Symbolic reasoning about unseen objects from multimodal sensory feedback for manipulation

Arthur N. Helsloot, Mert Imre, Carlos Hernandez Corbato, and Jens Kober

***Abstract*— Robotically manipulating objects can be very challenging when not all of the environment can be fully observed, e.g. in environments which are physically and visually accessible from only a single side. By using multimodal sensory feedback and symbolic reasoning, conclusions can be drawn about the presence of objects that cannot be observed directly. This paper presents the Symbolic Reasoning for Partially Observable Environments (SyRePOE) system, which uses an ontology to maintain its world model and a reasoner to infer information about unobservable objects. SyRePOE is demonstrated in simulation and on a real robot, where it is tasked with stocking a retail shelf.**

I. INTRODUCTION

In daily life, humans often are faced with small environments accessible only from a single side, such as cupboards in kitchens, or cabinets in retail stores. In such environments only a limited point of view, mostly only from the front, is available. As a result, there may be occlusions that cause some objects present in the environment to be partially visible, not visible, or not reachable for manipulation. Still, we manage to accomplish tasks in those environments, without the need to observe all the objects. For example, a shelf stocking task in a retail store can be completed without taking all products off the shelf first. To get robots to operate in such limited environments for similar tasks, there is a need for understanding the existence and placement of all objects, including the ones that are not directly observable.

To address this problem for specific tasks, like stocking of retail shelves, a pure decision making or planning approach could be taken, e.g. through Behavior Trees [1] or a Planning Domain Definition Language (PDDL) [2] based method. Such methods, however, have difficulty reasoning over and updating their current state, making it difficult to create a not task specific solution. Ontology based approaches can much more easily reason over and update their current state, and can be easily integrated into existing ontology based systems. Additionally, symbolic reasoning over the ontology content is a very powerful tool for dealing with dynamic environments, like those shared with humans.

To this end, we introduce the Symbolic Reasoning for Partially Observable Environments (SyRePOE) system. SyRePOE uses an ontology based approach with symbolic reasoning and multimodal sensory feedback to help the system understand what is happening in unobservable parts of its workspace and use that information for task planning. SyRePOE is applied to a shelf stocking task both in simulation and the real world. An impression of the system performing this task is given in Figure 1.

Fig. 1: Impression of SyRePOE stocking a shelf. On the left the Franka Emika Panda, the objects and shelves are shown. On the right the view of the shelf for pose detection is visible.

In short, the contributions of this paper are twofold:

1) a reasoner capable of inferring information about not directly observable objects, and
2) use of said information in task planning.

The remainder of this paper starts with an overview of related literature in Section II. Section III explains the SyRePOE system, its components, and its capabilities. This is followed by an example application of the system to stocking a retail shelf in Section IV. Lastly, Section V proposes future work for SyRePOE and presents some concluding remarks.

II. RELATED WORK

Multiple frameworks have been set up to apply ontologies for knowledge processing in robotics. One such framework is OUR-K [3], which uses logical reasoning, Bayesian inference, and heuristics to solve high level queries like 'bring me a cup', even under incomplete information. Another such framework is Knowrob [4]. This ROS integrated system uses ontologies, reasoning, episodic memories and world model simulations to help robots complete abstract tasks like setting a table or making a pizza. Knowrob has been shown to be quite versatile [5]–[8], though its implementation does not follow documented standardization, limiting its potential for integration in larger systems. Knowrob has been utilized by Winkler et al. [9] in a retail environment. They use knowledge about the shelves and objects to plan manipulation of partially occluded objects, removing obstructing objects on the way. They, however, only deal with partial occlusions, still allowing for full observability of the scene.

An interesting framework for processing Perception and Manipulation Knowledge (PMK) is presented by Diab et

Authors are with the AIRLab and Cognitive Robotics Department, Delft University of Technology, the Netherlands.
A.N.Helsloot@student.tudelft.nl,
{M.Imre, C.H.Corbato, J.Kober}@tudelft.nl

al. [10]. PMK uses a standardized ontology, modeled under SUMO [11], as a basis for the world model of the robot's environment and asserts new observations to it online. This symbolic knowledge of its environment is then used to enhance performance of the Task And Motion Planning (TAMP) module. In TAMP the high level decision making of what task to do and the lower level motion planning of how to accomplish this task are intertwined, allowing for the selection of alternative task plans if a certain motion plan turns out to be, for example, kinematically infeasible. By integrating symbolic knowledge into this process, queries about how to perform certain tasks and motions can be answered online, e.g. about reachability or collision objects. PMK allows for answering complex queries about spatial relations, object features, perception and planning. It can specify individual sensors and signal processing algorithms in the ontology, which allows PMK to answer queries like *What sensors are available to detect the color of this object?* Additionally, PMK explicitly specifies its tasks and actions in the ontology, allowing it to be deployed in a wide range of applications.

In another work by Diab et al. [12], they introduce an ontology specifically aimed at identifying and resolving geometrical, hardware and software failures. Their ontology, modeled under SUMO [11] and DUL [13], analyzes occurring failures and suggests alternative solutions to resolve the failures. The purpose is to provide a general solution, applicable independent from the task specifics. Self-adaptive systems could benefit from such a failure ontology, due to its self-diagnostic and solution proposing capabilities. Such an ontology is potentially extensible to detecting inconsistencies or oddities in the world model and proposing scenarios to explain them.

All the systems mentioned so far strongly rely on the observations and do not consider objects that are not directly detected. Therefore, they would fail in environments where not all objects are observable.

