IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 8, 2020, accepted June 16, 2020, date of publication June 22, 2020, date of current version July 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3003991

A Software Architecture for Service Robots Manipulating Objects in Human Environments

CHANGJOO NAM[1], (Member, IEEE), SEOKJUN LEE[2], JEONGHO LEE[1], (Student Member, IEEE), SANG HUN CHEONG[1], (Member, IEEE), DONG HWAN KIM[1], (Member, IEEE), CHANGHWAN KIM[1], (Member, IEEE), INCHEOL KIM[2], AND SUNG-KEE PARK[1], (Member, IEEE)

[1]Robotics and Media Institute, Korea Institute of Science and Technology, Seoul 02792, South Korea
[2]Department of Computer Science, Kyonggi University, Suwon 16227, South Korea

Corresponding author: Sung-Kee Park (skee@kist.re.kr)

This work was supported by the Development of ICT Technology Program of Institute for Information & Communications Technology Planning & Evaluation (2020-0-00096, Robot task planning for single and multiple robots connected to a cloud system).

ABSTRACT This paper presents a software architecture for robots providing manipulation services autonomously in human environments. In an unstructured human environment, a service robot often needs to perform tasks even without human intervention and prior knowledge about tasks and environments. For autonomous execution of tasks, varied processes are necessary such as perceiving environments, representing knowledge, reasoning with the knowledge, and planning for task and motion. While developing each of the processes is important, integrating them into a working system for deployment is also important as a robotic system can bring tangible outcomes when it works in real world. However, such an architecture has been rarely realized in the literature owing to the difficulties of a full integration, deployment, understanding high-level goals without human interventions. In this work, we suggest a software architecture that integrates the components necessary to perform tasks by a real robot without human intervention. We show our architecture composed of deep learning based perception, symbolic reasoning, AI task planning, and geometric motion planning. We implement a deep neural network that produces information about the environment, which are then stored in a knowledge base. We implement a reasoner that processes the knowledge to use the result for task planning. We show our implementation of the symbolic task planner that generates a sequence of motion predicates. We implement an interface that computes geometric information necessary for motion planning to execute the symbolic task plans. We describe the deployment of the architecture through the result of lab tests and a public demonstration. The architecture is developed based on Robot Operating System (ROS) so compatible with any robot that is capable of object manipulation and mobile navigation running in ROS. We deploy the architecture to two different robot platforms to show the compatibility.

INDEX TERMS Service robots, manipulation planning, AI reasoning methods.

I. INTRODUCTION

For decades, there have long been extensive research efforts on robotic manipulation of objects. The areas of research include hand and grasper design [1], grasp planning [2], task planning [3], motion planning [4], control [5], and perception [6]. As a result, object manipulation becomes one of the most successful applications in robotics. Many commercial products and open-source software are released and used in industrial and domestic environments. Examples are Pick-it [7] for perception, MoveIt for motion planning [8], and GraspIt for grasp planning [9].

On the other hand, the success of the DARPA Robotics Challenge (DRC) 2015 shows us a bright future of robots that can work in human environments. However, several reports from the challenge [10]–[12] tell us that interventions of human operators were necessary and caused major problems (e.g., fall, reset, large delay, task failure). Atkeson *et al.* (2018) [10] conclude that a greater autonomy is expected in the future than the human-moderated operation. These

The associate editor coordinating the review of this manuscript and approving it for publication was Shaohua Wan.

facts motivate the need of robots that can work in human environments with increased autonomy.

In this work, we present our software architecture for a robot performing navigation and manipulation tasks in human environments. Our goal is to grant autonomy to the robot across all processes so that the robot can provide services to humans with no aid from an operator. Thus, we aim to implement and integrate the processes for perception, knowledge representation, reasoning, task planning, and motion planning. We also want to develop an architecture that is compatible with any robot platform capable of mobile navigation and manipulation.

Higher-level components regarding knowledge, reasoning, and task planning for robots have less studied than lower-level components such as robot hardware and control. In the report [10], it is pointed out that the majority of DRC participants were specialized in robot hardware and control but weaker in perception, reasoning, and autonomy. Our review in Section II also shows this tendency as many of existing architectures lack of the ability to reason. Atkeson *et al.* (2018) [10] also discuss about the underestimated importance of the work in academia on deploying perception and autonomy software to a real robot. Alterovitz *et al.* [13] point out the lack of planning capabilities of commercially deployed robots, which limit the use of the robots in real world settings. Several recent works present progresses in those areas such as object pose estimation [6], reasoning and inference [14], [15], and task and motion planning [16], [17]. However, those state-of-the-art approaches have not been fully integrated into a deployable system, to the best of our knowledge.

Therefore, the objective of this work is to suggest a software architecture that integrates the components necessary to perform tasks by a real robot without human intervention. Also, we aim to provide implementation details and lessons learned to related communities. We integrate deep learning based perception, knowledge representation and processing, symbolic task planning, and motion planning into a software architecture to deploy it to a physical humanoid robot with a mobile base. Using our architecture, the robot can generate abstract task plans based on reasoning with collected knowledge from perception. The architecture bridges the abstract task plans and geometric motion plans through the task-motion interface that computes goal poses of the robot autonomously. If motion planning or execution of planned motions are not successful, the architecture modifies the task plans using feedback about the failures. We perform several experiments in environments where multiple objects need to be manipulated for fulfilling requests of users (Figure 1).

The main contribution of this work is in developing the software architecture that enables autonomous services of manipulation tasks without human supervisions. It is also our contribution that validating the architecture through a public demonstration performing nontrivial tasks which are not merely episodic but evolving depending on the result of preceding tasks. We develop the architecture in Robot

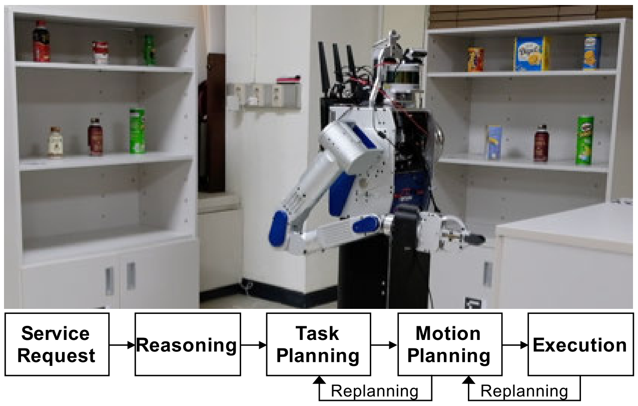


FIGURE 1. (Top) An example of human environments where the rob ot provides object manipulation services. The robot performs a fulfillment service to a customer. (Bottom) Different processes should be done for the service: (i) reasoning about the location of an ordered item, (ii) planning for navigation and manipulation tasks, (iii) detecting objects from streamed images, and (iv) planning for motions of the manipulator to pick and place objects. Replanning may occur if failures occur.

Operating System (ROS), so it is compatible with different robot hardware platforms, where the compatibility is shown with two different robot platforms. We also contribute to the robotics community by providing implementation details of the architecture deployed to real robots and lessons learned from the deployment.

II. RELATED WORK

Ontology-based knowledge model is widely used in robotic systems. OpenCyc [18] provides a general upper ontology written in OWL-DL. Knowrob [19] and ORO [20] expand the upper ontology proposed in [18] for indoor service robots. KnowRob provides a variety of classes representing events, objects, tasks, motions, mathematical concepts, and etc. KnowRob can represent details of individuals such as components and capabilities of robots and affordances of objects. There have been many applications and demonstrations presented that are based on KnowRob. For example, cloud-based [21] and web-based knowledge services [22] are used to perform tasks like making a pancake and serving a patient. ORO provides object and action classes and is used in demonstrations to perform tabletop tasks like packing and cleaning. OUR-K [23] and OMRKF [24] provide context, space, object, action, and feature classes. They divide knowledge into three layers: meta-ontology; ontology instance; and ontology layer. Their knowledge is used in a delivery scenario. Although the knowledge models show their compatibility with robotic systems, the capability of dealing with task failures from inaccurate and uncertain perception has not been demonstrated.

Task planning focuses on generating high-level discrete actions to achieve a goal. Classic AI planners (e.g., Fast Forward [25], Fast Downward [26]) generate a sequence of actions in an abstract action space. They reason about a set of feasible actions in each state and their effects on the environment incurring transitions to other states. Although they have generality across different domains and flexibility to failures, they have limited capabilities in reasoning about metric spaces. On the other hand, motion planning is specialized for finding a sequence of robot configurations which result in continuous motions. Sampling-based motion planning draws samples from the configuration space of the robot and connects the samples gradually to cover the entire space [27]. Optimization-based motion planning optimizes an initial path and converges to a locally optimal path quickly [28]. Both planning approaches could fail to find a trajectory given a goal pose especially if the robot needs to navigate among objects while avoiding collisions with them. If motion planning fails, the goal pose or the task plan requiring to achieve the goal pose may need a modification. However, the scope of motion planning is limited to generating motions so dealing with motion planning failures need to be done by other upper-level algorithms.

Recently, a significant effort has been made to fill the gap between task and motion planning. Task and motion planning (TAMP) approaches show successful achievements for filling the gap by generating symbolic task plans that are feasible [17], [29]–[31]. However, TAMP frameworks proposed in the literature have not been tested sufficiently in a full-stack architecture where sensing, reasoning, planning, and execution interplay to achieve a mission. Rather, tests are fragmented and less realistic. Only the coupling of task and motion planning has tested to prove the concept of TAMP and the efficiency of proposed pose computation methods. The TAMP approach is now at a mature stage so deploying it for real world scenarios is paramount.

On the other hand, an extensive number of deep learning models are proposed for object detection. One of the most successful approaches is region-base Convolutional Neural Networks (CNNs). R-CNN (Regions with CNN features) [32] uses a selective search [33] for a region proposal, CNNs to extract features, and SVM [34] for classification. Fast R-CNN [35] and Faster R-CNN [36] are proposed to improve detection performance and speed of R-CNN [32]. Yolo [37] achieves a faster detection rate of 45 fps which is appropriate for real-time applications. However, it has a drawback of a low recall ratio and poor accuracy. Yolo-v3 [38] resolves the problem by adopting predefined boxes with variable sizes and improves detection performance by using focal loss in the learning phase.

One can find grasping poses for an object from the estimated pose of the object or directly from images or 3D point clouds. Jain and Argall (2016) [39] propose a method that autonomously detects grasp poses of unseen objects. From point clouds from objects, the method categorizes object shapes into predefined geometric shape primitives (sphere, cylinder, or cuboid). In [40], grasps of novel objects are found from RGB-D images. The method quickly separates objects in the scene and generates candidate grasp rectangles. Among the candidates, a final grasp is chosen using a random forest model.

Many methods have been proposed recently to use deep learning techniques. In [41], the authors propose a method to detect grasps using two deep convolutional neural networks (CNNs). A deep CNN extracts features and sends them to a shallow CNN, which predicts the grasp pose. Chu *et al.* (2018) [42] propose a method to predict grasp candidates for unseen objects in RGB-D images. The method predicts multiple candidate grasps. Multiple grasps help generate various grasp and motion plans in subsequent planning processes. A comprehensive review of many recent works presented in [43].

The fields of study mentioned above have thrived independently. Along their advances, some of them have been combined together for robotics applications. The TAMP framework is an example of successful combinations. High-performance perception is a primary capability of autonomous robots so has been used by many robots across different domains. On the other hand, there has been a line of research on architecture designs to fully integrate the components necessary for autonomous execution of tasks.

In [44], the authors propose a hybrid software architecture (NUClear) for robots that centers on a message-passing system where the advantages of having a global store system (i.e., a blackboard system) are incorporated. This work focuses on designing the communication system to improve the performance of the architecture such as interface sizes and memory usages. On the other hand, our work focuses on the functionality and autonomy of the architecture so how given tasks are completed successfully without human aids. Thus, the work in [44] could help us optimize the communication system of our architecture to increase the efficiency of the architecture in low-performance systems and decrease communication latency.

In [45], a decentralized robotic architecture (SERA) is proposed for heterogeneous multiple robots and implemented in ROS. The architecture supports self-adaptation of individual robots and a team of robots by refining and reconfiguring plans and models. The validation of the architecture is done by 21 human experts in robotics. Also, the authors perform simulations and physical robot experiments. The architecture is designed to work with different robotic platforms, but the physical robot experiment is done with a single platform. Although the architecture has components for perception, task and motion planning, and reasoning for infeasible plans, it does not model knowledge systematically to be used in planning and execution. In addition, it requires human intervention in order to decompose a global mission into multiple local missions.

In [46], the authors review recent results in developing functionalities of mobile healthcare robots such as perception, navigation, human-robot interface, AI, and etc. They discuss about offloading heavy computation of the functionalities for delay-sensitive and communication-intensive tasks through edge computing. Our architecture does not rely on task offloading but is rather self-containing without external aids. However, edge- or cloud-computing can help add more

TABLE 1. Comparisons with existing architectures. The four criteria are chosen based on the contribution of our proposed work (i) aiming to develop a full-stack architecture including perception, reasoning, task planning, and motion planning, (ii) requiring no human aid while performing tasks, (iii) independent from robot platforms, and (iv) demonstrating in the real-world with physical robots.

