2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

An Open and Flexible Robot Perception Framework for Mobile Manipulation Tasks

Patrick Mania[1], Simon Stelter[1], Gayane Kazhoyan[1] and Michael Beetz[1]

***Abstract*— Over the last years, powerful methods for solving specific perception problems such as object detection, pose estimation or scene understanding have been developed. While performing mobile manipulation actions, a robot's perception framework needs to execute a series of these methods in a specific sequence each time it receives a new perception task. Generating proficient combinations of vision methods to solve individual perception tasks remains a challenge, as the combination depends on the requirements of the task and the capabilities of the robot's hardware.**

In this paper, we propose RoboKudo, an open-source knowledge-enabled perception framework that leverages the strengths of the Unstructured Information Management (UIM) principle and the flexibility of Behavior Trees to model task-specific perception processes. The framework can combine state-of-the-art computer vision methods to satisfy the requirements of each perception task and scales to different robot platforms. The generality and effectiveness of the framework are evaluated in real world experiments where it solves various perception tasks in the context of mobile manipulation actions in a household domain. Code and additional material are available at `https://robokudo.ai.uni-bremen.de/rkop`.

I. Introduction

Introducing service robots into people's homes presently still requires much research, as the robots are not yet competent enough to carry out mobile manipulation tasks robustly in household domains. One of the main challenges is solving the various perception tasks that are encountered in such domains, including one-shot perception and continuous tracking, recognizing various types of objects and humans, estimating object poses, etc. Consider, for example, the task of preparing a bowl of breakfast cereal. In order to fetch the milk from the fridge, the robot has to first detect a suitable handle on the fridge door to open it. Afterwards, it can use an object detection model to find the milk. If different variants of milk are present, e.g., lactose free and whole milk, the robot needs to read the text on the milk box or recognize the logo in order to fetch the correct type. To pour milk into the cereal bowl, the robot needs to first detect the closed lid and estimate its pose to be able to unscrew it.

Even though computer vision has developed robust and impressive methods for solving individual perception problems, an overarching method that can handle the variety of vision tasks in the household domain does not exist to date. A perception system must, therefore, allow to effectively combine available methods, while satisfying context-specific requirements in terms of the (1) vision methods appropriate for the current perception task and (2) the order in which the methods have to be executed. While a typical tabletop segmentation approach on pointclouds can detect standard objects with a sequential process — first plane detection, then tabletop point extraction, then clustering — a different method is required when working with transparent objects that are almost invisible to depth-based approaches. In addition to one-shot perception, more complex tasks may require a continuous and reactive process execution. For example, the robot may need to switch from an object detection process to a tracking process when it becomes important to keep a specific object in focus. It may also be needed to add verification steps at the end of the perception process, such as ensuring that the perceived scene is consistent with respect to the rules of physics. The need for perception frameworks that allow for flexible combinations of different vision methods to handle this large variation of perception tasks has been recognized in recent literature [10].

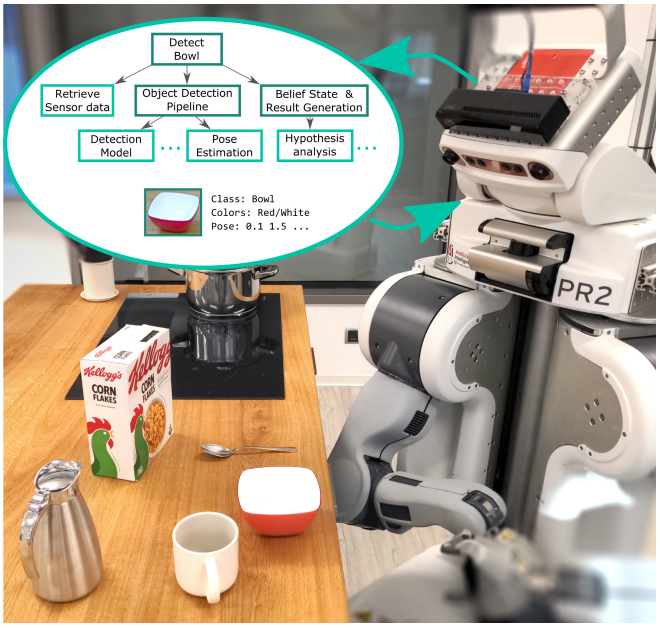

Fig. 1. The proposed perception framework allows robots to solve perception tasks by creating a combination of state-of-the-art computer vision methods based on the requirements of the task and the capabilities of the robot sensors. The sensor data is handled with the UIM approach and the perception processes are represented with Behavior Trees.