III. SYREPOE

In this section, the proposed system, SyRePOE, is explained. We first provide a brief overview of the approach which is visualized in Figure 2. SyRePOE consists of four modules, namely perception, ontology, reasoner, and planning. The perception module processes all the sensory information and passes the extracted information to the ontology. It consists of integrated off-the-shelf components. The ontology contains the knowledge of the system and maintains a world model. The reasoner module evaluates the world model, resolves the detected inconsistencies by reasoning, and updates the ontology. The ontology module and the reasoner module are the most significant components of the system and will be explained in further detail in Sections III-A and III-B, respectively. The planning module chooses the next action and action parameters based on the current world model in the ontology. It takes a PDDL [2] translation of the world model and passes that to the Fast-Forward planner [14]. The first action of the returned plan is then executed. The purpose of this action to get the system closer to the goal state, either via successful execution or via detecting an unsuccessful execution and so retrieving new information about the environment.

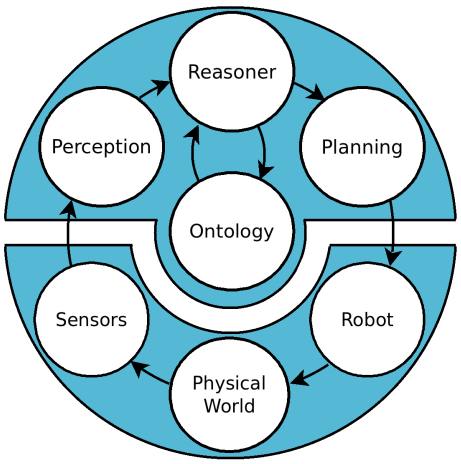


Fig. 2: Diagram showing the ontology and reasoner approach used in SyRePOE. The top section represents software, the bottom section hardware.

Below, first Subsection III-A will describe how the ontology represents and maintains its knowledge of environment structures and object shapes, and its world model. Then Subsection III-B presents the logic used for processing results of actions and inferring positions of fully occluded objects in the environment. This section ends with Subsection III-C, which presents the assessment of SyRePOE on the scales of autonomy recently proposed by Olivares-Alarcos et al. [15] to show the relation of SyRePOE to current literature.

A. The SyRePOE ontology

The ontology, implemented in OWL [16] through the use of Protégé [17], is initialized with knowledge about the environment, and object types it can expect to find in the environment. Most notably it contains dimensions and locations of sub-environments, like specific shelves or surfaces, dimensions of object types, and specifications of what object type can be expected in which sub-environment. This information can be used in proposing solutions to inconsistencies in the world model, e.g. for inferring presence of unseen objects explained in Section III-B. At run-time the ontology maintains a world model containing poses of the objects in specific sub-environments, and when and how these objects were last detected by asserting new information gathered from measurements.

An example of how an object is represented in the world model is shown in Figure 3. Blue boxes indicate classes in the ontology, ovals indicate instances, and arrows indicate object properties. Here 'object 1' is an instance of class 'Object'. This example object is of type 'type 1', as visible in the top right corner of Figure 3, indicated through the object property 'hasType'. Each instance of class 'ObjectType' has data properties indicating the spatial dimensions of this type of object. 'object 1' has object property 'onShelf', relating it to the instance 'shelf 1', which is the instance of class

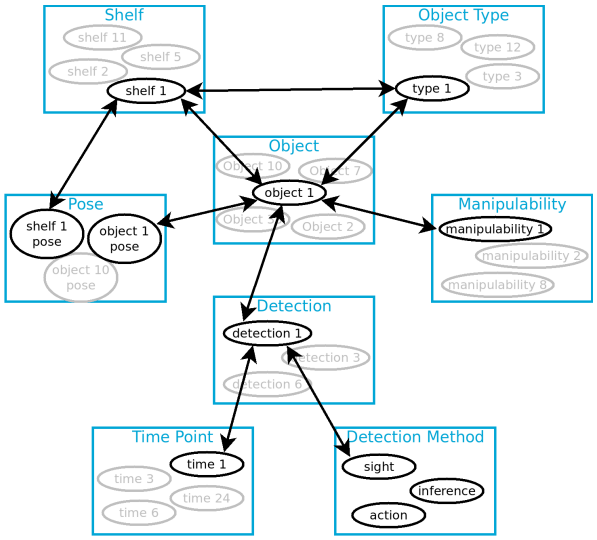


Fig. 3: Object description in the SyRePOE ontology. Blue boxes indicate classes, ovals indicate instances, arrows indicate object properties.

'Shelf'. Such shelves are examples of sub-environments. Instances of class 'Shelf', shown in the top left of Figure 3, also have data properties indicating their spatial dimensions. Additionally, each shelf relates to an instance of class 'ObjectType' through a 'containsType' object property. This indicates that all objects found on this shelf can be expected to be of that type. Both the shelf and the object have a defined pose. The pose of the shelf indicates where in the world the robot can find this sub-environment. The pose of the object indicates where in the sub-environment this object can be found. These poses are all instances of class 'Pose' and are connected to their object or sub-environment using the 'hasPose' object property. Each instance of class 'Pose' has a data property for each coordinate of its pose. 'object 1' also has an instance of class 'Manipulability' related to it through the 'hasManipulability' object property, shown right in Figure 3. This Manipulability class represents the belief on whether or not an object can be moved from its current pose by performing an action. As a consequence, the manipulability of an object depends on the action and its direction. Currently, the only action in the library of SyRePOE that can change the pose of an object in a sub-environment is pushing an object in the direction away from the open side of the sub-environment, namely a backwards push. Thus, the manipulability in the current version can be considered as binary variable, represented as a data property of the instance of class 'Manipulability'. Each object in the ontology always has a manipulability, but its value is not always known.

