s) that will transform the environment into its desired state. The next sections present how the stored knowledge is grouped and structured inside the Concept Hierarchy. Figure 3 presents the interaction of the knowledge in the **CH**.

A. ***Concepts**, **Instances** and **ValueDomains**: Knowledge Definitions, Containers, and Data Types*

In the example of Figure 2, the environment consists of many **Instances**, some of them being the spoon, the white and grey bowls, and the milk box. Respectively, the entities are instances of the *KitchenUtensil*, *Bowl*, and *MilkContainer* concepts; the latter two are subconcepts of *Container*.

Some differences between the left and right of Figure 2 are that the milk box has less content and a different location, the grey bowl has a different location, and the white bowl has now milk and cereal as contents. The terms content and location are instance properties. They are defined in the *Container* and *PhysicalEntity* concepts respectively. A property also defines the range of values that it can have. This value range is modeled as *ValueDomains*, knowledge data types.

The definition of a *Concept* in the **CH** includes which properties pertain to the concept and their associated *ValueDomain*. E.g., the location property of a *PhysicalEntity* has a *Location ValueDomain*. A *Location* is a pose displacement relative to a coordinate frame. And the *ValueDomain* of the content property of a *Container* is a *List* of *PhysicalEntities*.

Concepts are organized hierarchically and inherit the property definitions of their parent *Concepts*. The *Concept* itself does not specify a value for its properties; this is done by *Instances*, the containers of knowledge. Unspecified property values are set to the *UNKNOWN* value, making the world model follow an open-world assumption. *ValueDomains* are not bound to real data types used in current programming languages, to not tie the **CH** to a specific programming language, and to let the domain expert implement the optimal data structures for the particular application of need.

B. ***Functions***: *Mathematical Modifiers of ValueDomains*

The changes in the property values of concept properties are carried out by *Functions*. *Functions* have arguments *ValueDomains* and represent operations that can be executed on them. E.g., *Addition*, *Multiplication* of two *Numbers*, *Incrementing* one *Number*, *Inserting* one item to a *List*, and so on. The meaning of the *Function*, i.e., what it actually does with its inputs, is, as with *ValueDomains*, not necessarily specified in its definition of the **CH**. The application designer's job is to implement the meaning of the *Function* in a desired programming language using the data types selected as *ValueDomains*. The *Function*'s definition inside the Concept Hierarchy only needs to specify the function's interface and its name; similar to what a header file in C++ defines.

There are *Concept* properties that form a dependency cycle: *Agents* have *Grippers*, and a *Gripper* belongs to an *Agent*. Sometimes, changing an instance property must have an effect on a different property in a related instance, e.g., removing a gripper from an agent must update the belongingAgent of the removed gripper. To model this, *Concepts* define *hooks* on properties, which are represented as *Functions*.

C. ***Actions***: *Modifiers of Knowledge Containers*

Actions are used to formally model the changes in Figure 2. We define an *Action* as one change in the properties of an instance. E.g., the *ChangeLocation Action* changed the location property of the milk box, and *TransferContent* took 0.3L of milk from the milk box into the white bowl.

Actions do not describe how the change was achieved in the environment; they model **what** the change was. Actions have preconditions from and effects on their entity parameters (i.e. the instances on which the action is to be performed), which are modeled as *Functions*.

D. ***Skills***: *Action Implementation in Environments*

The *Concept* describing **how** the change represented by an *Action* was performed is a *Skill*. E.g., *Pushing*, *Pulling*, or *Carrying* are *Skills* that could have been used to *ChangeLocation* of the milk box and *Pouring* or *Scooping* could have been used to *TransferContents*. *Skills* are agent-, instance-, and environment-specific. They depend on the capabilities of the executing agent, the instance's properties, and the environment's configuration (e.g. obstacles) for successful execution. In addition to the preconditions and effects, *Skills* define two more *Functions*: the check for the skill being active and for it being successfully executed in the environment.