The research reported in this paper has been (partially) supported by the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 Project-ID 329551904 "EASE - Everyday Activity Science and Engineering", University of Bremen (http://ease-crc.org/). The research was conducted in subproject R02. This work was also supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017089 as part of the TraceBot project. We thank everyone who contributed to the software.

[1]Institute for Artificial Intelligence, University of Bremen, Germany

This paper presents RoboKudo, an open-source perception framework for mobile manipulation systems that allows to flexibly generate and execute task-specific perception processes that combine multiple vision methods (see Fig. 1). RoboKudo is based on a novel concept that we call Unstructured Information Management (UIM) on Behavior Trees (BTs), short UIMoBT, which is a mechanism to analyze unstructured data with non-linear process flows. We explain how this concept can be applied to robot perception and present the key representational structure of RoboKudo that implements the UIMoBT approach called Perception Pipeline Tree (PPT).

RoboKudo is developed to be included in a perception-action loop of a robot system. The system can state perception tasks to RoboKudo via a query-answering interface. The interface translates a perception task query such as "find a milk box in the fridge" into a specialized perception process, represented as a PPT, which contains a combination of suitable computer vision methods to fulfill the given task.

The contributions of this work are as follows:

1) We propose a novel concept of combining Unstructured Information Management with Behavior Trees called UIMoBT.
2) Applying this concept to robot perception results in a representational structure that we call PPT. We contribute a formalization of PPTs.
3) We provide RoboKudo, an open-source robot perception framework that builds upon the other two contributions. It is written in Python with C++ bindings and supports ROS.

RoboKudo has been successfully evaluated on multiple real robot platforms performing various mobile manipulation tasks (see Section V) and can be flexibly extended for new use cases. To our best knowledge, no other perception system has been shown to achieve comparable capability on real-world mobile manipulation robots in scenarios that require to adapt the perception process to changing perception tasks at run time (see Section II).

II. Related Work

There exists a wide range of approaches to solve perception tasks for specific use cases based on the available sensors, models and autonomy of the system [12, 21, 27]. While frameworks such as TensorFlow [22] or OpenCV [4] immensely support the creation of state of the art vision methods, it still requires other frameworks to make use of the required vision methods at the right time. [17] presented a combination of available deep neural networks for visual perception. They do not address the adaptation to a given perception task and, therefore, focus on the individual vision aspects. In contrast, RoboKudo flexibly selects different PPTs to suit the given task, as demonstrated in Section V. [1] proposed a notable example of a task-adaptable robotic perception system for mobile manipulation robots. Their system has in common with ours that not a single vision method is used to solve a perception task, but rather a combination of methods infers the required object information, such as a pose, color, etc. In contrast to the sequential perception processes of [1], RoboKudo exhibits flexible non-linear perception processes represented as PPTs, which enables robots to support a broader range of use cases.

UIM was originally proposed by IBM [8] and gained a lot of attention when the Watson system won against expert human competitors in the "Jeopardy!" Quiz show. The idea of UIM is to treat incoming data as unstructured information that needs to be analyzed by multiple processing components, called *experts*, that derive partial information from it and provide structure that allows to gain an understanding of the data. Information derived by the different experts is mostly complementary but can also be overlapping or even contradictory. The original implementation of UIM was mostly focused on text inputs and "simple linear flows" [8]. This is especially problematic for systems that need to support non-linear execution flows for their analysis. For example, in perception this could be switching between one-shot and continuous vision subprocesses, reacting to perception or system events, re-iterating or supporting nested recursive processes for the verification of previous results.

BTs are a representation with concise semantics that allows to flexibly switch between tasks [6]. They were originally developed [14] and are currently actively employed in the video game industry to model reactive behavior of artificial agents. BTs have received a lot of attention in robotics in the last years [16, 11, 3, 9]. Due to the flexibility, simplicity, generality and expressiveness to represent task policies, robotics research has investigated the application of BTs in plan execution [25] and control [19, 26], collaborative [23] and social robotics [7], etc. [5] provides a concise overview of BTs with a formal definition and discusses the specifics of applying them in robotics. An excellent, comprehensive introduction on the combination of BTs, AI and Robotics can be found in [6].