Each object also has at least one object property 'hasDetection' to an instance of class 'Detection', as shown at the bottom of Figure 3. This detection indicates how the object was detected and when that happened, though object properties 'hasDetectionMethod' and 'hasTimePoint' respectively. The method of detection can be one of three instances of class 'Detection Method', namely 'sight', 'inference' or 'action'. If the object has been observed visually, detection method 'sight' is used. If the object has been inferred to be present, but has not actually been seen, detection method 'inference' is used. When an action has been performed which has caused an object to end up at its current pose, detection method 'action' is used. The time of detection is registered by an instance of class 'TimePoint', which has a data property 'hasTime' indicating the time stamp of detection.

The SyRePOE ontology has been modeled under the PMK [10] ontology. Since PMK is modeled under SUMO [11], the SyRePOE ontology is automatically modeled under SUMO. Below a selection of the main classes in the SyRePOE ontology is shown together with a description of how each class could be represented in PMK.

- ObjectType $\subseteq$ Artifact with specified spatial dimensions
- DetectionMethod $\subseteq$ DeviceGroup
- Manipulability $\subseteq$ Quantity
- Pose $\subseteq$ QuantityAggregation
- Shelf $\subseteq$ Region with an ObjectType and a Pose
- Object $\subseteq$ Artifact with a Manipulability and a Pose

By integrating SyRePOE and PMK in future work, the strengths of the two systems could be combined, allowing the tight integration with TAMP offered by PMK to benefit from the world model reasoning capabilities offered by SyRePOE.

B. The SyRePOE reasoner

The reasoner of SyRePOE, implemented using SWI-Prolog [18], updates the world model through inference and passes requested information to other modules, like the planning or perception module as in Figure 2. It has four specific features.

Firstly, it supplies an interface for other modules to access the knowledge stored in the ontology. It does this by offering standard queries to request sets of information from the ontology, such as all dimensions of a certain object or sub-environment, or the time an object was last detected.

Secondly, it asserts the effect of actions to the ontology. After an action is executed, the action intent and the action result (either success or failure) are passed to the reasoner. The logic rules applied for a pushing action are shown in Eq. (1)-(5). In these and following equations $Object$ is the set of all instances of class 'Object'. Additionally, to keep the logic understandable, the rules are simplified compared to the real implementation to exclude detections and pose error margins. Also object specific depth is simplified to a single value for all objects. For Eq. (1)-(5), notice that the pushing actions considered are backwards pushes, that are defined along the positive y direction of the local coordinate frame of the sub-environment. Eq. (1) defines the variables for the equations below it. Here x and y are the coordinates of the start of the pushing action, t_s and t_e are the start and end time of the pushing action respectively, δ_y is the pushing distance, d is the depth the object itself spans, and s is the action result, which is a binary variable for representing whether an action intent is met or not. The action result is based on haptic feedback. An estimate of the effort the robot exerts while performing the action is compared to an

empirically determined threshold. If the effort exceeds this threshold, the action is aborted and assumed to have failed. Otherwise the action intent is considered the action effect. Based on the action intent and result, the reasoner concludes the effect on the objects. For the pushing example, when the action fails ($s = 0$), as in Eq. (2), the pose of the object that was attempted to be pushed will not be changed and its manipulability will be set to false, indicating that this particular object is blocked from moving further backwards. When the action succeeds ($s = 1$), as in Eq. (3), the reasoner recursively uses Eq. (4) and (5) to check which objects are behind the pushed object and so close to the pushed object that they will be moved as well, and updates the poses of these objects in the world model. It also asserts a new detection for each of these objects of method 'action'.

$$x, y, t_s, t_e \in \mathbb{R} \qquad s \in \{0,1\} \qquad \delta_y, d \in \mathbb{R}^+ \tag{1}$$

$$\big((s=0) \wedge (\exists O \in Object) obj\_at(O,x,y,t_s)\big) \rightarrow \big(\neg manipulable(O) \wedge obj\_at(O,x,y,t_e)\big) \tag{2}$$

$$\big((s=1) \wedge (\exists O \in Object) obj\_at(O,x,y,t_s)\big) \rightarrow push(O,\delta_y) \tag{3}$$

$$\Big((\exists \Delta \in \mathbb{R}^+)(\exists O \in Object) push(O,\Delta) \wedge \big(\forall P \in \{Object \backslash \{O\}\}\big)\Big(\neg obj\_behind(O,P) \vee \Big((\exists X_p \in \mathbb{R})(\exists Y_p \in \mathbb{R})\big(obj\_at(P,X_p,Y_p,t_s) \wedge (y+d+\Delta \leq Y_p)\big)\Big)\Big)\Big) \rightarrow obj\_at(O,x,y+\Delta,t_e) \tag{4}$$

$$\Big((\exists \Delta \in \mathbb{R}^+)(\exists O \in Object)\Big(push(O,\Delta) \wedge \big(\exists P \in \{Object \backslash \{O\}\}\big)\Big(obj\_behind(O,P) \wedge (\exists X_p \in \mathbb{R})(\exists Y_p \in \mathbb{R})\big(obj\_at(P,X_p,Y_p,t_s) \wedge (y+d+\Delta > Y_p)\big)\Big)\Big)\Big) \rightarrow \big(obj\_at(O,x,y+\Delta,t_e) \wedge push(P,y+d+\Delta-Y_p)\big) \tag{5}$$

Thirdly, it asserts the poses of observed objects to the ontology. After execution of an action the perception module is called. The poses of manipulated objects are then confirmed or rejected through visual information. Additionally, after each observation, the entire world model is analyzed based on a few aspects. First, the reasoner checks for intersecting objects in the world model and removes objects causing geometric conflicts from the world model. Second, the reasoner updates the manipulability of each object by propagating manipulability through lines of objects and from the back of the sub-environment using the logic rules expressed in Eq. (6)-(8). The interpretation of these equations is that an object is not manipulable if it is at the back of a sub-environment (Eq. (7)), or if it is directly in front of, or directly behind an object which is not manipulable (Eq. (7) and (8), respectively).