Architecture	Full-stack (missing component)	No human intervention	Platform-agnostic	Real-world demonstration
NUClear [44]	No (knowledge base, reasoning, UI)	Not clear	Yes	No
SERA [45]	No (knowledge base)	No	Yes	Yes
LAAIR [47], [48]	Yes	No	Yes	Yes
COROS-based [49], [50]	No (reasoning)	No	Yes	Yes
Fraiser [51]	No (knowledge base, reasoning)	Yes	Not clear	Yes
Proposed	Yes	Yes	Yes	Yes

functionalities to the architecture or increase the number of domains that the robot can work for.

In [47], [48], a layered software architecture (LAAIR) is proposed that enables robots to react to human interactions quickly. It has a top-level module that can flexibly access both reactive and deliberative controllers. Although the architecture has almost all functionalities to perform tasks independently, human operators are involved to provide high-level goals such as "navigate to the target" and "track the target".

In [49], [50], a component-based architecture implemented in ROS is proposed for service robots. It provides abstractions for developing robot applications by adding a web service interface where developers can interact with robots without backgrounds in ROS. The architecture has a knowledge base but it is merely a storage for information since no knowledge representation and reasoning is used with the information. Also, the demonstration with a Turtlebot includes only simple navigation tasks (e.g., delivery between indoor offices) where the start and goal locations should be given by a human user. The ability to reason will enable the robot to know which people needs the delivery service and where they are without human inputs.

The system architecture proposed in [51] supports robot manipulation services similar to our work. Like [48], the architecture aims to perform tasks in various scenarios of the RoboCup@Home league. The authors point out that the majority of research in manipulation focus on solving each of the problems of perception, grasping, motion planning, and user interfaces. Some work propose integrated systems but they still need human intervention [52]–[54] or a hardcoded 3D map of objects [55]. On the other hand, the architecture proposed in [51] aims to develop a complete system including the user interface through natural language. The architecture is tested in real-world environments using a mobile manipulator in the RoboCup league and show competitive results. However, the architecture lacks the ability to store knowledge and reason with them. The method used for task planning is not discussed so not clear but seems to be implemented simply like finite-state machines. Also, the test scenario introduced is relatively simple as it does not consider cluttered environments, different approaching angles for grasping, navigation of the mobile base, and searching where to place picked objects.

Based on the review of existing architectures, we find that there exists a substantial gap between the potential of robot autonomy and the reality and the following concerns should be addressed. First, existing architectures often do not have all functionalities for perception, knowledge representation, reasoning, task and motion planning, and user interface. Except [47], [48], they lack more than one of the functionalities. Second, many of the existing architectures and the work done for DRC 2015 [10]–[12] require some level of human intervention to complete given tasks. Another concern is the underrepresented importance of demonstrations with physical robots for nontrivial tasks in real-world environments. Moreover, most of the previous work do not show physical robot experiments across different robot hardware platforms although they claim that their architectures are not tied to particular platforms. Therefore, our work complements the existing work (as summarized in Table 1) by integrating perception, knowledge engineering, reasoning, and task and motion planning/replanning, which is deployed to different robot platforms to plan and perform for object manipulation without human supervision.

III. SYSTEM ARCHITECTURE

In this section, we give an overview of the software architecture and data flow between its components. Then we describe the functions and implementations of the components.

A. OVERVIEW

The architecture is described in Figure 2 where the data flow between the components is shown in Figure 3. Multi-Modal Sensing (MM) represents inputs from the robot sensors such as vision, tactile sensors, and joint encoders. Any sensor inputs can be added to the component. Perceptual Reasoner (PR) is in charge of generating grasp information of recognized objects. Working Memory (WM) contains the perception information of objects and environments that are currently observed. For example, the grasp information of an object that is currently detected stays in the working memory. Once the object disappears from the view of the robot, the information is also deleted. Manipulation Knowledge Manager (KM) collects knowledge from perceived data in order to maintain knowledge bases and a world model. Given a goal, KM reasons about the context which is used by Manipulation Task Manager (TM) to generate task plans. Action Manager (AM) computes feasible motions of the robot for both manipulation and navigation. However, AM finds trajectories of joints only if goal poses of the robot end-effector

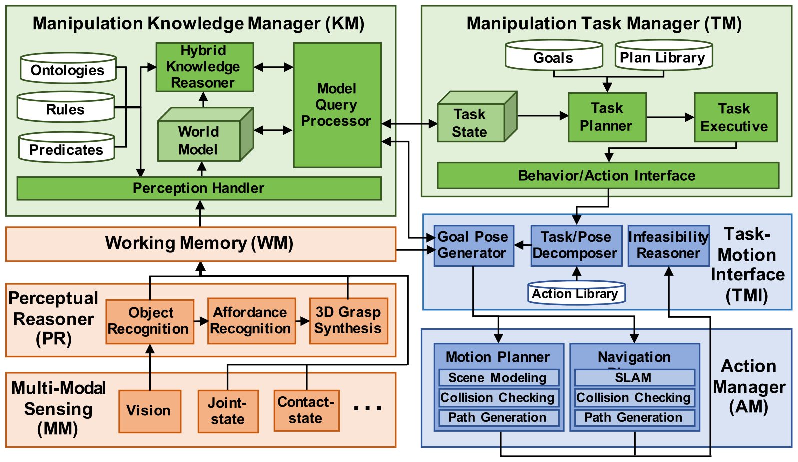


FIGURE 2. The software architecture for service robots performing manipulation and navigation tasks. Multi-Modal Sensing (MM) represents inputs from the robot sensors. Perceptual Reasoner (PR) is in charge of detecting objects and generating grasp information of them. Working Memory (WM) contains the perception information at the moment. Knowledge Manager (KM) collects information and maintains knowledge bases and a world model. Given a goal, KM reasons about the context which is used by Task Manager (TM) to generate task plans. Task-Motion Interface Manager (TMI) autonomously generates the goal poses of the robot for the task plans. Action Manager (AM) computes feasible motions of the robot for the goal poses.

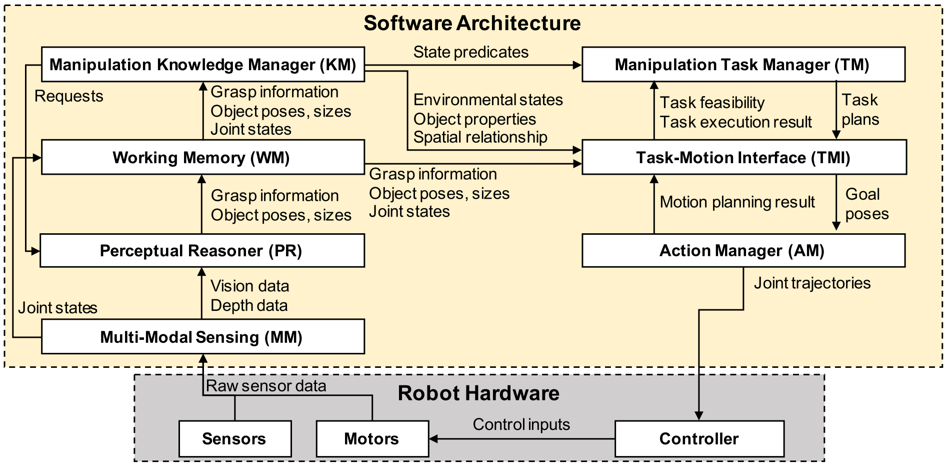


FIGURE 3. The data flow between components. The input to the architecture is sensory information and the output is a set of joint trajectories of a robot. The robot receives the trajectories and performs planned tasks by executing the trajectories.

and the mobile base are given. Thus, Task-Motion Interface Manager (TMI) autonomously generates the goal poses based on the abstract symbolic task plans. In the course of or after task execution, changes in the environment and robot are recognized by MM and PR. The updated states are used for subsequent task planing or replanning.

In addition to the increased autonomy, the advantage of the architecture is in its modular design. In order to support modularity, the architecture takes the message-passing communication system where the output of a component is used as the input of another component [44] without a global data store system. Each component is implemented in ROS so any algorithm can be used for the component if the communication protocols between components are met. The architecture also can work with different robot platforms if the robot and sensor models (e.g., described in URDF, standing for Unified Robot Description Format) are available.

According to the classification of robot architectures in [56], our architecture falls into the component-based model which stands on the composition of components and their interactions. The interactions occur between a requester (e.g., navigation component) and a provider (e.g., a controller). The requester—provider interaction occurs across the our architecture: KM requests a perception job to PR, TM requests for checking the feasibility of task plans to TMI, TMI asks AM to find feasible trajectories of joints, and etc.

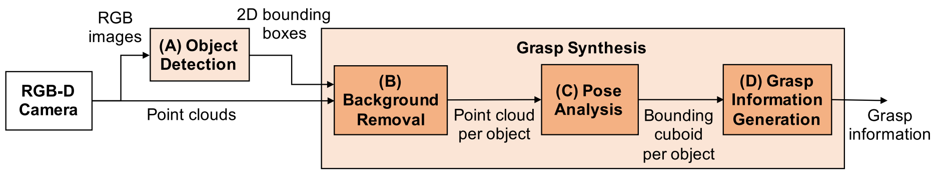


FIGURE 4. The process of Perception Reasoner. RGB-D images and point clouds are used to detect objects and their grasp information. (A) PR detects objects in the workspace. (B) PR removes the background points which are not from objects to obtain the point could for each object separately. Then PR segregates the 3D points of objects into multiple clusters. (C) PR generates a bounding cuboid of the point set for each object. The pose of the cuboid in the world coordinate system is computed. (D) The grasp information of each object is computed.

B. FUNCTIONS AND IMPLEMENTATIONS OF THE COMPONENTS

1) PERCEPTUAL RECOGNITION

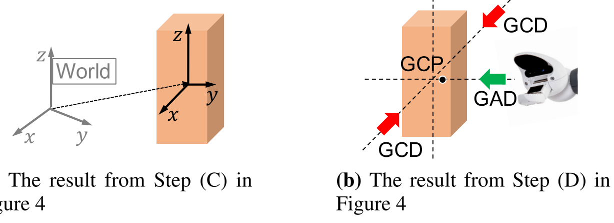
Multi-Modal Sensing (MM) delivers raw sensing data to components in the architecture in need of the data. In our implementation, MM sends images and depth information about the workspace to Perceptual Reasoner (PR). Also, MM sends joint states of the robot to Working Memory (WM). WM maintains information about the current state such as the object and joint information at the moment. Among them, necessary information are maintained by Knowledge Manager (KM) to be used in the future by other components. The inputs and outputs of MM and WM could change flexibly depending on the sensors available to the robot such as tactile or force/torque sensors.

PR processes visual sensing data in order for the robot to recognize and manipulate objects. PR provides pose, size, and grasp information of objects. Through the steps (A)–(D) shown in Figure 4, PR detects objects from images and then analyzes point clouds to compute grasp information of the detected objects.

(A) For object detection, PR can employ any off-the-shelf algorithm that finds object classes (i.e., labels) from 2D images and bounding boxes of the detected objects in the images. Also, the speed of detection should be sufficiently fast for real-time applications.

(B) For grasp synthesis, PR uses the depth information from the sensor. PR first removes the background points in the point cloud which are not from objects. In our implementation, PR finds those background points by estimating the planes around the objects such as the tabletop and shelves. After that, PR segregates the 3D points of objects into multiple clusters based on the locations of the 2D bounding boxes in the images (we use k-Nearest Neighbors [57]). Dominant points making up the largest cluster within a bounding box are selected as the 3D point set for the corresponding object.

(C) After processing the point cloud, PR analyzes the pose of each object to synthesize the grasp information. A bounding cuboid for each point set of an object and its three axes are computed (we use Principal Component Analysis (PCA)). Figure 5a shows the bounding cuboid with its three axes. Initially, the 6D pose of the cuboid is in the camera's coordinate system (i.e., relative to the camera pose). PR computes the 6D



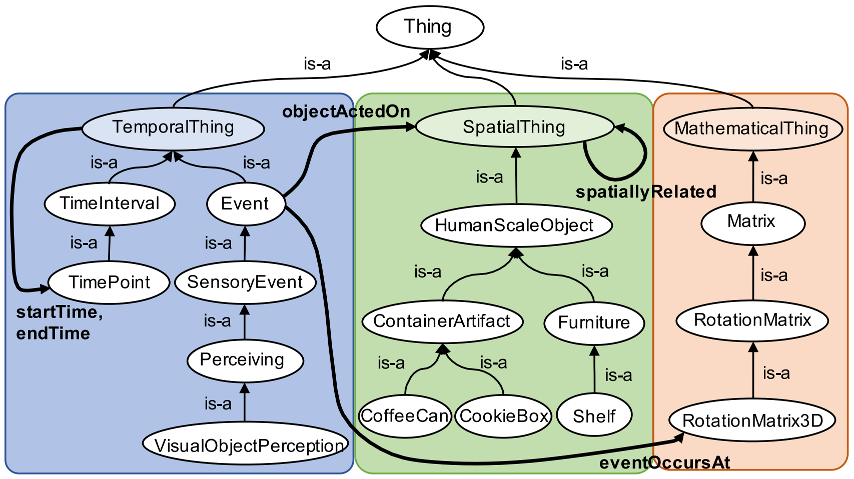
(a) The result from Step (C) in Figure 4

(b) The result from Step (D) in Figure 4

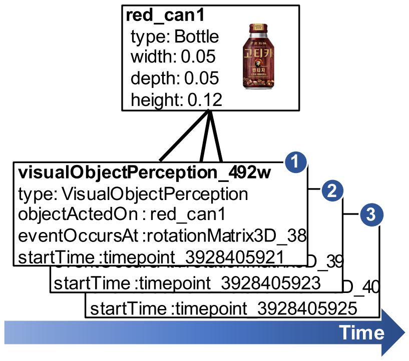
FIGURE 5. The results from PR. (a) The 6D pose of an object in the world coordinate system is computed. (b) The grasp information of an object is computed. A set of grasp information consists of the center point of the robot hand (GCP), the approaching direction of the robot hand (GAD), and the directions that the robot closes its hand (GCD).

pose of each object in the world coordinate system (we use tf package in ROS).