III. Perception Pipeline Tree

A. PPT process flow

A behavior tree $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a directed rooted tree with $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges (see an example in Fig. 2). When an edge $(x, y) \in \mathcal{E}$ is directed from x to y, x is called a *Parent* node and y is its *Child* node. The leaf nodes of a BT are *execution* nodes, they perform tasks and can be of type *Action* (▭) or *Condition* (⬭). The non-leaf nodes are *control-flow* nodes, they guide the execution flow through the tree and can be of type *Sequence* (→), *Fallback* (?) or *Parallel* (⇉).

The execution of the tree is driven by a *tick* signal which is generated periodically with a fixed frequency. The tick is propagated through the tree by the rules of different node types and may be forwarded to one or more child nodes. Nodes are only activated when receiving a tick and immediately return either a Success, Running or Failure value. *Action* nodes execute subtasks and immediately return one of the values based on the subtask execution state. These return values are used by the control-flow nodes to guide the propagation of the tick signal. A *Sequence* ticks its children one after another until either one returns Failure or

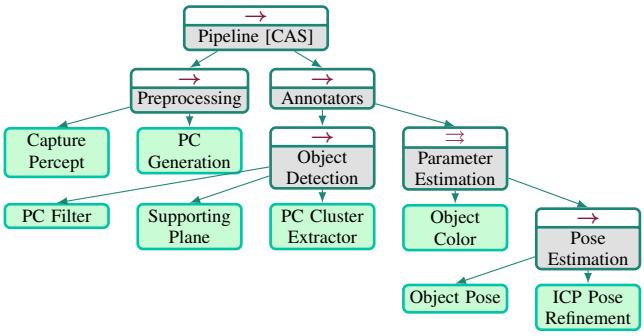

Fig. 2. Perception processes are modeled as Perception Pipeline Trees (PPTs) that are based on BTs. The nodes in the tree represent (1) control structures to control the flow of the process (grey) or (2) individual vision methods like NN-based object detection or RGBD segmentation (green).

Running, or all return Success. A *Fallback* ticks its children sequentially when they return Failure, until one returns either Running or Success, at which point the *Fallback* returns the same value. If no child is left to tick, a *Fallback* returns Failure. A *Parallel* ticks all children sequentially and checks afterwards their return values. If the number of children γ that returned Success is $> n_{success}$, a *Parallel* returns Success. If $\gamma > n_{failure}$ children returned Failure, a *Parallel* returns Failure. Otherwise Running is returned. A comprehensive definition of BTs can be found in [6, 13].

We propose using BTs as a process model for robot perception systems. Vision methods like Neural Networks or classical feature-based approaches and their pre- and post-processors are implemented as *execution* nodes. They can be self-contained, implementing a full vision method, or communicate with external programs via Remote Procedure Calls if they run on a separate machine or in docker containers. The *control-flow* nodes are used to execute the vision methods in the right order by respecting their desired inputs, reacting to failures and switching between processing modes, e.g., switching between one-shot detection and continuous tracking processes. As BTs are hierarchical, they can be used to combine multiple subtrees of perception subprocesses into a complete perception process accomplishing a high-level perception task. We denote a BT that models a perception process as described above a Perception Pipeline Tree (PPT).

Fig. 2 illustrates a simple example of a PPT to detect an object and compute its pose via tabletop segmentation. First, percepts are read from the sensors (*Capture Percept*) and the data is preprocessed to generate a pointcloud (*PC Generation*). Next, the search space is reduced by filtering the depth data (*PC Filter*). A supporting surface for objects is detected (*Supporting Plane*) to extract object candidates above this surface (*PC Cluster Extractor*). After these sequential steps, further analysis on object candidates can be parallelized to, e.g., concurrently compute the color histogram of the object (*Object Color*) and its pose (*Pose Estimation*).

B. PPT data flow

A robot perception system has to be able to infer task-relevant object information from high-frequent, large bandwidth sensory data streams. An approach to cope with these large amounts of data is UIM [2]. In this paper, we propose to realize the UIM approach of multi-expert analysis of large unstructured data through the flexible, modular and reactive representation of BTs. We call this concept UIMoBT (Unstructured Information Management on Behavior Trees). In the following, we describe how data flows in a PPT, realizing the UIMoBT concept.

Execution nodes that read in sensor data or are expert vision methods that extract information from the data are called *Annotators*. Annotators exchange information via a shared data structure that is called Common Analysis Structure (CAS) in the UIM framework. In UIM for perception, a CAS holds the sensor data as well as the annotations produced by the *Annotators*. A CAS is defined as $CAS(\mathcal{O}, \mathcal{M})$ where $\mathcal{O}$ denotes the received sensory observations and $\mathcal{M}$ is a set of annotations that refer to regions in $\mathcal{O}$. Over the course of a running perception process, the CAS is extended with annotations provided by the Annotators (see Fig. 3). $\mathcal{M}$ can contain annotations providing hypotheses about object properties or semantic information about the scene like object classes, shapes, recognized texts and more. Each annotation is grounded into a shared type system that allows the Annotators to request information with the desired semantics.