$$(\exists O \in Object)\big(prop\_back(O) \vee prop\_front(O)\big) \rightarrow \neg manipulable(O) \tag{6}$$

$$(\exists O \in Object)\Big(\neg manipulable(O) \vee obj\_at\_back(O) \vee \big(\exists P \in \{Object \backslash \{O\}\}\big)\big(obj\_direct\_behind(O,P) \wedge prop\_back(P)\big)\Big) \rightarrow prop\_back(O) \tag{7}$$

$$(\exists O \in Object)\Big(\neg manipulable(O) \vee \big(\exists P \in \{Object \backslash \{O\}\}\big)\big(obj\_direct\_front(O,P) \wedge prop\_front(P)\big)\Big) \rightarrow prop\_front(O) \tag{8}$$

Lastly, the reasoner infers presence of unobservable objects in the sub-environment. Eq. (9) and (10) present the logic rules used for this. Here variables for the object depth d_O and the sub-environment depth d_S are used. According to Eq. (9), if an object is registered as not manipulable, and it is not at the back, nor does the object have another object directly behind it, then there must be a block behind this object. If the dimensions of this block could fit an integer number of objects (Eq. (9)), objects are recursively added in between the blocked object and the back of the sub-environment by Eq. (10). These objects are asserted as instances to the world model with a detection of detection method 'inference'. For simplicity, in the equations shown here it is assumed there are no known objects in between the block and the back of the shelf.

$$(\exists O \in Object)\Big(\neg manipulable(O) \wedge \neg obj\_at\_back(O) \wedge \big(\exists P \in \{Object \backslash \{O\}\}\big) \neg obj\_direct\_behind(O,P) \wedge (\exists X_b \in \mathbb{R})(\exists Y_b \in \mathbb{R})\big(obj\_at(O,X_b,Y_b-d_O) \wedge (\exists N \in \mathbb{N}_0)(d_O N = d_S - Y_b + 0.5 d_O)\big)\Big) \rightarrow block(X_b,Y_b,N) \tag{9}$$

$$(\exists X_b \in \mathbb{R})(\exists Y_b \in \mathbb{R})(\exists N \in \mathbb{N}_0) block(X_b,Y_b,N) \rightarrow (N=0) \vee \big((\exists O \in Object) obj\_at(O,X_b,Y_b) \wedge block(X_b,Y_b+d_O,N-1)\big) \tag{10}$$

C. SyRePOE on autonomy scales

Olivares-Alarcos et al. [15] review several frameworks that use ontologies to enhance autonomy in robotics. They do so based on different scopes, among which an Ontology Scope that checks if certain concepts are defined in the ontology, and a Reasoning Scope that goes into the capabilities a reasoning system should have to exhibit autonomy. Here SyRePOE is reviewed based on these same scales to show the relation of SyRePOE to current literature.

1) Ontology Scope: Of the concepts an ontology for autonomous robotics should have according to [15], only the concepts 'Object' and 'Environment map' are covered by SyRePOE. These, together with the concept 'Affordance', are discussed here. For more details about the other concepts

used by Olivares-Alarcos et al. please consult [15]. In the SyRePOE ontology an 'Object' represents a physical thing that is at a specific pose in a specific sub-environment. This falls under the definition used by Olivares-Alarcos and therefore 'Object' can be considered covered in the SyRePOE ontology. Olivares-Alarcos defines an 'Environment Map' as a description of the working environment of the system. For the SyRePOE system shelves are working environments and therefore the 'Shelf' class can be considered a subclass of 'Environment Map'. The concept of 'Affordance' entails the actions an object allows to be performed with or on it. No consensus has been reached yet on how to model this concept, though many suggestions have been made [19]–[21]. The 'Manipulability' class in the SyRePOE ontology defines for the action of moving an object away from the robot, whether it can be performed on this object or not. This is part of what affordance is, though it is nowhere near the full definition of affordance. Therefore 'Affordance' is not considered covered by the SyRePOE ontology.

2) Reasoning Scope: Of the reasoning skills considered in [15], only the concepts "Reasoning and belief maintenance" and "Execution and action" are covered by SyRePOE. These, together with "Perception and situation assessment" and "Problem solving and planning", are discussed here. For more details about other reasoning skills please consult [15].

"Reasoning and belief maintenance" The SyRePOE reasoner evaluates the environmental knowledge to propagate manipulability and infer presence of unseen objects. These are clear examples of reasoning for belief maintenance.

"Execution and action" SyRePOE reasons about the effects an action is expected to have on the world. It considers both what happens when the action is successful and when the action failed. This is a clear example of reasoning over actions and their execution.

"Perception and situation assessment" Perception is aimed at the recognition of objects and events in the scene, while situation assessment means to evaluate what has been perceived and draw conclusions from that. Since the input expected from the perception module to the SyRePOE system is a list of object poses, perception cannot be considered a part reasoned about. Situation assessment, however, is considered in SyRePOE, since it allows for queries like *What objects are to the right of object A?* or *Are objects A and B in contact?*.

"Problem solving and planning" SyRePOE does not plan itself. Instead it translates information from its ontology to a PDDL problem description and passes that on to an external planner module, which will select the next action to take. Since the ontology is not involved in planning, Olivares-Alarcos et al. [15] would not consider planning to be covered by SyRePOE.

IV. CASE-STUDY

The reasoning capacities of the SyRePOE are shown both on a simulation setup and the real robot in similar scenarios.