(D) After computing the bounding cuboid and its pose, PR generates candidates of the grasp information for each object by considering the approaching direction and the kinematics of the gripper used for manipulation. The grasp information is threefold: (i) Grasp Center Point (GCP), (ii) Gripper Approaching Directions (GADs), and (iii) Gripper Closing Directions (GCDs). These three elements specify a unique grasp. GCP means the center of a grasp shown as a point in Figure 5b. The robot uses this point to locate the center of its hand so GCP is determined by considering the size of the robot hand. GAD (the green arrow in Figure 5b) is a vector indicating the direction that the robot approaches to GCP. GAD is determined by the relative location of the robot to the object. There could be more than one GAD depending on the workspace of the robot. For each GAD, a set of vectors indicating the directions that the robot closes its hand to grasp the object. It is GCD shown as the red arrows in Figure 5b. If only GCP and GAD are given, the robot cannot determine the rotation of the hand around the GAD vector. In other words, the robot does not know how much to rotate its wrist to grasp the object. GCD is determined by considering the size of the bounding cuboid, the size of the robot hand, and the pose of the object. GCP, GAD, and GCD comprise a set of grasp information. If PR generates multiple sets of grasp information for a single object, other components using them select the most appropriate set based on available information (e.g., the size and stable grasp of the object). If no such information is available, one set could be chosen randomly.



(a) The left hierarchy shows classes and properties to represent the knowledge related to time. The class hierarchy in the middle is for the spatial information. The hierarchy shown in the right has classes and properties to represent the knowledge object poses mathematically.


red_can1
type: Bottle
width: 0.05
depth: 0.05
height: 0.12
visualObjectPerception_492w
type: VisualObjectPerception
objectActedOn : red_can1
eventOccursAt :rotationMatrix3D_38
startTime :timepoint_3928405921
startTime :timepoint_3928405923
startTime :timepoint_3928405925
1
2
3
Time

(b) At the top, the basic information of the canned coffee is shown. Every time PR perceives objects, obtained information is stored in the world model. The circled numbers indicate the order of acquired information over time.

FIGURE 6. An example of ontology and knowledge implemented in Knowledge Manager (KM). (a) The ontology has different class hierarchies for temporal, spatial, and mathematical information. (b) Pieces of knowledge of an object are generated using the ontology.

If the chosen grasp is turned out to be invalid, other sets can be used.

2) KNOWLEDGE MANAGEMENT

Knowledge Manager (KM) maintains high-level knowledge about the robot, goal, tasks, and the environment. As shown in Figure 2, KM includes an ontological knowledge base, rules for reasoning, and predicates for representing states. The world model contains low-level metric information about the environment from perceived information through Perception Handler. Model Query Processor answers queries from other components. Hybrid Knowledge Reasoner produces high-level knowledge by applying spatio-temporal reasoning rules to the metric information in the world model. KM is implemented in Prolog and Java and interfaced by rosjava package in order to communicate with other ROS nodes through topics and services.

As shown in Figure 3, KM receives various sensory information from WM through Perception Handler. Perception Handler has a synchronous interface (i.e., by request) and an asynchronous interface (i.e., by event) to deal with different types of sensory information. The synchronous interface is used to receive the information requiring considerable computing resources such as object and grasp information. If they are computed without a request, PR may consume the majority of computing resources so delay all the other jobs. The asynchronous interface is used to receive critical information related to safety and system operation such as emergency stop signals or battery levels.

The ontology consists of the conceptual and the relational layer. Basically, the ontology is represented by description logic (DL) such as Resource Description Framework (RDF) schema and Web Ontology Language (OWL). Prolog is used to represent axioms and inference rules that satisfy the relationships between objects and concepts. Figure 6a shows an example of an ontology for manipulation services. Note that this ontology shows important classes only owing to the space limit.

In the left (blue) of Figure 6a, the class hierarchy shows classes and properties to represent the knowledge related to time. Event class represents temporal events such as perceiving objects. TimeInterval class represents the duration of events where particular time points are described by TimePoint class. TemporalThing class at the top-level defines the properties for the start and end time, so lower-level classes can inherit such properties to represent temporal information of events. The class hierarchy in the middle (green) is for the spatial information. Subclasses defined to describe objects (e.g., CoffeeCan, CookieBox, and Shelf) that belong to SpatialThing class. spatiallyRelated property describes spatial relationships between instances of the class, with subproperties such as topological relations (e.g., on-Physical), directional relations (e.g., inFrontOf), and distance relations (e.g., near). Also, the class could have objectActedOn property to represent an event about objects. For example, objectActedOn can describe that VisualObjectPerception occurs to CoffeeCan if a coffee can is visually perceived by the robot. In the right (pink), the hierarchy has classes and properties to represent the knowledge about mathematical information. MathmaticalThing class is used to represent data structures about perceived objects. RotationMatrix3D can be used to represent a 3D pose of an object described by eventOccursAt property.

Figure 6b shows an example of the knowledge about an object represented by the ontology. At the top, the basic information of the canned coffee known as a priori is shown, such as the class type and the dimension. Every time PR perceives the object, Perception Handler stores the object information in the world model in the form of ontology. The circled numbers indicate the order of acquired information over time.

Model Query Processor handles queries from other components [58]. For example, TMI sends queries to KM about the environment (e.g., poses of static obstacles), object properties (e.g., sizes, weight), and spatial relationships (e.g., objects occluding a target object). Model Query Processor finds low-level knowledge from the world model to answer the queries. If answering a query requires reasoning for higher-level knowledge, Hybrid Knowledge Reasoner answers it by applying inference rules about spatio-temporal relationships. If the world model does not contain knowledge to answer some queries, it provides answers once the necessary knowledge is registered in the world model.

Figure 7 shows an example of a query and an answer. TM sends a query to KM to know the object(s) on top of the shelf through a ROS message. The message consists of a predicate on-Physical and parameters Top and shelf03. Model Query Processor translates the message to a Prolog query on-Physical(Top, 'shelf03') and sends it to Hybrid Knowledge Reasoner. Hybrid Knowledge Reasoner returns an answer Top = coffeeCan03, Top = cookieBox02. The answer is formatted in a ROS message by Model Query Processor and then sent to TM.

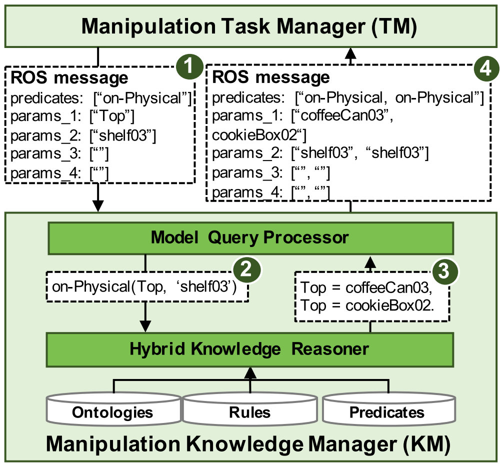

Manipulation Task Manager (TM)
1
ROS message
predicates: ["on-Physical"]
params_1: ["Top"]
params_2: ["shelf03"]
params_3: [""]
params_4: [""]
4
ROS message
predicates: ["on-Physical, on-Physical"]
params_1: ["coffeeCan03", cookieBox02"]
params_2: ["shelf03", "shelf03"]
params_3: ["", ""]
params_4: ["", ""]
Model Query Processor
2
on-Physical(Top, 'shelf03')
3
Top = coffeeCan03,
Top = cookieBox02.
Hybrid Knowledge Reasoner
Ontologies
Rules
Predicates
Manipulation Knowledge Manager (KM)

FIGURE 7. An example reasoning about spatial relationships of objects. The circled numbers indicate the order of the reasoning process. 1) A query asking about objects on a shelf is sent to KM via a ROS message. 2) Model Query Processor translates it to a Prolog query. 3) Hybrid Knowledge Reasoner returns an answer. 4) The answer is returned to TM via a ROS message.

3) SYMBOLIC TASK PLANNING

Task Manager (TM) manages goals to be achieved and generates symbolic task plans (in the form of logical predicates) to achieve the goals. TM works reactively to the changes such as new or modified goals. Such changes result in recomputation of the task plans. As shown in Figure 2, TM consists of the goals, plan library, and task state. A goal can be given or generated automatically via a cyclic observation. TM sends queries to KM to know the context in each cycle and posts goals dictated by the context. For example, an object organizing service has a rule that a shelf should be replenished if some items are missing. The rule is saved in KM as a predicate. If a particular item is not in the shelf, TM posts a goal taking the item from a storage and placing it to the shelf. The plan library is prepared upfront by considering the components and capabilities of the robot. Task states are represented by logical predicates. In order to maintain recency of the task state, TM requests state information to KM.

Task plans are generated and executed by Task Planner and Task Executive based on the information given by the goals, plan library, and task state. Task Planner generates multiple plans in an online manner for multiple goals simultaneously. Among the plans for multiple goals, Task Executive selects those plans that should be executed immediately according to the schedules determined by the planner. Technically, Task Planner and Task Executive are implemented by JAM, a BDI-theoretic agent architecture [59].

Task Executive generates subgoals while executing a plan through communication with KM. Then Task Planner generates task plans for the subgoals. During execution, some tasks may require motions of the robot. If some of the motions are not feasible (e.g., owing to a kinematic singularity), the whole plan including such infeasible motions cannot achieve the goal. Thus, TM needs to check the feasibility of motions before sending the plan out to the robot. In our architecture, TMI is in charge of checking feasibility of motions through interactions with AM. Figure 3 shows the data flow between TM and TMI for feasibility checking. TM asks feasibility of robot motions, which are necessary to execute tasks, to TMI via the behavior/action interface. If a plan is not executable owing to some infeasible motions, TM replans and repeats the same process until it establishes an executable plan. Even though a plan is verified to be executable, the robot might fail to execute the plan owing to contingencies such as robot malfunctions or collisions incurred by dynamic obstacles. In this case, TM also replans to establish a new plan based on the updated task states.

On the other hand, the task plans generated by TM are represented by predicates and symbolic variables. The predicates represent actions of the robot while symbols represent objects or locations. For example, PickObject(*box*, *table*) means picking a box on a table. Although this representation can express rich information about the goal and context, it is not directly understandable by the robot. Rather, the robot needs an exact goal pose of its hand to grasp the box and geometric representation of obstacles in the workspace. In the following section, we describe how TMI bridges between symbolic task plans and geometric motion plans.

TM can perform offline and online planning. Offline planning establishes a full set of task plans achieving the goal before execution. Then the plans are sent to TMI to be executed by the robot. If one of the task plan fails during execution, TM replans to modify the plan. Online task planning generates each task plan on the fly and sent it to TMI immediately. Depending on the execution result returned by TMI, TM generates the next task.

4) INTERFACING BETWEEN SYMBOLIC TASKS AND GEOMETRIC MOTIONS

We develop Task-Motion Interface (TMI), an interface between a task planner and a motion planner to deal with the difference between the information that the two planners handle. The functions of TMI in the architecture are: (A) splitting abstract tasks into atomic subtasks, (B) generating goal poses for tasks, (C) generating intermediate poses between the poses if necessary, and (D) dealing with planning failures. As described in Section III-B3, TM can generate a full set of task plans offline or a single plan in an online manner. Regardless of the planning mode, the input, output, and the processes of TMI are the same. We describe the processes with a running example of picking an object as shown in Figure 8. Notice that TMI is written in Python.

(A) We use the interface to resolve the problem caused by the inability of motion planners in dealing with abstract tasks. Motion planners generate motions according to a query specifying a goal pose so cannot handle abstract tasks directly without decomposing them into a sequence of subtasks. For example, a task PickObject(*box*, *table*) is composed of subtasks such as moving the end-effector to the box from a ready pose, grasping the box, and then returning back to the ready pose. A motion planner should receive a query for each subtask so the decomposition is necessary.

Therefore, TMI decomposes abstract task plans. TM also can decompose task plans but increasing the number of atomic tasks reduces the efficiency of symbolic task planning significantly.[1] In our manipulation services, tasks involving motions can be done by a mix of three primitive actions that are picking, placing, and base moving. TMI decomposes manipulation tasks into atomic arm and gripper actions as shown in Figure 8. TMI breaks navigation tasks into linear and angular actions. The atomic actions are stored in the action library in TMI. In the current implementation, the decomposition is hard-coded as there are only three primitive actions. If the architecture is with a large number of primitive actions, KM could store the rule for decomposition and tell TMI how to decompose.

(B) We implement a method to deal with the difference between the information handled by the task planner and the motion planner. Motion planners generate trajectories of joints in order to accomplish tasks such as grasping or releasing objects and navigating to a distant waypoint while avoiding collisions. Thus, the input to a motion planner is a goal pose of the robot end-effector or the mobile base. However, symbolic task planners are not able to specify such poses since they cannot reason about geometric relationships between robot kinematics and obstacles.