A PPT is a nested structure that can potentially grow very broad and deep, for example, when using multiple cameras at the same time for different tasks or when employing separate hypothesis verification processes. To ensure modularity, we propose to create one CAS for each subtree that has a certain task-specific semantic purpose. Take, for instance, two subtrees A and B. Subtree A models an object detection sequence that is able to find objects with the head camera of the robot. Subtree B uses images from a camera on the robot's hand to track an object in a visual servoing control loop that guides object grasping. The CAS of subtree A — CAS_A — stores annotations that refer specifically to the object detection steps using the head camera sensor image. Subtree B can look up the results of object detection from CAS_A, transform the estimated object pose into the frame of the hand camera, start tracking the object in the hand camera and then store the annotations and results of tracking in CAS_B. A similar reasoning can be provided to organize a "hypothesize and verify" process [18], where one subtree generates perception results and the second subtree uses these to instantiate a rendering environment for visual verification.

A subtree that has a CAS associated with it is called a *Pipeline*. Each Annotator, thus, belongs to a certain *Pipeline*. In Fig. 2, the entire BT represents one Pipeline. Pipelines are used as a representation of perception subprocesses that are *observation-specific* and *task-centric*.

Observation-specificity allows a Pipeline in the PPT to focus solely on the expected sensory inputs. For example, data from an RGB-only hand camera will be processed by its own Pipeline that will only employ algorithms on RGB data. If data from multiple sensors would be stored in one Pipeline, each Annotator in the Pipeline would have to be parametrized which camera data it should analyze.

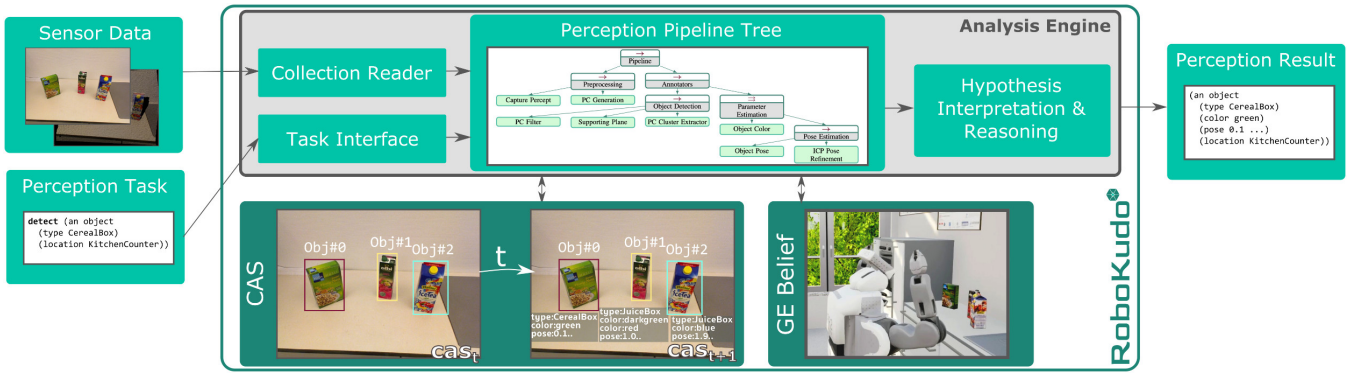

Fig. 3. The architecture of RoboKudo, the proposed perception system. Each perception task received from a high-level control program is processed with a task-specific Perception Pipeline Tree (PPT). During analysis, experts within the PPT share information through a Common Analysis Structure (CAS). The inferred belief of the world feeds into a physics-driven, photorealistic Game Engine-based Belief State (GE Belief) for imagistic reasoning.

A pipeline is task-centric if it explicitly models the subprocesses of a perception task for a specific semantic purpose. Task-centricity allows to avoid storing semantically unrelated data in one CAS, resulting in a more structured association between the perception subprocesses and the information they manage. In contrast, BTs that do not implement the proposed UIMoBT approach often feature a global "blackboard" data storage that is shared with all tree nodes. Task-centricity also promotes explainability, as the system can reason about meaningful subprocesses and model them with an explicit representation, clearly showing the relationship between the different parts of the perception process such as object detection, tracking or verification.