A. Scenario description

In the scenario, visible in Figures 4 and 5, a Franka Emika Panda arm is positioned statically in front of a cabinet of shelves. The goal is to stock the shelf with objects in a grid-like organization. A couple objects are positioned on the shelf in advance, though it is unknown to the system how many and where. The ontology is initialized with knowledge of spatial dimensions of the shelf and the objects on it, where the shelf is with respect to the robot, and how many objects could fit on the shelf.

For action planning ROSPlan [22] is used. The actions available to the planner are placing a new object at the front of the shelf in any line, or pushing the front object backwards by one object depth in any line. In simulation, trajectory generation and tracking is realized through MoveIt [23]. For the real robot, trajectories are generated offline using an inverse kinematics algorithm and tracked online using an Active Inference Controller (AIC) [24]. A new object will be available at a fixed pose near the robot when needed. To allow room for the gripper, some space is left in between lines of objects on the shelf.

B. Simulation

The simulation runs in Gazebo. The haptic feedback is realized by looking at sudden increases in joint effort. If such a sudden increase is detected, the action is aborted and considered to have failed.

The shelf can contain a maximum of 10 objects. Perception is realized by asking Gazebo for the poses of the objects and passing only the poses of the front most objects to the reasoner.

The initial configuration of objects on the shelf is generated as a grid of objects, where each position in the grid has an equal probability of being occupied or empty. Each object pose is disturbed by maximally one tenth the object width sideways and maximally one object depth backwards if that does not result in any collisions. By adding such random deviations in object poses a more realistic configuration is generated, compared to starting with each objects strictly in the aforementioned grid.

In running the simulation, it has been shown SyRePOE is able to correctly infer presence of occluded objects and the system can reliably stock the shelf. An example execution is

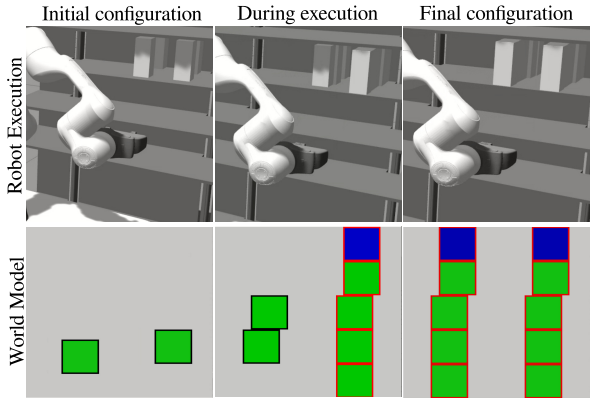


Fig. 4: Three snapshots from the experiments in Gazebo, and the corresponding world models are provided on the top and the bottom, respectively.

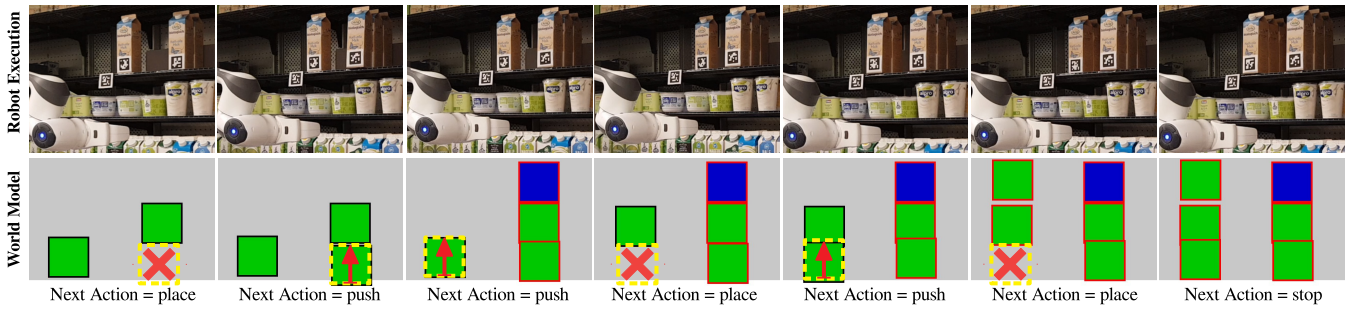


Fig. 5: Snapshots from the real world experiments showing SyRePOE stocking an initially partially full shelf. The environment is shown on top and the world model SyRePOE has at that time on the bottom, together with the selected action and action target in yellow dashed outline. The shelf is initialized with three objects (left), and is fully stocked in the end (right).

shown in Figure 4. In the world model each square represents an object. Green or blue filling means the object has been seen or inferred to be there, respectively. Black borders mean the object is potentially manipulable, red borders mean the object is not manipulable. Figure 4 shows snapshots of before, during and after the execution and the world models of SyRePOE for corresponding times. It can be seen that 10 objects have been successfully stocked and SyRePOE has correctly inferred the presence of objects which could not have been observed directly.

C. Real robot

Similar scenarios have been tested on a real robot in a mock-up store. The rules as stated in Section IV-A are the same. Figure 1 shows a photo of the setup. For these experiments milk cartons were used. A maximum of 6 objects fit the shelf. For perception, Aruco markers [25] were mounted on the shelf and on each object. From these markers the pose of each object with respect to the shelf was calculated. By making use of relative positions of markers, the position of the camera becomes irrelevant. Even though each object has a unique marker, the markers are not used to identify individual objects, only their poses relative to the shelf marker. To generate the motion trajectories an inverse kinematics algorithm is used. The trajectories were generated offline to speed up the execution. The use of the AIC for trajectory tracking allows for compliance and easy access to opposing forces. The horizontal component of the estimate of the opposing force to the end effector supplied by the AIC is used for haptic feedback. If this component exceeds a predefined threshold, the action is aborted and considered to have failed.