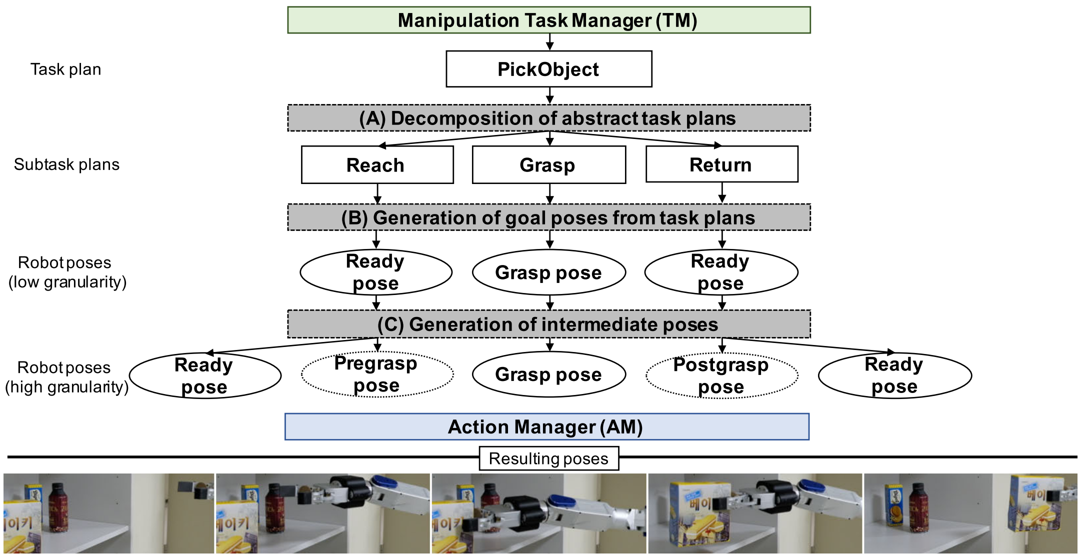
Thus, TMI provides goal poses of the end-effector and the mobile base that are missing in the task plans. For the goal pose of the end-effector (described by $[x, y, z, \phi, \theta, \psi]$), TMI asks WM for the grasp information of objects if the task plan is for picking. If the plan is for releasing, TMI asks KM for the geometric information of empty slots. For the goal pose of the mobile base ($[x, y, \theta]$), TMI asks KM about the location of the target object or an empty slot to move close to either of them. Since the object or the slot would not be in the current sensing range of the robot, relevant information may not reside in WM. Thus, KM answers the query about the location of the object or the slot. Then TMI determines an appropriate base pose where the robot can detect and reach the object or the slot. Once the robot achieves the goal base pose, WM can provide the grasp information of the target or the geometric information of the slot with neighboring objects. Based on the information from WM, appropriate goal poses of the end-effector are computed. Figure 8b shows different grasp poses. Release poses for ReleaseObject plan also can be computed autonomously. Once goal poses are computed, TMI sends them to the motion planner. The geometric information of movable objects and static obstacles are sent together for collision checking.

(C) We implement an additional decomposition method in TMI to increase the computational efficiency of motion planning. The level of granularity of the task plans could be still high in order to compute trajectories quickly. For example, the computation of the trajectory from the ready pose to a grasp pose includes frequent collision checking, which demands high computational costs, around the target object and other nearby obstacles. The same applies to release poses. Motion planners may not be able to return an answer to a planning query within a cutoff time. Thus, TMI refines the primitive actions by breaking them into granular actions for faster computation of trajectories. Approaching a target object or an empty slot can be split into navigating in the obstacle-free space quickly and approaching cautiously to the target while avoiding collisions in clutter.

Figure 8a shows the processes (A), (B), and (C) for PickObject plan. From TM, TMI receives a predicate representing the plan where its symbols are instantiated with *box* and *table*. Then (A) TMI decomposes the task plan to subtasks of reaching the object, grasping the object, and returning back to the ready pose. Next, (B) TMI generates a goal pose of the robot for each subtask by aggregating information such as the pose of the box and the spatial relationship between the box and other obstacles on the table. Finally, (C) intermediate poses (pregrasp and postgrasp) are generated to increase computational efficiency of motion planning. At the bottom of Figure 8, the sequence of the robot poses are shown.

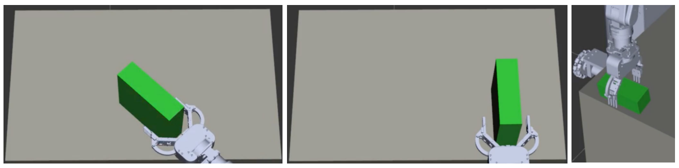
(D) Another important function of TMI is dealing with task and motion plan failures as shown in Figure 8c. TMI (i) adjusts goal poses if motion planning fails and (ii) checks whether the motion planning or execution of a task plan

[1]Classical planning is in **PSPACE** and temporal planning is in **EXPSPACE** [60]

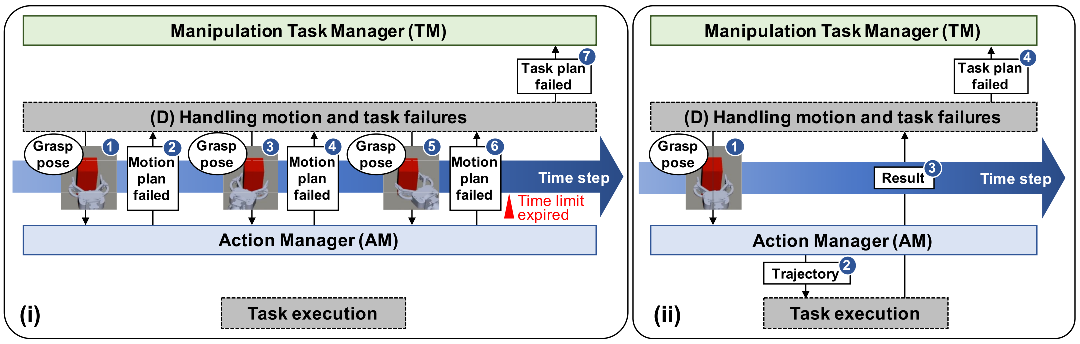


Manipulation Task Manager (TM)
Task plan
PickObject
(A) Decomposition of abstract task plans
Subtask plans
Reach
Grasp
Return
(B) Generation of goal poses from task plans
Robot poses (low granularity)
Ready pose
Grasp pose
Ready pose
(C) Generation of intermediate poses
Robot poses (high granularity)
Ready pose
Pregrasp pose
Grasp pose
Postgrasp pose
Ready pose
Action Manager (AM)
Resulting poses

(a) Functions (A–C) for interfacing between TM and AM. (A) An abstract task plan from TM is decomposed into subtasks. (B) Goal poses are generated for each subtask. (C) Intermediate poses are generated if necessary.



(b) Different grasp poses determined based on the poses of the object and the table



(c) Function (D) for handling failures. The circled numbers indicate the chronological order of the process. (i) TMI adjusts poses until motion planning succeeds. If a time limit expires, it reports a failure of the task plan to TM with a possible cause of failure. (ii) TMI reports a failure of a task plan with a possible cause its execution fails.

FIGURE 8. Functions of Task-Motion Interface (TMI) for PickObject plan. Other plans go through the same process. (a) TMI provides goal poses of the robot to AM from a task plan given by TM. At the bottom, the resulting poses for ready, pregrasp, grasp, postgrasp, and ready are shown. (b) TMI generates poses for subtasks. (c) TMI handles failures of task and motion plans.

is successful. If not successful, TMI reasons about the situation. We explain how the adjustment and reasoning is done. First, TMI adjusts the pose if motion planning fails and then sends the new pose for another trial of motion planning. TMI repeats the adjustment until motion planning succeeds. If a predefined time limit expires before a success, the adjustment stops and TMI reports the failure to TM. The implementation for the adjustment can vary depending on the application domain, the environment, robot hardware or etc. In our implementation, the adjustment is done by varying the goal orientation of the end-effector while fixing the goal position. The variation of orientation is done by considering the robot kinematics and the shape of the object to be grasped or released. In most cases in our experiments, rotating around the z-axis perpendicular to the x-y plane (i.e., varying the yaw angle) is successful.

Second, TMI reasons about failures to provide feedback to the task planner. A task plan can fail during either motion planning time or execution time. If a failure occurs during motion planning, it means that some goal poses cannot be achieved. Few possible causes can be reasoned by checking the distance between the robot and the goal pose or finding obstacles around the goal. If the distance is longer than the working range of the robot, the goal pose is out of the robot workspace so TM may insert a base moving action. If there are some objects identified between the goal and the robot, they would block the goal so TM may command to remove the obstacles. On the other hand, there exist many possible implementations to detect failures at execution time. In our implementation, TMI collects the result of motion planning and the values of robot joints. If motion planning for all poses for a task is successful but the object is not grasped or released (known by reading gripper's joint value), grasp information could be inaccurate or the empty slot could be found wrongly. With the information, TM may generate a plan that perceives the environment to recompute the grasp information or the empty slot.

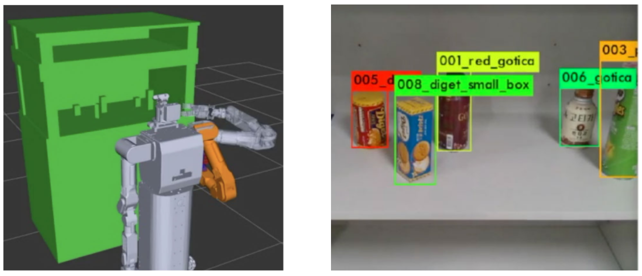
5) GEOMETRIC MOTION PLANNING

Action Manager (AM) interfaces between the architecture and the robot hardware. AM generates joint trajectories for the goal poses and sends them to the robot controller. AM handles motion planning of both manipulation and navigation.

For manipulation, we use randomized motion planners which can work in an anytime fashion as the completion of services is more important than optimality in our manipulation services. Among many possible implementations of the randomized motion planner, we use the Open Motion Planning Library (OMPL) [61] that provides many variants of Rapidly-exploring Random Tree (RRT) and Probabilistic Roadmap (PRM). The library is available in MoveIt motion planning framework [8] running on top of ROS. Since our goal is to develop an architecture that does not depend on particular hardware platforms, MoveIt is appropriate as it only needs a model of the robot used (i.e., URDF).

The input to the motion planner is the goal pose of the end-effector, the poses of movable objects, and the poses of static obstacles within the workspace of the robot which are all in the robot's coordinate system. MoveIt constructs a virtual planning scene as shown in Figure 9a. The objects in the scene are shown in Figure 9b. The output of the motion planner is a trajectory of joints of the manipulator. AM sends the output to the robot and then the controller in the robot will control the actuators to follow the trajectory.

For navigation, we use global and local navigation modes [62], [63] that are in the world and robot's coordinate systems, respectively. For global navigation, we use a 2D simultaneous localization and mapping (SLAM) algorithm to localize the robot and detect obstacles in the environment. We use a dynamic path planning algorithm based on A* search. This global mode is used to move between places where the robot performs manipulation tasks. However, localization could be inaccurate in feature-poor environments. Localization errors may prevent the robot from getting close enough to the objects. Thus, we use the local mode for



(a) A motion planning scene **(b)** The view from the robot

FIGURE 9. Motion planning done by Action Manager (AM). (a) A motion planning scene is constructed from the pose information about the objects and obstacles in the workspace of the robot. (b) Detected objects in the view of the robot.

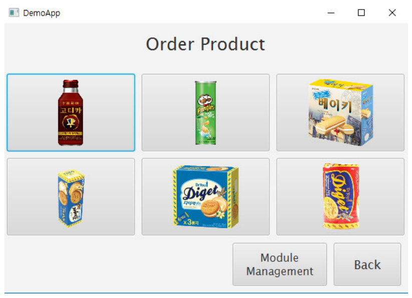


FIGURE 10. The screenshot of the GUI for a grocery fulfillment service.

fine-tuning of the base pose once the robot arrives the place to perform a manipulation task. The local navigation receives a relative goal pose (i.e., $[\Delta x, \Delta y, \Delta\theta]$) from the current base pose. The local mode does not consider collision avoidance but generates a linear path to the goal. If the robot arrives at a place using the global navigation but the robot cannot reach the objects, the robot could utilize the local mode to adjust the base pose slightly to have the objects within a reachable area.

6) USER INTERFACES

We implement the front-end of the architecture with a graphic user interface (GUI) and a text-to-speech (TTS) tool to provide services to humans.[2] An example of the GUI is shown in Figure 10 assuming a service bringing objects to users. The GUI is connected wirelessly to the barebone computer running the architecture. We set up a peer-to-peer wireless network for the communication. A recipient of the service selects an object in the GUI. The GUI sends the user input to TM through the network so a goal is generated. The GUI can run in a tablet PC or a desktop PC. If necessary, the GUI also shows the information about collected knowledge and task plans in another window. The TTS tool tells the user about the action that the robot is going to perform. We annotate each action predicate and let the TTS tool read the annotation before a predicate is executed by the robot.

[2]We do not implement natural language processing to interpret human verbal descriptions of tasks.