Being particular to certain agents, instances, and/or environments, a *Skill*'s property *ValueDomain* definition can include concepts that instances must not be sub-concepts of. *Flying*, a *Skill* associated with the *ChangeLocation Action*, has *Birds* as the *ValueDomain* for its entity property and also the restriction of the entity not being a *Penguin*, known not to be able to fly.

The *Skill* concept defines the *manipulations* property: a triple of agent-gripper-object entities (an agent uses a gripper to manipulate an object). *Skill* subconcepts define a default value for the *manipulations* property, that all *Skill* instances will have unless overwritten by the instance. E.g., *Pouring*, the skill describing the process of tilting the *Container* $from$ located in the *Agent* a's *Gripper* g over the *Container* $into$, defines the manipulation's default value to be $\{(a, g, from)\}$. The subconcept of *Pouring*, *PouringWith2Grippers*, which has the *into Container* in the *Agent* a's second *Gripper* g_2, overwrites *Pouring*'s default value with $\{(a, g, from), (a, g_2, into)\}$. This feature of the **CH** makes it a non-monotonic knowledge base.

Skills also have an actionAssociation property, that creates the association between a *Skill* and the, possibly multiple, *Actions* that the *Skill* implements in the environment.

E. ***Affordances***

Determining or estimating entity affordances is still an open problem in robotics. Affordances indicate which actions and skills can be performed on the entity. I.e., into which action or skill property can the entity be substituted. We represent the action-affordances of an entity e as the set of pairs $Aff(e) = \{(a, p) \mid a \in Actions, p \in E(a),\ e$ is a subconcept of the property p of the action $a\}$, where $E(a)$ is the tuple of a's entity-properties. Skill-affordances are defined similarly.

Affordances are thus related to the definition of action and skill properties. The *Concepts* that are the *ValueDomains* of action and skill properties thus know as which properties their instances can be used. Therefore, affordances do not need to be stored explicitly in the **CH**; they are the entity-*ValueDomains* of the action and skill properties.

F. ***Motion Primitives, Abilities***: *Building Blocks of Skills*

Abilities are the building blocks of skills. They are the generalization of motion primitives; a motion is not always needed as a part of skill execution. For *Walking*, the agent must perceive the environment to determine a collision-free walking path. Perception does not always require a motion from the agent. Another ability is waiting until something has happened, which does not involve any motion. Yet, it is an important part of a skill's execution, e.g. during collaboration with multiple agents. *Skill* execution is thus converting its parameter values into correct parameters for the executing agent's abilities.

G. ***Tasks***: *Multiple Changes in the Environment*

In our work, a task is a desired state of the environment. A state of the environment specifies instance properties. This desired state, when compared to the current state of the environment, results in differences in entity properties: a set of instance property changes, i.e. a set of actions. The task can also define constraints on the ordering of actions or constraints on their execution that must be considered by a planner when parsing the set of actions into a sequence of skills.

In Figure 2, the task is to transform the environment into the state on the right. By comparing the two environments, the **task** planner determines the differences in the values of instance properties in the environment and determines the *Actions* that have been done. It is then a smart agent's job to associate the *Actions* with *Skills* that implement the *Action*, that fit the abilities of the agent, and that are executable in the environment configuration.

The following section presents an example of using the **CH**'s knowledge in a household setting.