The robot was faced with three scenarios: a completely empty shelf, a completely full shelf, and a partially full shelf. Three times the system was given the same task, to stock the shelf. In all three cases the system succeeded. Figure 5 shows photos and visualizations of the world model from each step in stocking the partially full shelf. These images show the system successfully accomplishes its task and correctly inferred the presence of an object which could not have been observed directly. Note that the shelf initially contains three objects, but only two are visible. The third object is inferred after the push action in the second step. Videos showing the execution and results of all three experiments are available as supplementary material.

During execution the planning time of each iteration was measured. This planning time is defined as the time the system takes from having just completed the previous action to having decided which action to execute next. Over all experiments the mean planning time is 0.6176 seconds with the maximum at 1.165 seconds.

V. DISCUSSION AND CONCLUSION

In this section, first proposals are made for future work on SyRePOE. Then the final conclusions are presented.

A. Future work

Due to the explicit definition of certain concepts in the ontology, SyRePOE has great potential for integration into other ontology based systems, like Knowrob or PMK. A proposal for integration in PMK was already made in Section III.

Future research could also expand SyRePOE itself, e.g. to allow it to track orientations of objects. It currently assumes objects are approximately facing forward. In that case pushing objects in their center will correct slight deviations from this orientation, negating the need to track orientation. To expand applicability of SyRePOE and assure objects on the shelf are facing forward, orientations need to be tracked, considered and corrected.

Lastly, to deploy SyRePOE in a real store, the ontology needs information about the shelves and products it could encounter in the store. To aid the generation of such store specific ontologies, a converter could be made which takes the necessary information from the store's or franchise's database and asserts it to the ontology.

B. Conclusion

In this work the SyRePOE system has been presented. The logic used to infer presence of objects which cannot directly be detected in environments which are physically and visually accessible from a single side has been explained. SyRePOE has been evaluated based on related literature. Its performance has been shown both in simulation and on a real robot. Proposals have been made for future work on SyRePOE, including suggestions for incorporating it in a higher level system and expanding its own capabilities.

REFERENCES

[1] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on robotics*, vol. 33, no. 2, pp. 372–389, 2016.

[2] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL-the planning domain definition language," 1998.

[3] G. H. Lim, I. H. Suh, and H. Suh, "Ontology-based unified robot knowledge for service robots in indoor environments," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 3, pp. 492–509, 2010.

[4] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels, "Knowrob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 512–519.

[5] M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Marton, "Robosherlock: Unstructured information processing for robot perception," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1549–1556.

[6] M. Tenorth and M. Beetz, "A unified representation for reasoning about robot actions, processes, and their effects on objects," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1351–1358.

[7] D. Beßler, M. Pomarlan, and M. Beetz, "OWL-enabled assembly planning for robotic agents," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 1684–1692.

[8] M. Tenorth, G. Bartels, and M. Beetz, "Knowledge-based specification of robot motions." in *ECAI*, 2014, pp. 873–878.

[9] J. Winkler, F. Bálint-Benczédi, T. Wiedemeyer, M. Beetz, N. Vaskevicius, C. A. Mueller, T. Fromm, and A. Birk, "Knowledge-enabled robotic agents for shelf replenishment in cluttered retail environments," *arXiv preprint arXiv:1605.04177*, 2016.

[10] M. Diab, A. Akbari, M. Ud Din, and J. Rosell, "PMK — a knowledge processing framework for autonomous robotics perception and manipulation," *Sensors*, vol. 19, no. 5, p. 1166, 2019.

[11] I. Niles and A. Pease, "Towards a standard upper ontology," in *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, 2001, pp. 2–9.

[12] M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. Bateman, and M. Beetz, "An ontology for failure interpretation in automated planning and execution," in *Iberian Robotics conference*. Springer, 2019, pp. 381–390.

[13] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari, "Wonderweb deliverable d18 ontology library (final)," *ICT project*, vol. 33052, p. 31, 2003.

[14] J. Hoffmann, "FF: The fast-forward planning system," *AI magazine*, vol. 22, no. 3, pp. 57–57, 2001.

[15] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas *et al.*, "A review and comparison of ontology-based approaches to robot autonomy," *The Knowledge Engineering Review*, vol. 34, 2019.

[16] D. L. McGuinness and F. Van Harmelen, "OWL web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[17] M. A. Musen, "The protégé project: a look back and a look forward," *AI matters*, vol. 1, no. 4, pp. 4–12, 2015.

[18] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "Swi-prolog," *arXiv preprint arXiv:1011.5332*, 2010.

[19] M. T. Turvey, "Affordances and prospective control: An outline of the ontology," *Ecological psychology*, vol. 4, no. 3, pp. 173–187, 1992.

[20] J. Ortmann and W. Kuhn, "Affordances as qualities." in *FOIS*, 2010, pp. 117–130.

[21] L. A. Moralez, "Affordance ontology: towards a unified description of affordances as events," *Res Cogitans*, vol. 7, no. 1, pp. 35–43, 2016.

[22] M. Cashmore, M. Fox, D. Long, and et al., "Rosplan: Planning in the robot operating system," in *International Conference on Automated Planning and Scheduling*. Association for the Advancement of Artificial Intelligence, 2015, pp. 333–341.

[23] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[24] C. Pezzato, R. Ferrari, and C. H. Corbato, "A novel adaptive controller for robot manipulators based on active inference," *Robotics and Automation Letters*, vol. 5, no. 2, pp. 2973–2980, 2020.

[25] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and vision Computing*, vol. 76, pp. 38–47, 2018.