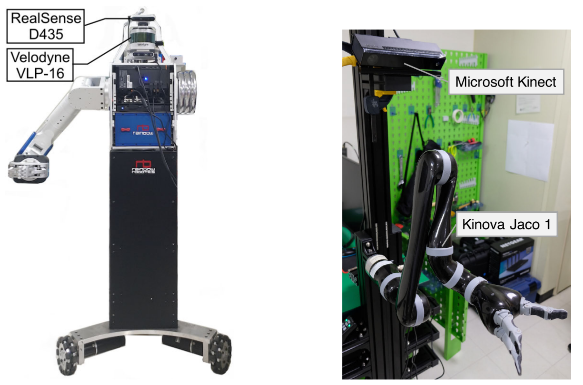
IV. EXPERIMENTS AND DEMONSTRATIONS

In this section, we describe the robots and environments used in experiments. Then we show how lab tests and demonstrations are done. In the early stage of development, we used a high fidelity dynamic simulator V-REP [64] with Vortex physics engine. The simulator was useful to check if the components are working correctly with robots. However, we had to stop using it owing the gap between simulated environments and reality. For example, sensing and wireless communication could not be simulated perfectly as models of noise are different from the reality. Also, the simulated lights and textures were not exactly the same with real ones. Thus, deep neural networks trained with real object images do not perform correctly with simulated objects. In addition, simulations of physics on objects were sometimes unrealistic. Thus, we moved on to real robots and environments to test the architecture.

A. PHYSICAL ROBOTS

We use Mobile Hubo (M-Hubo) [62], a wheeled humanoid with three omnidirectional wheels (Figure 11a) to test the entire architecture. The robot is under development so we are able to use the right arm only, which is with 7 DOFs. The robot has two barebone computers where one is used for the software architecture (Intel i7, 16G RAM, GTX960) and the other is in charge of hardware control (Intel i7, 8GB RAM, an integrated graphics unit GT4e). The two computers communicate over a wire. The low-level real-time control of M-Hubo is done through PODO framework [65]. M-Hubo is with an interface [63] between PODO and ROS. The output from our architecture, which is a trajectory formatted as a ROS message trajectory_msgs, is translated to real-time control input. A LIDAR (Velodyne Puck VLP-16) and an RGB-D camera (Intel RealSense D435) are mounted in the head. The LIDAR is used to capture the overall and long-range surrounding environment. The camera is tilted down to obtain visual and depth information from the region of interest of the robot.[3]

[3]More information about M-Hubo can be found in [62].



(a) M-Hubo, a wheeled humanoid **(b)** Jaco 1 anchored at a post

FIGURE 11. **The robots used to test the proposed software architecture.**

Also, we use another robot to show compatibility and modularity of our architecture that can work with different hardware and can be partially deployed. We combine PR, TMI, and AM with a task planner which determines what to remove in what order [66]. For this test, we use a 6-DOF manipulator Kinova Jaco 1 anchored at a fixed base (Figure 11b). An RGB-D camera (Kinect V2) is installed above the manipulator where the whole workspace of the robot can be captured. A desktop computer running the components is with Intel i7, 16G RAM, and GTX1060. For the change of the robot, we only have to modify AM to replace the URDF robot model and the ID of the robot. The format of the output from the task planner does not change.

For motion planning of the both robots, we use RRTConnect [67] which shows the best performance (in terms of computation time) in our pilot studies. BiTRRT [68] shows similar performance as they are all bidirectional planners.[4] We impose a time limit of 0.5 second for each query of motion planning. If a trajectory is not found until the limit expires, AM returns a failure so TMI modifies the goal pose until motion planning succeeds or the number of pose modification exceeds a predefined limit.

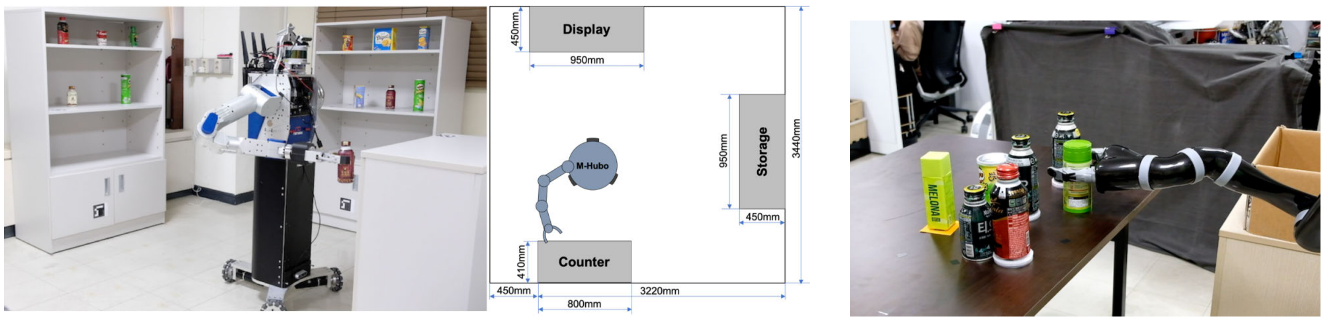
B. ENVIRONMENTS

We develop the proposed software architecture to provide manipulation services where the service domain is not tied to a particular environment. The architecture can be used in any environment where the robot can recognize movable objects and static obstacles, the robot has an enough configuration space to perform tasks, and no human exists within the workspace of the robot. The architecture does not have a measure for securing physical safety of nearby humans. Thus, using it with robots that are closely interacting with humans is our future work, which is detailed in Section V.

For testing of our architecture, we choose human environments that have shelves and tables. M-Hubo performs fulfillment tasks in a small retail store setting as shown in Figure 12a. M-Hubo receives an order from a customer through a graphic interface, moves to a display from a counter, picks the ordered item, returns to the counter, and places the item on the counter. If the ordered item is not found in the display, M-Hubo finds it in the storage. If the item is not in the storage as well, the robot returns to the counter with empty hands. M-Hubo is also tasked with replenishing jobs when the robot is not serving an order. If the display does not have a particular item but the storage does, the robot relocates the item from the storage to the display.

The environment in the M-Hubo experiment seems similar to those in bin picking problems. Objects are irregularly placed in the shelves. Objects could occlude each other. Empty spaces to place objects should be recognized on the fly. No information about orders is known beforehand. However,

[4] [69] show a useful comparative study about available motion planners in OMPL. We also compare the two fastest planners found in [69] where motion planning for a picking motion takes 2.93 sec ($\sigma = 1.10$) with RRTConnect and 3.25 sec ($\sigma = 1.04$) with BiTRRT over 10 repetitions.



(**a**) M-Hubo performing tasks while moving between the places

(**b**) Jaco performing tasks at a fixed location

FIGURE 12. The environments where tests were done. (a) M-Hubo navigates between different places (counter, display, and storage) to perform manipulation tasks such as fetching an ordered item from the display and placing it on the counter. (b) Jaco 1 performs manipulation tasks such as relocating some obstacles to retrieve a target object while fixing its base at a post.

the hardware used in our experiments is not specialized for the industrial fulfillment task as bin-picking robots are often with a suction cup that is more versatile so can manipulate objects more easily than grippers and fingers. If we have a manipulator equipped with a suction cup, we believe that our architecture can work in industrial setting as well.

We set up another environment for Jaco 1 to focus more on object manipulation itself but not navigation. Jaco 1 simply performs pick-and-place actions in a tabletop setting shown in Figure 12b. Jaco 1 removes objects occluding a target object until the target can be retrieved from clutter. A basket is prepared next to the robot to place the removed objects.

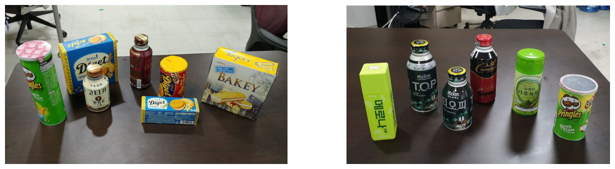
In both environments, we assume that overhand grasps are not allowed so the robot should approach objects from the side. Task and motion planning becomes much simpler if the target object can be grasped from the top without removing objects occluding the target. Also, motion planning for overhand grasps is less difficult as the robot end-effector does not need to navigate among objects. Thus, this assumption does not make the manipulation problem easier.

C. FUNCTIONAL TESTS

In this section, we describe test results of the components in the architecture. We also show details of the implementation used in experiments. Some components cannot be tested independently. For example, synthesizing grasp information and generating goal poses of the end-effector are a means of object manipulation but not an end in itself. Thus, some test results are qualitative and illustrative rather than quantitative.

1) PERCEPTION

For object detection, PR uses Yolo-v3 network [38] for M-Hubo and Faster R-CNN [36] for Jaco 1. They both generate the output in the same format that is bounding boxes of detected objects in 2D images. We use seven objects shown in Figure 13a for the M-Hubo dataset and six objects shown in Figure 13b for the Jaco 1 dataset. For the M-Hubo dataset, we collect around 100 images per object and 300 images of grouped objects. In the images of grouped objects, some objects occlude each other. For the Jaco 1 dataset, we collect



(**a**) The objects used in M-Hubo experiments

(**b**) The objects used in Jaco 1 experiments

FIGURE 13. Objects for experiments. Graspable objects by M-Hubo and Jaco 1 are chosen.

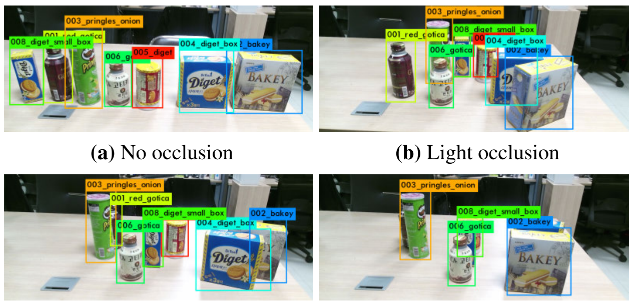
300 images per object. We augment the datasets by changing the scale, rotation, translation, and HSV (hue, saturation, and value) color. We change the scale within the range of 50–110%, the rotation between -30 and 30 degrees, and the translation between -10–10 pixels. In the HSV color space, the variation ranges -5–5, 5–30, and -10–20 for hue, saturation, and value, respectively. In the training of Yolo-v3, the learning late begins from 5×10^{-3} and gradually decreases to 5×10^{-4} and 5×10^{-5}. The weight decay, the momentum, and the mini-batch size are 5×10^{-3} and, 0.9, and 32, respectively. Faster R-CNN network is trained with 100k steps and batch size of 1.

Grasp synthesis is done in the same way for both datasets. The background is far from objects so we limit the depth range in the point clouds to remove the background points. To remove points from the supporting plane of the objects (e.g., tabletop), we extract a dominant plane from point clouds and remove the points that belong to the plane. However, plane extraction is costly since it should be done for every frame of the input stream. Since we use the architecture online, we find a plane by using a synthetic marker (ArUco marker [70]) for a faster background removal. After removing the points from the background and the tabletop, PR computes grasp information as described in Section III-B1.

We test the detection method with the seven objects used in the M-Hubo experiment. Notice that the test results with Faster R-CNN are not included as they show similar results. We show four different arrangements with varying degrees of occlusion in Figure 14. The accuracies of the

TABLE 2. Average query processing time of KM with standard deviations.

Predicate	type	empty-hand	open-hand	close-hand	on-Physical	graspedBy	Total
Mean query processing time (ms)	12.0	39.3	32.8	34.0	15.9	57.0	191.0
Std. dev.	1.82	7.46	8.58	3.46	5.27	24.89	5.09



(a) No occlusion **(b)** Light occlusion

(c) Light occlusion **(d)** Severe occlusion

FIGURE 14. Detection results with seven objects. The accuracies of detection (# correctly detected objects / # all objects) are (a) 100%, (b) 100%, (c) 100%, and (d) 57.1%. Confidence scores of objects range between (a) 96–99%, (b) 93–99%, (c) 81–99%, and (d) 99%.

detection (# correctly detected objects / # all objects) with the arrangements are 100%, 100%, 100%, and 57.1% in Figure 14a–14d, respectively. Confidence scores of each object range between 96–99%, 93–99%, 81–99%, and 99% for Figure 14a–14d, respectively. In other tens of test arrangements with minor occlusion, the detection accuracy achieves almost 100% where confidence scores range between 80–99%. With severe occlusion (like Figure 14d where some objects are occluded significantly), the detection accuracy drops to 50–80%. However, the robot does not grasp occluded objects directly without relocating some front objects. Thus, most of the objects that the robot grasps are correctly detected with high confidence scores.

The result of grasp synthesis seems correct as GCPs are inside objects, GADs point toward objects from robot's position, and the robot can close its gripper from where GCDs direct. However, testing grasp synthesis is not easy as there is no ground truth for grasp information. Thus, grasp synthesis is evaluated together with other components in comprehensive tests in Section IV-D2.

2) KNOWLEDGE MANAGEMENT

As described in Section III-B2, KM collects perception instances of objects and the robot and infers high-level predicates. We show how KM collects knowledge from perception with a test case in the environment shown in Figure 12a. The collected knowledge and their relationship can be represented by a knowledge graph as shown in Figure 15. Technically the graph visualization is done by using a visualization library vis.js.

Initially, the knowledge base does not contain the perception instances of the coffee can. Thus, only hubo_hand and displayMiddleShelf, which represent the hand of M-Hubo and the middle shelf of the display, exist in the graph. In Figure 15a, M-Hubo detects the coffee can so the perception instance coffee_can is added to the world model of KM. Through geometric reasoning using perception instances and the semantic map, KM finds that the can is currently on the middle shelf of the display. Thus, coffee_can is linked to displayMiddleShelf with a directed edge on-Physical property. After M-Hubo grasps the can (Figure 15b), a directed edge graspedBy from coffee_can to hubo_hand is added. Since the object is no longer on the shelf (Figure 15c), the directed edge on-Physical is deleted.