Having introduced PPTs and the CAS, we can now formalize PPTs. A Perception Pipeline Tree (PPT) is a variant of a BT that is defined as $\mathcal{G}(\mathcal{V}, \mathcal{E}, \phi, \mathcal{P})$. An *Annotator* node a, $a \in \mathcal{A} \subset \mathcal{V}$, is an $Action$ or $Condition$ node that accesses a cas for its computation. Function $\phi : \mathcal{A} \rightarrow CAS$ maps the set of *Annotator* nodes to their corresponding CAS. As previously described, a cas is scoped by a *Pipeline* $p \in \mathcal{V}$, which is a specialized *Sequence* node. The set $\mathcal{P}$ contains tuples (p, cas) that associate each *Pipeline* p to an individual cas. Therefore, ϕ is defined by traversing the PPT upwards from the input $v \in \mathcal{V}$ until the currently visited node v_i is found in tuple (p, cas) with $p = v_i$. In that case, cas is returned by ϕ, otherwise the upwards traversal is continued. Each *Annotator* a must be a direct or an indirect child of *Pipeline* p. In each valid PPT, $|\mathcal{P}| > 0$ must hold.

IV. Architecture of the Framework

The previous section presented PPTs, the cornerstone representation of the proposed perception framework, RoboKudo. This section describes the architecture of the framework and its external interfaces, as illustrated in Fig. 3.

The inputs of the framework are (1) the data provided by the sensors of the robot (*Sensor Data*) and (2) a task description of the object or phenomena that shall be perceived to support the current step of the robot action plan (*Perception Task*). A robot performing mobile manipulation actions needs to solve a plethora of perception tasks to acquire accurate knowledge about the world and to detect changes therein, e.g., *"detect a cereal box on the table"* or *"track the blue cup when the human is pouring from it"*. RoboKudo takes as input a vague description of a perception task in a form of a list of nested key-value pairs.

The *Analysis Engine (AE)* is the main processing entity of RoboKudo. It collects sensor data (*Collection Reader*) and processes it based on the incoming perception tasks. The *Task Interface* translates a perception task description into a modification and parametrization of the PPT suitable for solving the given task, as described in Subsection IV-A. Then the AE runs the PPT in a real-time loop (*Perception Pipeline Tree*). The *CAS* data structures facilitate information exchange between the nodes in the PPT. After the execution of the PPT is finished, the annotated hypotheses are collected and interpreted (*Hypothesis Interpretation & Reasoning*). In this step, the AE fuses the results of all the annotators.[1] Resulting information is stored in a hybrid belief state (*GE Belief*), similar to [18].

Ultimately, the final result is communicated to the other components of the robot system (*Perception Result*). Note that this can be information requested by the task-level controller of the robot as well as the motion controller that uses continuous perception feedback to control robot body parts in a real-time closed loop (see Section V).

A. Task Interface: Mapping of Perception Tasks to PPTs

A PPT is designed by considering which *Annotator* node can provide the required perceptual information for a given perception task. The ordering of the Annotators in the tree depends on their inputs and outputs w.r.t. their Pipeline's CAS, as each Annotator requires specific sensor data or inputs from other Annotators. If Annotator a requires only a single input that another Annotator b provides, both Annotators are put under a Sequence node as its direct or indirect children. If Annotators a and b do not depend on each other but both require the output of another Annotator c, a and b are put in a Parallel node after c to speed up computation. If Annotators a and b can both potentially provide an

[1] Works such as [20] show how to fuse annotations in this step.

output required by c but only one would suffice, they can be composed into a Fallback node to represent priority-based execution of alternative approaches for generating the required output. Note that Annotator nodes in RoboKudo can run in either a blocking or a threaded mode, where the latter does not block the execution of the tree and, therefore, allows true parallelization.

A PPT can be changed during run time by the Task Interface. This allows the system to analyze the incoming perception task and change subtrees based on the Annotators and control-flow nodes required for this task. The current perception task can be accessed by the Annotators to select at run time the correct parametrization for the given task, e.g., to adjust hyperparameters to suit the specified problem class or to choose a suitable object detection model.