Symbolic reasoning about unseen objects from multimodal sensory feedback for manipulation

A. N. Helsloot

Technische Universiteit Delft

TU Delft

Symbolic reasoning about unseen objects from multimodal sensory feedback for manipulation

by

A. N. Helsloot

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday December 11, 2020 at 10:00 AM.

Student number:	4348184	
Project duration:	April 6, 2020 – December 11, 2020	
Thesis committee:	Dr. ir. J. Kober,	COR, TU Delft
	Dr. ir. C. Hernandez Corbato,	COR, TU Delft
	Ir. M. Imre,	COR, TU Delft
	Dr. ir. J. Sijs,	DCSC, TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

This document, together with the paper by the title *Symbolic reasoning about unseen objects from multi-modal sensory feedback for manipulation*, authored by Arthur N. Helsloot, Mert Imre, Carlos Hernandez Corbato, and Jens Kober, forms my graduation thesis for the double degree of Master of Science in Systems & Control and Mechanical Engineering, track Biomechanical Design, at the Delft University of Technology. The mentioned paper has been submitted to the International Conference on Robotics and Automation (ICRA) 2021.

I would like to thank my daily supervisor, Mert Imre, for his feedback and support, both mental and technical. Additionally I extend my gratitude to Carlos Hernandez Corbato and Jens Kober for their feedback and timely course corrections. Lastly, I would like to thank everybody from AIRLab Delft for the brainstorm sessions and available office space and test facilities.

A. N. Helsloot
Delft, November 2020

Contents

1 Introduction

1.1. Document buildup

This document consists of appendices to the paper by the same title and author as this thesis. It is arranged in chapters which each form an appendix to the section of the paper with the same number. The only exception to this is the conclusion in Chapter 6, which concludes the entire thesis. It is recommended to first read the paper in full before consulting this document for further details.

1.2. AIRLab Delft

This study is performed in collaboration with AIRLab Delft. AIRLab stand for Artificial Intelligence for Retail Lab and is a collaboration between Delft University of Technology and Ahold Delhaize. The Lab aims to research and develop smart robotic solutions for use in retail environments. The Lab has a testing facility and robot at its disposal. The robot consists of a Franka Emika Panda robot arm mounted on top of a Clearpath Boxer navigation robot. It is shown in Figure 1.1. This is also the robot used in this thesis.

Figure 1.1: Photo of the AIRLab robot.

1.3. Research question

The main question addressed in this thesis is:

Does reasoning over a symbolic world model allow for high situational awareness in robotic task planning with fully occluded objects?

Here situational awareness is considered higher when the poses of more objects in the environment are known relative to the number of objects present in the environment. This situational awareness is often limited by occlusions, especially in environments which are both physically and visually accessible from only a single side. Simple examples of such environments are shelves in retail stores or cupboards in kitchens. When performing a task in such an environment, like stocking a retail shelf, it is important to realise what is going on on the shelf to select an appropriate next action to perform and to know when the goal has been accomplished.

This problem is addressed in this thesis using a symbolic reasoning approach. Motivation for this approach is supplied in Section 1.5. The running example task used is the stocking of a retail shelf. The complete system set up for this thesis is designed such that it can:

- fully stock a shelf where the objects initially on the shelf are in a grid pose (as defined in Table 1.1), independent of where in the grid these objects are,
- reason about whether the shelf is completely stocked or not, and
- easily be applied to different shelves, stores, objects or environments.

1.4. Assumptions

Throughout this work a couple of assumptions are made. Some are relevant to the entire SyRePOE system, others only to the case study discussed in Chapter 4. Here an overview of all made assumptions relevant to the entire SyRePOE system is given. These assumptions state the boundary conditions of the type of problem SyRePOE is expected to handle.

- The environment is quasi-static. This means that dynamic effects can be neglected. In other words, when observing the environment at any time, it must be in equilibrium, but it does not have to be the same as it was at the last observation.
- All sub-environments are rigid, intact, cuboid, and of known dimensions and location.
- All sub-environments are closed at the back, such that no objects can fall off or out.
- All objects are rigid, intact, cuboid, and of known dimensions.
- Objects do not topple, tilt, rotate, or stack.
- Visual feedback of the sub-environment is available, such that the system receives poses, relative to the sub-environment, of objects in the sub-environment to which line of sight is present from the front of the sub-environment.
- Haptic feedback from performed actions is available, such that the system receives an estimate of the opposing force on the end effector during action execution.

1.5. Motivation

In this thesis a symbolic reasoning approach to object manipulation in environments with unseen objects is taken. An alternative to a symbolic world model could be to keep track of the world model in a physics environment. Such a geometric world model is, however, better suited to contain different types of information from a symbolic world model. For example, to infer presence of an unseen object, the push action could be simulated under different assumptions: once with the unseen object present, once without. The outcomes of the simulations could be compared to reality and the scenario which closest resembles reality can be taken as closest to the truth. This method theoretically allows for equal levels of situational awareness as the symbolic reasoning approach presented in this work, but it requires simulation with a physics engine, which takes a lot of computational effort (both power and time). Such a simulation based method does not scale well with the number of scenarios that need to be considered. This shows inferring presence of unseen objects is quite difficult when using only a geometric world model. Additionally, a geometric world model is very good at containing geometric information, like shapes and poses, but other types of information, like semantic knowledge, is simply not suitable to be contained in a geometric world model.

The presence of unseen objects could alternatively be concluded using an ‘if push got blocked, then line is full’ approach. Then the assumption is made that when the opposing force registered

during a pushing action exceeds a threshold, the space behind the object being pushed, must be filled with other objects. Behaviour trees [7] or Planning Domain Definition Language (PDDL) [13] based approaches can easily capture this type of behaviour. In ideal cases, such a method could allow for similar levels of situational awareness to the system proposed in this thesis. However, they generally have trouble dealing with unknown information. They often assume that anything not known to be true, must be false. This is called the closed world assumption. A symbolic reasoning approach allows to formalise assumptions and to reason over, reflect on, and update its current state using a large collection of knowledge and measurements from a multitude of sources at once, without the need for the closed world assumption. This way high situational awareness can still be attained with less predefined assumptions.