The query processing time measures the performance of the reasoning function of KM. We measure the processing time where the world model of KM has 2,886 semantic triples[5] (1,006 triples for definition of class hierarchy and property, 1,880 triples for perception instances). The result is shown in Figure 16 and Table 2. We test six different types of queries that are type, empty-hand, open-hand, close-hand, on-Physical, and graspedBy. The total in the right of the graph means that we query all the six predicates at once. All parameters in the predicates are variable. The processing time for type is the shortest because it requires pattern matching only. Other predicates take longer processing time because they require geometric computations. Especially, predicates related to the robot (e.g., graspedBy) are slower as they require geometric computations incorporating both objects and robot joints. The mean processing time is less than 0.2 second even for the query for the full state with six predicates. It shows that the query processing speed of our KM is sufficiently fast for real-time applications.

3) TASK MANAGEMENT

Similar to grasp synthesis, measuring the performance of the task planner independently is not straightforward because a success in task planning does not guarantee a success in manipulation tasks by the robot. Thus, we show how task planning and replanning are done to achieve a manipulation goal with a test case.

In Figure 17, we show two sequences of action predicates generated to achieve a goal fulfilling an item (a coffee can) for a customer in the environment shown in Figure 12a. Figure 17a shows a sequence if the task plan is executed without any robot failure. In the beginning, TM sends a query to KM to know the location where the coffee can is stored. After reasoning, KM answers that the can is in the display. TM generates MoveBase to move the robot to the display. Once the robot finishes moving, it needs to locate the target object so

[5] A triple is the atomic entity in the RDF data model that consists of subject, predicate, and object expressions.

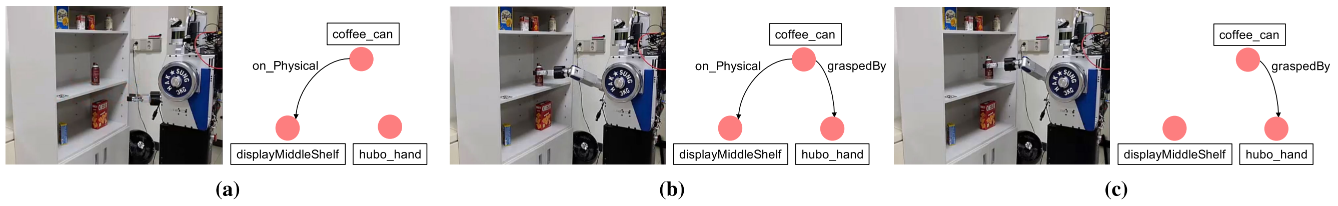


FIGURE 15. An example of showing how the knowledge changes while M-Hubo performs tasks. (a) coffee_can is linked to displayMiddleShelf with a directed edge on-Physical property. (b) After M-Hubo grasps the can, a directed edge graspedBy from coffee_can to hubo_hand is added. (c) The can is no longer on the shelf so the directed edge on-Physical is deleted.

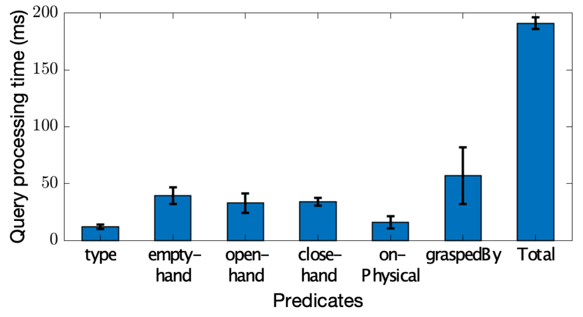


FIGURE 16. Query processing time of KM. We test six different types of queries that are type, empty-hand, open-hand, close-hand, on-Physical, and graspedBy. The total in the right of the graph means that we query all the six predicates at once.

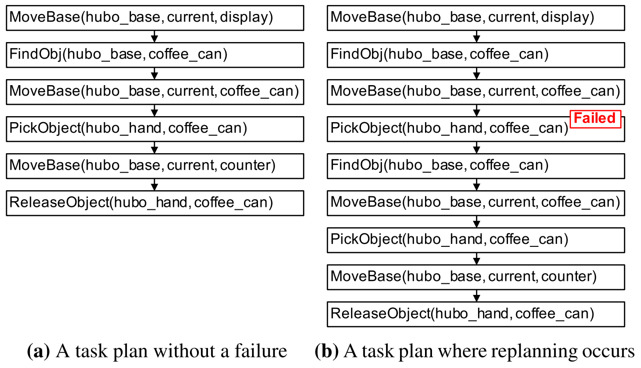

MoveBase(hubo_base, current, display)
FindObj(hubo_base, coffee_can)
MoveBase(hubo_base, current, coffee_can)
PickObject(hubo_hand, coffee_can)
MoveBase(hubo_base, current, counter)
ReleaseObject(hubo_hand, coffee_can)
MoveBase(hubo_base, current, display)
FindObj(hubo_base, coffee_can)
MoveBase(hubo_base, current, coffee_can)
PickObject(hubo_hand, coffee_can)
Failed
FindObj(hubo_base, coffee_can)
MoveBase(hubo_base, current, coffee_can)
PickObject(hubo_hand, coffee_can)
MoveBase(hubo_base, current, counter)
ReleaseObject(hubo_hand, coffee_can)
(a) A task plan without a failure
(b) A task plan where replanning occurs

FIGURE 17. Sequences of action predicates generated by TM. The task plans achieve a goal fulfilling an item (a coffee can) for a customer in the environment shown in Figure 12a. (a) The robot moves to the display where the target object is located, finds the target, move closer to the target, pick the target, moves to the counter, and release the target on the counter. (b) When the robot fails to pick the target, TM replans to update the target pose before the robot tries again to pick the target.

FindObj is generated. After the robot estimates the pose of the target, the robot moves its base close to the target. Then TM generates PickObject, MoveBase, and ReleaseObject in a row to pick the target, move to the counter where the customer waits, and release the target on the counter. On the other hand, replanning occurs in Figure 17b when the robot fails to pick the target. TM let the robot locate the target again through FindObj and change the base pose for the updated target pose.

4) MANIPULATION

We test the functions of TMI and AM for manipulation that are generating goal poses of the end-effector and collision-free motions, respectively. Given grasp information of objects and the environment, we run TMI and AM to grasp an object. We test two different types of objects. First, the robot tries to grasp objects that can be grasped from any direction. Second, the robot tries to grasp an object that has limited reachable directions. The tests are done with M-Hubo in MoveIt motion planning scenes. The base pose is determined such that the end-effector is in the middle of the shelf. The base pose does not change during the tests.

If motion planning is not successful, TMI modifies the pose, and AM finds motions for the modified pose. The pose modification is done by changing the approaching angle of the end-effector. Specifically, the yaw angle (rotating around the z-axis) of the pose changes from the initial grasp orientation. For the target object that can be grasped from any direction, the initial orientation is set to zero so the end-effector approaches from the front. For the target with limited reachable directions, the center of the reachable directions is the initial orientation. If AM fails to find motions, following trials are with ± 8 degrees from the previous orientation. We allow TMI and AM to have at most 11 trials with different approaching angle (i.e., ranges between -40–40 degrees). If all trials fail, the manipulation task (i.e., PickObject) fails.

We have two sets of tests for the objects without and with limited reachable directions, respectively. These tests are to see the performance of TMI and AM for manipulation in clutter in which the robot should avoid collisions with nearby objects. For Set 1, we arrange eight objects uniformly at random in a confined space as shown in Figure 18a. The robot tries to grasp the four front objects. For Set 2, we arrange the target object at the center with three different orientations (-40, 0, and 40 degrees) where other objects are randomly

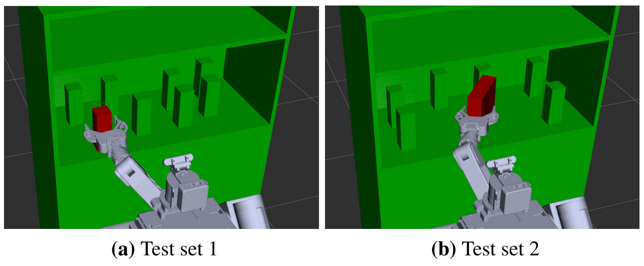


FIGURE 18. Screenshots of manipulation tests to measure the performance of TMI and AM on generating grasp poses and collision-free motions. (a) Set 1 is with objects that can be grasped from any direction. (b) Set 2 is with objects that have limited reachable directions. The target object (red) is rotated -40 degrees.

TABLE 3. The result from testing the functions of TMI and AM for manipulation. The success rate of generating valid grasp poses and motions, the average numbers of trials until success, and the average computation time for motion planning are shown. Standard deviations are shown in the parentheses. The statistics are from 50 random instances in each case.

Object	Set 1 (no limited reachable direction)				Set 2 (limited reachable directions)		
	Location 1	Location 2	Location 3	Location 4	Orientation 1	Orientation 2	Orientation 3
Success rate (%)	100	100	100	100	82	100	100
No. of trials	2.48 (1.66)	1.02 (0.14)	1.10 (0.36)	1.06 (0.24)	4.60 (3.08)	1.08 (0.27)	1.04 (0.20)
Planning time (sec)	3.64 (2.46)	1.46 (0.27)	1.52 (0.66)	1.31 (0.51)	6.78 (4.53)	1.56 (0.47)	1.34 (0.35)

located as shown in Figure 18b. We test total 350 random instances across across both sets. We measure the success rate of generating valid grasp poses and motions. We also average the number of trials and the motion planning time until success.

The result is summarized in Table 3. In Set 1, Locations 1–4 indicate the locations of the objects in the front line from the left to the right, respectively. For example, the object at Location 1 is shown in red in Figure 18a. In Set 2, Orientations 1–3 indicate the orientations of the target object (shown in red in Figure 18b) from −40, 0 and 40 degrees, respectively. Except for Orientation 1, AM succeeds to generate collision-free motions to grasp the objects in all test instances, which also indicates that TMI generates valid grasp poses. Since the robot uses the right arm only, the robot does not have a large configuration space to grasp the object rotated left. Thus, the success rate is 82%, and the number of trials and planning time are greater than other orientations. Orientation 3 with the object rotated right is easier to grasp since the robot can easily achieve the goal orientation with the right arm. Similarly, the number of trials and the motion planning time for the object in the left (Location 1) are greater than others as the object is located far from the right arm.

5) NAVIGATION

The global navigation mode of AM is implemented using move_base [71] and cartographer [72] packages in ROS. A 2D map is generated and the robot is localized in the map by cartographer package. The goal pose of the robot base in the world coordinate system is determined by TMI such that the robot can observe the whole space that objects are located. However, the localization error is up to 0.2 m in both x- and y-axis and 0.1 radian in the environment shown in Figure 12a. The error seems to occur owing to unsuccessful scan matching in the indoor environment with similar features. Due to the localization error, sometimes the objects go beyond the field of view of the robot even after the robot arrives at the goal location. Even though the robot is fortunate to see all objects, the objects could be out of the workspace of the manipulator.

In order to resolve this problem, we use the local navigation mode of AM. We implement the local mode using ArUco markers. We add a camera in the lower-body of M-Hubo and attach the ArUco markers on the sides of the shelves and the counter. Once M-Hubo detects a marker, it moves to the predefined location where the robot can reach objects as many as possible. This additional localization using the markers makes the base pose errors negligible. The motion planner uses poses of objects in robot's coordinate system so the base pose error does not affect motion planning once the objects are located within the workspace of the robot.

D. A PUBLIC DEMONSTRATION

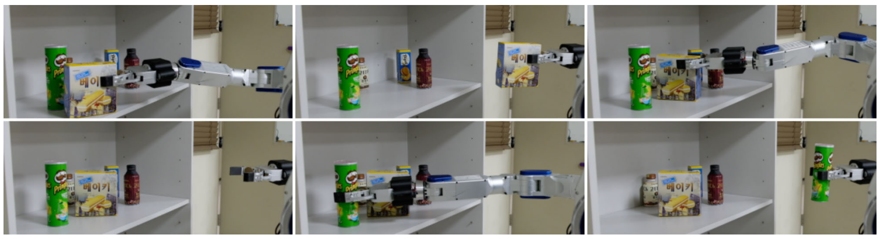
We had a public demonstration that M-Hubo performs a grocery fulfillment scenario in the environment shown in Figure 12a. Before the demonstration, we had several comprehensive tests in the lab. We describe the lab tests in detail and then sketch how the public demonstration was done outside the lab.

1) COMPREHENSIVE TESTS

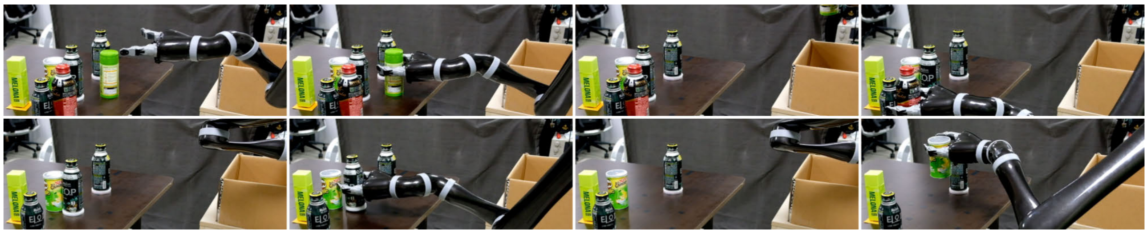
In the lab, a customer ordered an item using the GUI shown in Figure 10 at the counter. TM generated the plan shown in Figure 17a to bring the item from the display to the counter. If the item was not found in the display so PR failed in FindObj, KM inferred that the item could be occluded by other objects. Thus, TM generated MoveBase to adjust the robot base pose to have different view points avoiding occlusions. If the item was not found after several trials of FindObj at different locations, KM reasoned that the item could be in the storage. Thus, TM generated a predicate to move the robot to the storage. After the item was found, TM generated predicates to fulfill the item. For PickObject, TMI and AM generated grasp poses and motions for the poses.