V. Experiments

As this paper presents a framework for combining individual vision methods in a flexible, modular, intuitive and easy to use manner and does not propose a novel vision algorithm, none of the commonly used perception datasets was suitable to evaluate the proposed approach. The evaluation dataset for this paper has to originate from an embodied agent performing mobile manipulation actions in the real-world and has to address a variation of vision tasks. Unfortunately, common datasets typically concentrate only on one specific task. Therefore, we designed a multi-step scenario involving mobile manipulation and human-robot interaction actions that require a variety of perception tasks, including one-shot and continuous perception as well as object detection, human activity recognition and body configuration estimation. All perception tasks are tackled by the same running instance of RoboKudo for the complete duration of the experiment, whereby the used perception processes are adapted at run time towards the currently active perception task. We report on the variations of vision method combinations that employ parallelization(⇉), looping(↻) and subtrees(⊞), as well as provide statistics on execution success rates and run times.

All PPTs used in the scenario have the common parent subtree, shown in Fig. 4. The ⟨*Subtree*⟩ node is a placeholder where vision task specific subtrees are inserted at run time.

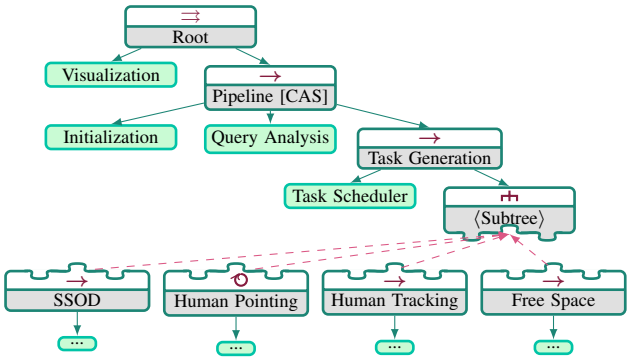

Fig. 4. Root PPT with an embeddable task subtree node.

The key parts of the experiment can be seen in Fig. 5. A Toyota HSR robot assists a human in carrying groceries to a

Fig. 5. Photos showing the key steps of the experiment. (1) One-shot object detection and pose estimation. (2) Continuous perception to check if the human is pointing at one of the detected objects. (3a) Continuous perception for human tracking and following and (3b) pointing detection for intended storage place. (4) Free space estimation and placing. See the video at `https://robokudo.ai.uni-bremen.de/rkop`.

storage place. The steps of the robot plan are the following: (1) Perform single-shot object detection and pose estimation (*SSOD*) for unknown objects. (2) Move the head to look in the direction of the human. (3) Continuously detect and monitor the human until she points at one of the previously detected objects (*Human Pointing*). (4) Grasp the object pointed at from the table. (5) Move the head to look at the human again. (6) Follow the human in a perception-control loop and concurrently monitor if the human is pointing to a storage place (*Human Tracking*). (7) Move the head to look at the storage place. (8) Perform single-shot free space estimation on the storage place (*Free Space*). (9) Place the object in a free space in the storage location. In steps (1), (3), (6) and (8) the proposed perception system is queried. For motion planning and control we use Giskard [26].

SSOD is done by exploiting semantic, spatial knowledge about the environment to filter RGBD data with the *Region Filter* and detecting objects by clustering with *PC Clustering*. Afterwards, the pose is estimated and the object information is stored internally.

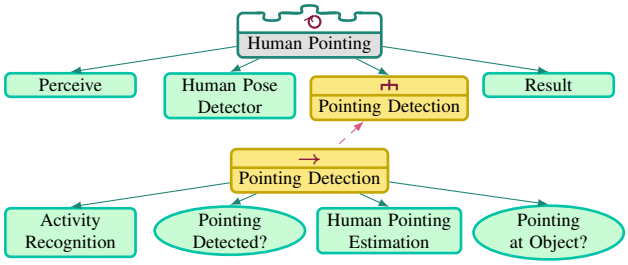

Fig. 6. Human pointing task subtree with pointing detection subtree.

Human Pointing (see Fig. 6) features a state of the art human detection and keypoint estimation(joints) from YOLOv8 [15] with the *Human Pose Detector*. To recognize a human pointing activity, we use the CLIP multi-modal vision

model [24] to query the image of the detected human for typical activities ("standing", "pointing", etc.) in the *Activity Recognition* annotator. If a pointing activity is detected and annotated in the CAS(*Pointing Detected?*), we employ a self-implemented *Human Pointing Estimation* which uses the estimated joint positions of the detected human to estimate the pointing direction. If the pointing targets a previously detected object (*Pointing at Object?*), the PPT returns a description of it to the robot control for grasping (*Result*).