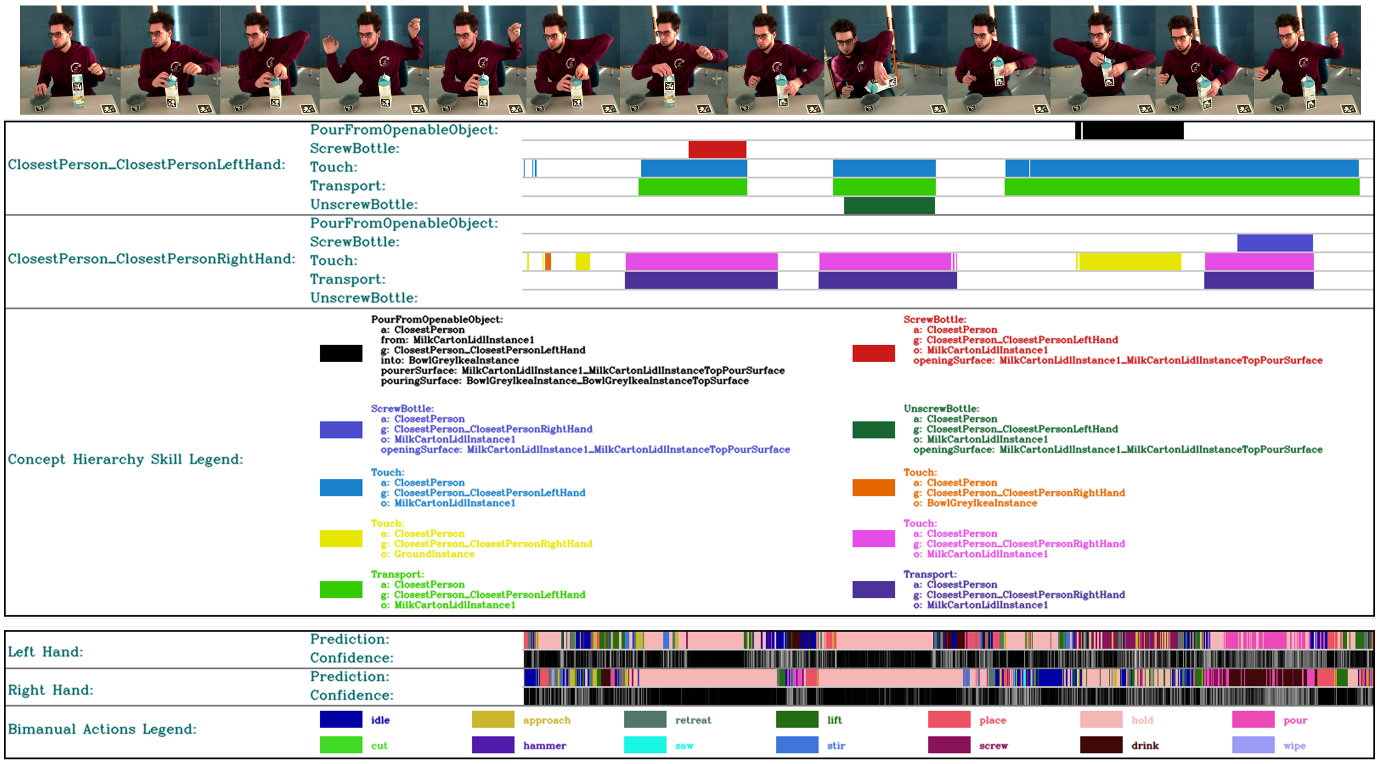
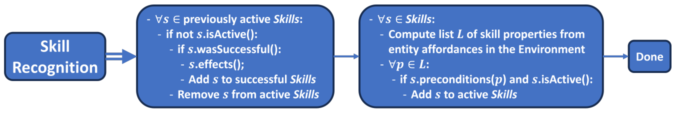


Fig. 4: Skill recognition result of a bimanual task of pouring milk into a bowl. The demonstration is composed of closing and then opening a milk box with the left hand, pouring milk into the bowl, and closing the milk box with the right hand. The upper figure presents the *Skill* instances our method recognizes for each hand and *Skill* type. The colors help distinguish *Skills* of the same type with different parameters. The lower figure shows the results of [9] that was trained on the Bimanual Actions dataset [10] (darker confidence = higher softmax output).



```
- ∀s ∈ previously active Skills:
    - if not s.isActive():
        - if s.wasSuccessful():
            - s.effects();
            - Add s to successful Skills
        - Remove s from active Skills
```

```
- ∀s ∈ Skills:
    - Compute list L of skill properties from entity affordances in the Environment
    - ∀p ∈ L:
        - if s.preconditions(p) and s.isActive():
            - Add s to active Skills
```

Fig. 5: The procedure for *Skill* recognition.