Picking could fail if collision-free motions are not found or motions are found but the robot fails to grasp the object. When picking failed, KM reasoned about the geometric relationship of the item with other nearby objects. If the item was not directly accessible since other objects occluded the item, TM generated MoveBase several times to adjust the robot base pose to avoid occlusions. If the adjustments did not resolve the problem, TM generated motion predicates to relocate the objects occluding the item. During relocation, KM found empty slots through geometric reasoning. TM generated predicates to place the occluding objects to the empty slots. TMI and AM generated poses and motions for ReleaseObject. Figure 19a shows M-Hubo relocating the yellow cookie box that occludes the green snack can.

On the other hand, picking could fail even though the item was not occluded. Then KM reasoned about the geometric relationship between the robot, the target, and the environment. If the location of the robot was not appropriate (e.g., too far from the target or blocked by static obstacles),



(a) M-Hubo relocates the yellow cookie box to an empty space and grasps the target item.



(b) Jaco 1 relocates three objects to grasp the target item in the back.

FIGURE 19. Snapshots of comprehensive tests performing pick-and-place tasks. (a) The full architecture is deployed to M-Hubo. (b) PR, TMI, and AM are deployed to Jaco 1 and work with a third-party task planner.

TM generated motion predicates to adjust the robot base. The rest for picking and releasing was done the same with the above description.

We inserted several unexpected situations while the robot was executing motion predicates. We placed a new order while the robot was performing another task. If the robot was serving a previous order, TM added a new goal for the new order after the current goal. Thus, predicates for the new goal were added to the end of the current predicates. If the robot was replenishing the display after some items were fetched, TM added a new goal for the order in front of other goals as fulfilling an order had the highest priority. Thus, new predicates were added to the front of the existing predicates. After achieving the new goal, TM resumed performing the previous goal.

Through extensive practices for the comprehensive test spanning over few weeks, we fixed bugs in the codes and reduced errors in object pose estimation. We found that few hard-codings were inevitable. Since we did not implement the method to detect and analyze the structure of the furniture, we hard-coded the structure of the furniture. Note that the pose of the furniture was recognized using the ArUco markers but not hard-coded. We also added a rule for grasp synthesis to exclude the grasp from the top of objects. The robot hardware and the environment did not allow overhand grasps. If such overhand grasps are considered, TMI and AM attempt to find grasp poses and motions, which must end in failure. Therefore, overhand grasps were ignored by TMI. With this, we could save time for the unnecessary motion planning. After elaborating the implementation through trials and errors, M-Hubo succeeded in the fulfillment task nine times out of ten.

In addition to testing the full architecture with M-Hubo, we tested part of the architecture with Jaco 1. PR, TMI, and AM worked for object relocation with a third-party relocation planner [66]. The test showed that the modular design of our architecture is effective. Figure 19b shows Jaco 1 relocating objects to grasp the target item in the back.

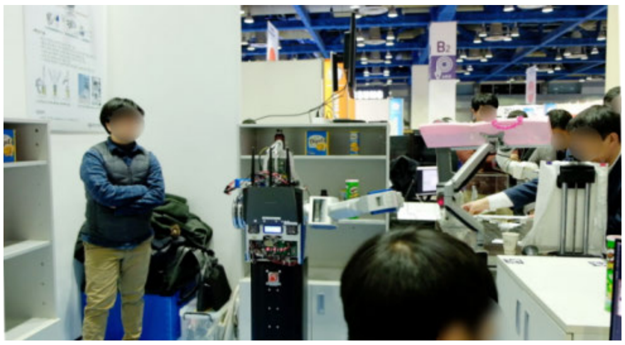


FIGURE 20. A snapshot of the public demonstration in an exhibition hall. The arrangement of the furniture was the same with the lab setting whereas their coordinates were slightly different. Several environmental conditions were unfavorable: carpeted floor incurring slips, poor lighting condition, and no wired power supplies for the bearbone computers and the robot actuators.

2) THE DEMONSTRATION

The demonstration was held in an exhibition hall (COEX in Seoul, Korea) for one day. The demonstration was open to public. The item to be ordered could be chosen by the audience. The pose of the items in the shelves also could change. The only requirement for the demonstration was that the space between items as the robot cannot grasp items if they are too close.[6]

We set up the same environment with the test environment in the lab (Figure 20). The size of the booth was slightly smaller than the lab space so the exact positions of the furniture were different (up to 5 cm). Our architecture was able to tolerate the difference. Other environmental conditions were

[6]If the robot was with nonprehensile actions like pushing and dragging, more packed arrangements could be allowed.

less favorable than the lab setting. First, the floor at the venue was carpeted so more slips occurred during navigation. Also, the carpet was wrinkled as it was not perfectly glued on the floor. Thus, the mobile base occasionally failed to reach the desired goal location. Next, the booth was dark so the objects in the shade of shelves were not detected occasionally. Sometimes, the robot was not able to detect the object to be grasped. Failing to detect nearby obstacles incurred collisions with them as they were not considered in motion planning. We did not use wired power supplies so the running time was limited by the capacity of the batteries. Some network delays seemed to occur. The delays did not directly affect the performance of the robot as the architecture was running entirely on the robot without a remote human operator.

We were not able to overcome the problem with navigation. Thus, we fine-tuned the local navigation mode such that an offset value was added to move farther. Thus, the difference of the traveled distance owing to slips could be canceled out. Even with the other unfavorable conditions, our architecture was able to recover from few failures. Especially, it was effective to adjust the base pose when the target object was not detected. If the pose of an object was noisy so the robot could not grasp the object, the robot tried again after updating the pose or changing its base pose. As a result, several rehearsals and the demonstration were finished successfully (the demonstration is shown in the attached video material).

E. DISCUSSIONS

As discussed in Section II and Table 1, the novelty of our architecture is having all the following features: i) the "full-stack" development implementing perception, knowledge base, reasoning, task planning, motion planning, task-motion interface, and human interface, ii) the ability to perform tasks without human interventions, iii) the platform-agnostic development (validated with different robot platforms), and iv) nontrivial demonstrations with physical robots. While deploying the architecture to the real world, we learned several lessons. We share them for further advances in developing a software architecture for autonomous task execution by a robot.

1) OBJECT POSE ESTIMATION IS THE KEY COMPONENT FOR SUCCESSFUL MANIPULATION

While many high-performance object detection methods have been proposed for real-time applications, pose estimation of small objects remains as a difficult task. The accuracy of state-of-the-art methods ranges between 24.5–79.2% [73] for the dataset with objects occluding each other (Occluded LINEMOD [74]). With the best known method achieving 79.2% [75], the robot would fail eight times out of ten. Even with a correct pose estimation, a pose estimate could have errors up to 2 cm in the YCB dataset [76]. In our tests, pose errors around 2 cm often incurred failures in grasping. In our tests, the occlusion of objects was less severe than the Occluded LINEMOD dataset so the pose errors were acceptable. However, we expect more failures if the objects are more occluded. Simple filtering (e.g., Kalman, particle filter) could help reduce noisy pose estimates.

2) RICHER SENSORY FEEDBACK IS NECESSARY TO DEAL WITH FAILURES

Even though the robot succeeded to grasp an object, the object could fall out from the hand if the grasp is not stable. Similarly, releasing objects could fail depending on the pose of objects in the robot hand. The information that we used to monitor task execution were limited to detect such situations that could cause task failures. Tactile sensors, force/torque sensors, or visual tracking of objects could help detect risky situations. With additional predicates describing such situations, the task planner will be able to generate plans to prevent failures.

3) HIGH-FIDELITY DYNAMIC SIMULATION COULD BE HELPFUL BUT CANNOT SUBSTITUTE PHYSICAL EXPERIMENTS

As described in the beginning of Section IV, there were gaps between simulations and the reality. Dynamic simulations could be useful to build an architecture especially checking communicated messages between different components. However, it was necessary to use a real physical robot and objects for functional testing.

4) RANDOMIZED MOTION PLANNERS DECREASE LEGIBILITY OF ROBOT MOTIONS

Those methods sample configurations of the manipulator randomly. Thus, resulting behaviors of the manipulator could be unpredictable to humans. If the robot works nearby humans, unpredictable motions would harm humans or limit human activities. It is necessary to increase legibility of robot motions instead of keeping humans away from those robots.

5) HARD-CODING IS STILL NECESSARY

As mentioned in Section IV-D2, we needed to hard-code some parameters for navigation. If the robot can learn such parameters autonomously, the architecture could work more adaptively in unknown environments. Also, more hard-codings for TMI and AM bring faster completions of tasks since we could save the time for generating poses and motions.

V. CONCLUSION

In this paper, we show our development of a software architecture for service robots performing manipulation services in human environments. The architecture enables the robot to perform the following processes autonomously without human intervention: (i) detecting objects to be manipulated and analyzing their grasp information, (ii) maintaining knowledge about the environment, (iii) reasoning about tasks and context, (iv) generating and modifying abstract task plans, (v) interfacing between symbolic task plans and motion plans by generating goal poses for the task plans, (vi) computing collision-free motions of the robot, (vii) aggregating information about motion and task planning failures for replanning, and (viii) interacting with human users through a GUI and a TTS tool. We believe that integrating all the processes into a working system and deploying it to real robots is an important contribution as such a full-stack integration working on nontrivial scenarios has been presented rarely in the literature. We believe that the implementation details and lessons learned will be helpful for future developments of working robots in human environments.

Our future work is to use the proposed architecture in environments where humans interact with robots closely. For the goal, each component of the architecture should equip additional functions. PR should be able to recognize and anticipate human activities. Also, KM needs to build and maintain models of human. TM and TMI should generate task plans and poses by considering humans. The motion plans generated from AM need to avoid areas that humans work and stay. The low-level controller must be capable of emergency stop using inputs from range sensors and force/torque sensors. We plan to improve the architecture gradually by adding those capabilities. Another future work is eliminating hard-codings for fully autonomous executions of tasks. On the other hand, we are interested in evaluating our architecture on its non-functional and quality requirements. [56] point out that a majority of studies neglect architectural evaluation but focus on the validation of the overall solution. As we also evaluate the performance of the functionalities of components and the overall service, we will measure attributes such as portability, ease of use, characteristics of software, and run-time efficiency through systematic evaluation processes. The result can be used to improve non-functional and quality requirements of the architecture.

ACKNOWLEDGMENT

The authors would like to thank Kyungwoong Kang, Jaeyoon Lee, Sangeui Lee, Kiho Lee, Eunsoo Jeon, Myungha Song, Moonyoung Lee, Saihim Cho, Dr. Hyundae Yang, and Dr. Hyunseob Park for helping the public demonstration.

REFERENCES

[1] L. Birglen and T. Schlicht, "A statistical review of industrial robotic grippers," *Robot. Comput.-Integr. Manuf.*, vol. 49, pp. 88–97, Feb. 2018.

[2] I. Akinola, J. Varley, B. Chen, and P. K. Allen, "Workspace aware online grasp planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 2917–2924.

[3] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 2917–2924.

[4] J. Luo and K. Hauser, "An empirical study of optimal motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 1761–1768.

[5] N. Hogan, "Impedance control: An approach to manipulation," in *Proc. Amer. Control Conf.*, Jul. 1984, pp. 304–313.

[6] S. Zakharov, I. Shugurov, and S. Ilic, "DPOD: 6D pose object detector and refiner," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1941–1950.

[7] *Pick-It*. [Online]. Available: https://www.pickit3d.com

[8] I. A. Sucan and S. Chitta. *MoveIt*. Accessed: Mar. 25, 2016. [Online]. Available: http://moveit.ros.org

[9] A. T. Miller and P. K. Allen, "Graspit! A versatile simulator for robotic grasping," *IEEE Robot. Automat. Mag.*, vol. 11, no. 4, pp. 110–122, Jun. 2004.

[10] C. G. Atkeson *et al.*, "What happened at the DARPA robotics challenge finals," in *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*. Springer, 2018, pp. 667–684.

[11] S. Kim *et al.*, "Team SNU's control strategies to enhancing robot's capability: Lessons from the DARPA robotics challenge finals 2015," in *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*. Springer, 2018, pp. 347–379.

[12] P. Oh, K. Sohn, G. Jang, Y. Jun, and B.-K. Cho, "Technical overview of team DRC-Hubo@UNLV's approach to the 2015 DARPA robotics challenge finals," *J. Field Robot.*, vol. 34, no. 5, pp. 874–896, Aug. 2017.

[13] R. Alterovitz, S. Koenig, and M. Likhachev, "Robot planning in the real world: Research challenges and opportunities," *AI Mag.*, vol. 37, no. 2, pp. 76–84, 2016.

[14] M. Tenorth and M. Beetz, "KnowRob: A knowledge processing infrastructure for cognition-enabled robots," *Int. J. Robot. Res.*, vol. 32, no. 5, pp. 566–590, Apr. 2013.