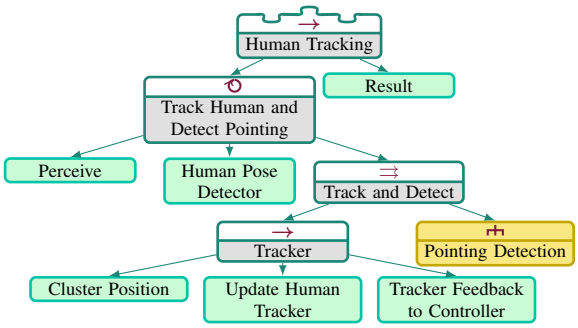

Fig. 7. Human tracking task subtree reusing pointing detection from Fig. 6.

Human Tracking is the most complex tree (see Fig. 7). It concurrently (*Track and Detect*) (a) tracks the human in the camera frame and feeds her 2D position to the control system and (b) uses the *Pointing Detection* subtree to check if the human is pointing at a storage place. Before (a) and (b), percepts are received (*Perceive*) and humans are detected with *Human Pose Detector*. For (a) we developed a particle filter approach that heuristically filters human detections and associates the observation z_t^X with the filter, given $\hat{s_t}$ as the estimated position:

$$z_t^X = \underset{z \in Z_t}{\operatorname{argmin}} \|z - \hat{s_t}\|.$$

In the final step, the *Free Space* subtree reuses the *Region Filter* of the *SSOD* subtree to recognize if known storage regions contain 3D points, implying blocked storage places.

As can be seen in the PPTs, some of the annotators are reused in different perception tasks, which demonstrates the modularity of our approach. Because perception processes are modeled as PPTs, one can easily visualize them and their runtime state. Designing new processes into maintainable structures is supported by having to abide by the strict semantics of behavior trees.

We conducted experiments with eight different persons and ten household objects (see Tab. I). Experiments where motion plans failed or hardware issues occured were repeated. A total of 31 experiments were analyzed.

TABLE I: Quantitative results of our real world experiments

	Pointing	Grasping	Following	Free Space	Placing	Overall
Success	29/31	29/29	23/29	23/23	22/23	22/31
%	93.55	100	79.31	100	95.65	70.96

The process handling of the framework was successful in all runs. Most failed runs (4) were caused by manipulation (Object slipped) errors in different experiment phases. Three failed runs were caused by lost human tracking in the following task, when humans were distant from the robot. Two runs failed due to incorrect object pointing estimation.

The system processed 4418 percepts. Component run times have been recorded. Components in the experiment which are extracting key information and have an average run time above 2ms are depicted in Figure 8. The complex CLIP neural network for visual concepts used in the *Activity Recognition* node has the highest impact on the run time.

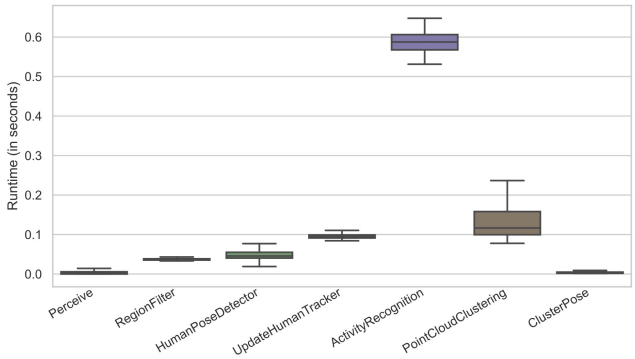

Fig. 8. Boxplot showing runtime distributions for key vision components in the experiment. The complex CLIP-based Activity Recognition, while versatile, is the most time-consuming. The control PC running the framework contains an i7-12700K CPU, 32 GB RAM and an RTX 3080 10GB GPU.

Additional evaluation material is accessible on the webpage mentioned in the abstract, which also provides results from experiments on PR2 and Tiago, as shown in Fig. 9.

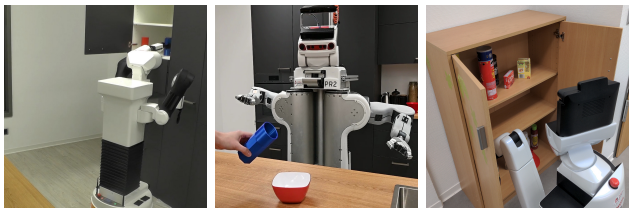

Fig. 9. Photos from the auxiliary experiments with different robots performing mobile manipulation actions. (left) One-shot detections for pick & place and pouring actions. (middle) A continuous perception task with object tracking. (right) HSR in a RoboCup@Home scenario. Details are available at `https://robokudo.ai.uni-bremen.de/rkop`.