III. Action and Skill Recognition

We present the **CH**'s use in recognizing actions and skills in a household environment. We have used OpenPose [11] and AprilTags [12] to get human hand 3D positions and object 3D poses from a Realsense [13] camera. Having a system based on visual input, the functions checking if a *Skill* is active are visual, geometrical relations between object and agent instances. With each new camera frame, the objects and agents are recognized and localized inside the environment. Then, the skill recognition process starts and is presented in Figure 5. Using the actionAssociations property, the successfully executed skills determine the actions in the environment.

Figure 4 compares the observed skills from our method with [9] on a milk pouring example where the milk box is closed and then opened with the left hand and closed again with the right hand. The *Transport* skill defines its *o* parameter as a *MovableObject*, which is why the *MilkCartonLidlInstance* was transported and the *GroundInstance* not, even if touched with the right hand.

Compared to a learning method that outputs probabilities of recognized skills, our method certainly determines successfully executed skills. It can also determine multiple skills active at the same time as well as with which objects the agent interacted. The probabilistic method has misdetections possibly explained by the agent executing similar motions with his hand that "look like" other actions, even if no object was touched. Even if the agent moves his hand in a lifting manner, which the learning-based method recognizes, it fails to understand that no object is lifted or even held in the hand. This is the advantage of a model-based method: the verifiability and explainability of a decision; lifting did not occur because the hand did not touch an object in that frame.

IV. Conclusion

This paper describes the building blocks of our knowledge modeling framework for household applications, the Concept Hierarchy. This definition of concepts and instances efficiently represents household objects in manipulation tasks. It supports dynamic changes of concept properties due to performed actions and enables inheritance and overwriting of default property values. Our parametrizable skill and action definition allows both a general and restrictive specification of instances, e.g. the *Flying Skill*'s agents are *Birds* except *Penguins*.

Future work includes using the defined knowledge for learning and executing *Tasks* and creating a planner to leverage the intrinsic liberties of *Task* descriptions. Furthermore, we plan to infer unknown instance properties from the open-world assumption by observing the effects and preconditions of performed actions on the instance.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence, Global Edition A Modern Approach*. Pearson Deutschland, 2021. [Online]. Available: https://elibrary.pearson.de/book/99.150005/9781292401171

[2] World Wide Web Consortium, "Owl," accessed 11. Mar. 2024. [Online]. Available: https://www.w3.org/OWL/

[3] M. Tenorth and M. Beetz, "Knowrob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013. [Online]. Available: https://doi.org/10.1177/0278364913481635

[4] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira, "An introduction to the syntax and content of cyc," in *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:9826831

[5] X. Sun, Y. Zhang, and J. Chen, "Rtpo: A domain knowledge base for robot task planning," *Electronics*, vol. 8, no. 10, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/10/1105

[6] L. A. Pineda, A. Rodríguez, G. F. Pineda, C. Rascón, and I. V. M. Ruiz, "A light non-monotonic knowledge-base for service robots," *Intelligent Service Robotics*, vol. 10, pp. 159–171, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:30298067

[7] C. Strasser and G. A. Antonelli, "Non-monotonic Logic," in *The Stanford Encyclopedia of Philosophy*, summer 2019 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2019.

[8] Y. Kong and Y. Fu, "Human action recognition and prediction: A survey," *International Journal of Computer Vision*, vol. 130, no. 5, pp. 1366–1401, May 2022. [Online]. Available: https://doi.org/10.1007/s11263-022-01594-9

[9] H. Xing and D. Burschka, "Understanding spatio-temporal relations in human-object interaction using pyramid graph convolutional network," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 5195–5201.

[10] C. R. G. Dreher, M. Wächter, and T. Asfour, "Learning object-action relations from bimanual human demonstration using graph networks," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 1, pp. 187–194, 2020.

[11] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[12] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2011, pp. 3400–3407.

[13] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," *CoRR*, vol. abs/1705.05548, 2017. [Online]. Available: http://arxiv.org/abs/1705.05548