[15] A. Saxena, A. Jain, O. Sener, A. Jami, D. K. Misra, and H. S. Koppula, "RoboBrain: Large-scale knowledge engine for robots," 2014, *arXiv:1412.0691*. [Online]. Available: http://arxiv.org/abs/1412.0691

[16] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Proc. Workshops 24th AAAI Conf. Artif. Intell.*, 2010, pp. 33–42.

[17] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 639–646.

[18] D. B. Lenat, "CYC: A large-scale investment in knowledge infrastructure," *Commun. ACM*, vol. 38, no. 11, pp. 33–38, Nov. 1995.

[19] M. Tenorth and M. Beetz, "Representations for robot knowledge in the KnowRob framework," *Artif. Intell.*, vol. 247, pp. 151–169, Jun. 2017.

[20] S. Lemaignan, M. Warnier, E. A. Sisbot, A. Clodic, and R. Alami, "Artificial cognition for social human–robot interaction: An implementation," *Artif. Intell.*, vol. 247, pp. 45–69, Jun. 2017.

[21] M. Tenorth, A. C. Perzylo, R. Lafrenz, and M. Beetz, "Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth framework," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 3, pp. 643–651, Jul. 2013.

[22] M. Beetz, M. Tenorth, and J. Winkler, "Open-EASE—A knowledge processing service for robots and robotics/ai researchers," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 1983–1990.

[23] G. H. Lim, I. H. Suh, and H. Suh, "Ontology-based unified robot knowledge for service robots in indoor environments," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 41, no. 3, pp. 492–509, May 2011.

[24] I. Hong Suh, G. Hyun Lim, W. Hwang, H. Suh, J.-H. Choi, and Y.-T. Park, "Ontology-based multi-layered robot knowledge framework (OMRKF) for robot intelligence," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2007, pp. 429–436.

[25] J. Hoffmann, "FF: The fast-forward planning system," *AI Mag.*, vol. 22, no. 3, p. 57, 2001.

[26] M. Helmert, "The fast downward planning system," *J. Artif. Intell. Res.*, vol. 26, pp. 191–246, Jul. 2006.

[27] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[28] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 489–494.

[29] N. T. Dantam, S. Chaudhuri, and L. E. Kavraki, "The task-motion kit: An open source, general-purpose task and motion-planning framework," *IEEE Robot. Autom. Mag.*, vol. 25, no. 3, pp. 61–70, Sep. 2018.

[30] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 1, pp. 104–136, Jan. 2018.

[31] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2015, pp. 1923–1929.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[33] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vis.*, vol. 104, no. 2, pp. 154–171, Sep. 2013.

[34] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. Appl.*, vol. 13, no. 4, pp. 18–28, Jul./Aug. 2008.

[35] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.
[36] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
[38] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*. [Online]. Available: http://arxiv.org/abs/1804.02767
[39] S. Jain and B. Argall, "Grasp detection for assistive robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 2015–2021.
[40] J. Zhang, M. Li, Y. Feng, and C. Yang, "Robotic grasp detection based on image processing and random forest," *Multimedia Tools Appl.*, vol. 79, nos. 3–4, pp. 2427–2446, Jan. 2020.
[41] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 769–776.
[42] F.-J. Chu, R. Xu, and P. A. Vela, "Real-world multiobject, multigrasp detection," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3355–3362, Oct. 2018.
[43] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," *Multimodal Technol. Interact.*, vol. 2, no. 3, p. 57, Sep. 2018.
[44] T. Houliston, J. Fountain, Y. Lin, A. Mendes, M. Metcalfe, J. Walker, and S. K. Chalup, "NUClear: A loosely coupled software architecture for humanoid robot systems," *Frontiers Robot. AI*, vol. 3, Apr. 2016.
[45] S. Garcia, C. Menghi, P. Pelliccione, T. Berger, and R. Wohlrab, "An architecture for decentralized, collaborative, and autonomous robots," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2018, pp. 75–7509.
[46] S. Wan, Z. Gu, and Q. Ni, "Cognitive computing and wireless communications on the edge for healthcare service robots," *Comput. Commun.*, vol. 149, pp. 99–106, Jan. 2020.
[47] Y. Jiang, N. Walker, M. Kim, N. Brissonneau, D. S. Brown, J. W. Hart, S. Niekum, L. Sentis, and P. Stone, "LAAIR: A layered architecture for autonomous interactive robots," 2018, *arXiv:1811.03563*. [Online]. Available: http://arxiv.org/abs/1811.03563
[48] R. Shah, Y. Jiang, H. Karnan, G. Briscoe-Martinez, D. Mulder, R. Gupta, R. Schlossman, M. Murphy, J. W. Hart, L. Sentis, and P. Stone, "Solving service robot tasks: UT Austin Villa@Home 2019 team report," 2019, *arXiv:1909.06529*. [Online]. Available: http://arxiv.org/abs/1909.06529
[49] A. Koubaa, M.-F. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, and A. Mahmoud, "Turtlebot at office: A service-oriented software architecture for personal assistant robots using ROS," in *Proc. Int. Conf. Auto. Robot Syst. Competitions (ICARSC)*, May 2016, pp. 270–276.
[50] A. Koubaa, M.-F. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, and A. Mahmoud, "Mybot: Cloud-based service robot using service-oriented architecture," *Robótica*, vol. 107, pp. 8–13, 2017.
[51] T. Keleştemur, N. Yokoyama, J. Truong, A. A. Allaban, and T. Padir, "System architecture for autonomous mobile manipulation of everyday objects in domestic environments," in *Proc. 12th ACM Int. Conf. Pervas. Technol. Rel. Assistive Environ.*, Jun. 2019, pp. 264–269.
[52] C.-H. King, T. L. Chen, Z. Fan, J. D. Glass, and C. C. Kemp, "Dusty: An assistive mobile manipulator that retrieves dropped objects for people with motor impairments," *Disab. Rehabil., Assistive Technol.*, vol. 7, no. 2, pp. 168–179, Mar. 2012.
[53] A. Jain and C. C. Kemp, "EL-E: An assistive mobile manipulator that autonomously fetches objects from flat surfaces," *Auto. Robots*, vol. 28, no. 1, p. 45, 2010.
[54] M. Ciocarlie, K. Hsiao, A. Leeper, and D. Gossow, "Mobile manipulation through an assistive home robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5313–5320.
[55] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe, "HERB: A home exploring robotic butler," *Auto. Robots*, vol. 28, no. 1, pp. 5–20, Jan. 2010.
[56] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *J. Syst. Softw.*, vol. 122, pp. 16–39, Dec. 2016.
[57] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *Amer. Statistician*, vol. 46, no. 3, pp. 175–185, Aug. 1992.
[58] S. Lee and I. Kim, "A robotic context query-processing framework based on spatio-temporal context ontology," *Sensors*, vol. 18, no. 10, p. 3336, Oct. 2018.
[59] M. J. Huber, "JAM: A BDI-theoretic mobile agent architecture," in *Proc. 3rd Annu. Conf. Auto. Agents (AGENTS)*, 1999, pp. 236–243.
[60] J. Rintanen, "Complexity of concurrent temporal planning," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 7, 2007, pp. 280–287.
[61] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.
[62] M. Lee, Y. Heo, J. Park, H.-D. Yang, H.-D. Jang, P. Benz, H. Park, I. S. Kweon, and J.-H. Oh, "Fast perception, planning, and execution for a robotic butler: Wheeled humanoid M-Hubo," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 1–8.
[63] M. Lee, Y. Heo, S. Cho, H. Park, and J.-H. Oh, "Motion generation interface of ROS to PODO software framework for wheeled humanoid robot," in *Proc. 19th Int. Conf. Adv. Robot. (ICAR)*, Dec. 2019, pp. 456–461.
[64] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 1321–1326.
[65] T. Jung, J. Lim, H. Bae, K. K. Lee, H.-M. Joe, and J.-H. Oh, "Development of the humanoid disaster response platform DRC-HUBO+," *IEEE Trans. Robot.*, vol. 34, no. 1, pp. 1–17, Feb. 2018.
[66] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, "Fast and resilient manipulation planning for target retrieval in clutter," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Mar. 2020, pp. 1–7.
[67] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. ICRA. Millennium Conf. IEEE Int. Conf. Robot. Automat. Symp.*, vol. 2, Apr. 2000, pp. 995–1001.
[68] D. Devaurs, T. Simeon, and J. Cortes, "Enhancing the transition-based RRT to deal with complex cost spaces," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 4120–4125.
[69] J. Meijer, Q. Lei, and M. Wisse, "An empirical study of single-query motion planning for grasp execution," in *Proc. IEEE Int. Conf. Adv. Intell. Mechatronics (AIM)*, Jul. 2017, pp. 1234–1241.
[70] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, Jun. 2014.
[71] E. Marder-Eppstein. *ROS Move_Base Package*. [Online]. Available: http://wiki.ros.org/move_base
[72] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 1271–1278.
[73] *Papers With Codes: 6D Pose Estimation using RGB*. [Online]. Available: https://paperswithcode.com/sota/6d-pose-estimation-on-occludedlinemod
[74] A. Krull, E. Brachmann, F. Michel, M. Y. Yang, S. Gumhold, and C. Rother, "Learning analysis-by-synthesis for 6D pose estimation in RGB-D images," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 954–962.
[75] C. Song, J. Song, and Q. Huang, "HybridPose: 6D object pose estimation under hybrid representations," 2020, *arXiv:2001.01869*. [Online]. Available: http://arxiv.org/abs/2001.01869
[76] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and model set: Towards common benchmarks for manipulation research," in *Proc. Int. Conf. Adv. Robot. (ICAR)*, Jul. 2015, pp. 510–517.

CHANGJOO NAM (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, South Korea, in 2009, and 2011, respectively, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2016. He was a Postdoctoral Fellow with the Robotics Institute, Carnegie Mellon University. He is currently a Senior Research Scientist with the Robotics and Media Institute, Korea Institute of Science and Technology, Seoul. His research interest includes task planning for multirobot coordination and manipulation.

SEOKJUN LEE received the M.S. and Ph.D. degrees in computer science from Kyonggi University, South Korea, in 2017 and 2020, respectively. He is currently a Postdoctoral Researcher with Kyonggi University. His research interests include machine learning, knowledge representation and reasoning, task and motion planning, computer vision, and intelligent robotics systems.

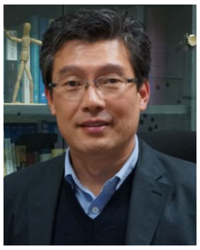
JEONGHO LEE (Student Member, IEEE) received the B.S. and M.S. degrees in mechanical system design engineering from the Seoul National University of Science and Technology, in 2015 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Korea University. He is also a Researcher with the Robotics and Media Institute, Korea Institute of Science and Technology, Seoul, South Korea. His research interests include object detection, affordance detection, and generation of grasp information.

SANG HUN CHEONG (Member, IEEE) received the B.S. degree in robotics from Kwangwoon University, in 2017, and the M.S. degree in mechanical engineering from Korea University, in 2019. He is currently a Researcher with the Robotics and Media Institute, Korea Institute of Science and Technology, Seoul, South Korea. His research interests include task and motion planning, and object manipulation.

DONG HWAN KIM (Member, IEEE) received the B.S. degree in electrical engineering and the M.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University (SNU), Seoul, South Korea, in 1999, 2001, and 2006, respectively. From 2006 to 2007, he was a Senior Engineer with Samsung Electronics, Company Ltd., Suwon, South Korea. In 2007, he joined the Korea Institute of Science and Technology (KIST), Seoul, where he is currently a Principal Researcher with the Center for Intelligent Robotics. During his period at KIST, he stayed at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, as a Visiting Research Associate, from 2007 to 2010. His current research interests include computer vision, pattern recognition, machine learning, image processing, and their applications, including object recognition, image segmentation, visual grasp affordance detection, and 3D reconstruction.

CHANGHWAN KIM (Member, IEEE) received the B.S. degree in mechanical engineering and the M.S. degree in machine design engineering from Hanyang University, Seoul, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in mechanical engineering from The University of Iowa, Iowa City, IA, USA, in 2002. From 2002 to 2004, he was a Research Associate with the Robotics and Automation Laboratory, University of Notre Dame, Notre Dame, IN, USA. He is currently with the Robotics and Media Institute, Korea Institute of Science and Technology (KIST), Seoul. His research interests include human motion imitation and motion generation of a humanoid, human modeling, motion planning of mobile robots, cooperation of multiple robots, and rehabilitation robots.

INCHEOL KIM received the M.S. and Ph.D. degrees in computer science from Seoul National University, South Korea, in 1987 and 1995, respectively. He is currently a Professor with the Department of Computer Science, Kyonggi University, South Korea. His research interests include machine learning, knowledge representation and reasoning, task and motion planning, computer vision, and intelligent robotics systems.



SUNG-KEE PARK (Member, IEEE) received the B.S. and M.S. degrees in mechanical design and production engineering from Seoul National University, Seoul, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in computer vision from the Korea Advanced Institute of Science and Technology, Seoul, in 2000. He has been with the Robotics and Media Institute, Korea Institute of Science and Technology (KIST), Seoul, where he is currently a Principal Research Scientist. During his period at KIST, he held a visiting position with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, in 2005, where he was involved in object recognition. His research interests include robot vision and robot intelligence.