VI. Conclusion

This paper introduced RoboKudo, an open-source framework for robot perception that generalizes towards diverse perception tasks. The central idea of the approach is to model perception processes through Behavior Trees and leveraging the Unstructured Information Management principle for data communication between nodes. The concepts are implemented in data structures we call PPTs. We presented the architectural foundation of RoboKudo and demonstrated how perception task queries seamlessly map onto PPTs. Empirical validation through an extensive case study highlighted the versatility of our framework, effectively supporting a wide array of perception tasks from single-shot to continuous processes as well as a diverse set of reusable vision expert methods enabling robots to dynamically adapt to a broad range of perceptual requirements.

REFERENCES

[1] Ferenc Bálint-Benczédi. "Task-adaptable, Pervasive Perception for Robots Performing Everyday Manipulation". PhD thesis. 2020.

[2] Michael Beetz et al. "RoboSherlock: Unstructured information processing for robot perception". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[3] Bruno Bouchard et al. "Modeling Human Activities Using Behaviour Trees in Smart Homes". In: *Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference*. PETRA '18. Association for Computing Machinery, 2018.

[4] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[5] Michele Colledanchise and Lorenzo Natale. "On the Implementation of Behavior Trees in Robotics". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5929–5936.

[6] Michele Colledanchise and Petter Ögren. "Behavior Trees in Robotics and AI: An Introduction". In: *CoRR* abs/1709.00084 (2017). arXiv: 1709.00084.

[7] Sara Cooper and Séverin Lemaignan. "Towards using Behaviour Trees for Long-term Social Robot Behaviour". In: *17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2022.

[8] David Ferrucci and Adam Lally. "UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment". In: *Natural Language Engineering* 10.3–4 (2004).

[9] Razan Ghzouli et al. "Behavior trees in action: a study of robotics applications". In: *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*. 2020, pp. 196–209.

[10] Florenz Graf et al. "Toward Holistic Scene Understanding: A Transfer of Human Scene Perception to Mobile Robots". In: *IEEE Robotics and Automation Magazine* 29.4 (2022), pp. 36–49. DOI: 10.1109/MRA.2022.3210587.

[11] Danying Hu et al. "Semi-autonomous simulated brain tumor ablation with RAVENII Surgical Robot using behavior tree". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[12] Alexandros Iosifidis and Anastasios Tefas. *Deep learning for robot perception and cognition*. Academic Press, 2022.

[13] Matteo Iovino et al. "A survey of Behavior Trees in robotics and AI". en. In: *Robotics and Autonomous Systems* 154 (Aug. 2022).

[14] Damian Isla. "Handling complexity in the Halo 2 AI". In: *Game Developers Conference 2005 Proceeding*. 2005. URL: https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai.

[15] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[16] Simon Jones et al. "Evolving Behaviour Trees for Swarm Robotics". In: *Distributed Autonomous Robotic Systems: The 13th International Symposium*. Ed. by Roderich Groß et al. Cham: Springer International Publishing, 2018, pp. 487–501.

[17] Chung-Yeon Lee et al. "Visual Perception Framework for an Intelligent Mobile Robot". In: *17th International Conference on Ubiquitous Robots (UR)*. 2020.

[18] Patrick Mania et al. "Imagination-enabled Robot Perception". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

[19] Alejandro Marzinotto et al. "Towards a unified behavior trees framework for robot control". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[20] Daniel Nyga, Ferenc Balint-Benczedi, and Michael Beetz. "PR2 looking at things — Ensemble learning for unstructured information processing with Markov logic networks". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[21] Miguel Oliveira et al. "A perceptual memory system for grounding semantic representations in intelligent service robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014.

[22] Bo Pang, Erik Nijkamp, and Ying Nian Wu. "Deep learning with tensorflow: A review". In: *Journal of Educational and Behavioral Statistics* 45.2 (2020).

[23] Chris Paxton et al. "CoSTAR: Instructing collaborative robots with behavior trees and vision". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[24] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[25] Francesco Rovida, Bjarne Grossmann, and Volker Krüger. "Extended behavior trees for quick definition of flexible robotic tasks". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[26] Simon Stelter, Georg Bartels, and Michael Beetz. "An open-source motion planning framework for mobile manipulators using constraint-based task space control with linear MPC". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022.

[27] Abolfazl Zaraki et al. "Design and evaluation of a unique social perception system for human–robot interaction". In: *IEEE Transactions on Cognitive and Developmental Systems* 9.4 (2016), pp. 341–355.