1.6. Terminology

Before diving into the details of this thesis, a couple of often used terms should be explained to prevent confusion. Table 1.1 can be used as a reference throughout reading this work.

Table 1.1: Terminology used throughout this work

Term	Meaning
SyRePOE	Pronounce: **sir**-uhp. Stands for Symbolic Reasoning for Partially Observable Environments. This is the name of the system proposed in this thesis.
sub-environment	A structure, like a shelf or cupboard, in which objects can be found.
object	A thing in the workspace of the robot, which the robot can pick, place, push or otherwise interact with.
manipulability	A property of an object which indicates whether or not the object can be moved backwards from its current position.
world model	The description of the environment the robot uses. This world model is updated through measurements and used for decision making and action planning.
pose of a sub-environment	Indicates where in the world the robot can find this sub-environment. This is expressed in terms of an X- and Y-coordinate to define the position of the front left corner of the sub-environment on the floor plan, a Θ-coordinate to define its orientation on the floor plan and a Z-coordinate to define the height of the sub-environment above the floor.
pose of an object	Indicates where in the sub-environment, relative to the front left corner of the sub-environment, the geometrical centre of an object is located.It uses an X-coordinate to indicate the sideways position and a Y-coordinate to define the distance to the front of the sub-environment.
grid configuration	The configuration of objects in the sub-environment that has all objects standing in non-overlapping lines behind each other.
grid pose	A pose in the sub-environment such that the object fits in the grid configuration.
row	One line in the grid configuration reaching from the front to the back of the sub-environment.

2 Background

2.1. What is an ontology?

In engineering an ontology is a representation of domain knowledge. It organises concepts into classes and relates these concepts to each other through properties. Ontologies contain four main constructs of representing knowledge: classes, instances, object properties and data properties. The ontologies discussed in this work are all implemented in the Web Ontology Language (OWL) [14].

2.1.1. Classes and instances

Classes represent concepts or groups of concepts. They can be seen as sets. Different sets can overlap, contain or be disjoint with each other and so can classes. An example of a class is 'Dutch people'. Another class could contain this class (e.g. 'people'), could partially overlap this class (e.g. 'people who speak Dutch') or could be disjoint with this class (e.g. 'cans of cola'). When a class contains another class, they can be represented using a superclass/subclass relation. For example, since any Dutch person is by definition a person, 'Dutch people' can be represented as a subclass of 'people'.

Instances represent individuals. An instance belongs to at least one class, but can always belong to multiple classes at the same time (just think of the superclass/subclass relation). An example of an instance can be your friend 'Stefan'. This indicates only a single person and is therefore best represented by an instance, for example of the class 'people'.

2.1.2. Object properties and data properties

Properties can tell you something about individuals. Object properties can link instances to each other, while data properties can link instances to other data, like numbers. An example of an object property can be the property 'friend of'. This property can link the individual 'Jasper' to the individual 'Stefan'. An example of a data property can be 'has age', which could link individual 'Jasper' to the number 27.

Properties can be used to define classes through so called class restrictions. For example, the subclass 'Dutch people', of superclass 'people', can be defined as the set of all individuals of class 'people' with a 'has citizenship' property linking it to Dutch. Dutch could for example be represented as an individual of another class or a string, but that is irrelevant for this example. This way the ontology knows that any individual that is a person and has Dutch citizenship is called a Dutch person.

2.1.3. Ontology triples and the open world assumption

An ontology is often represented as a set of triples. Every triple consists of a subject, a property and an object, usually presented in that order. An example of such a triple is ('Stefan', 'type', 'people'), indicating that 'Stefan' is an individual of class 'people'. In case such a triple exists in the ontology, it is considered a known fact. If the triple does not exist, it is assumed to be unknown; possibly true and possibly false. This assumption is called the open world assumption. Here is an example of how this can be used: Let's say there is an object, but it is unknown where exactly it is. The object is an instance of class 'Object' and has an object property linking it to an instance of class 'Pose'. This instance of class 'Pose', however, has no data properties linking it to specific numbers. This way it can be represented that the object is known to have a pose, but what this pose is, is unknown.

2.2. Related Work

This section starts with a discussion of specific ontologies related to robotic manipulation from literature and their content and expressivity. It continues with a discussion of robotic frameworks from literature that utilise such ontologies for robotic manipulation or related tasks.

2.2.1. Related ontologies

The first ontology from literature that should be discussed is the Suggested Upper Merged Ontology (SUMO) [16]. This ontology is not specifically aimed at robotics, but is the largest publicly available domain ontology currently in existence. It forms the basis for many other ontologies. By modelling other ontologies under such a standard ontology, systems that use these ontologies can easily be integrated with each other, since they agree on the definitions of standard terms. This promotes extensibility, which is a much desired trait of systems in robotics.

Literature also offers ontologies aimed more specifically at robotics applications. The Core Ontology for Robotics and Automation (CORA) [11], based on SUMO, aims to define basic concepts relating to robotics. It provides general classes like 'Robot', 'Artifact' and 'Quantity'. Using such classes, a system could be given a more high level understanding of the type of things it is dealing with.

The Knowrob framework [3] is a knowledge representation system that is specifically designed for robotic manipulation at a human scale. Its accompanying ontology allows for more specific definitions than CORA or SUMO through classes like 'Scissors', 'LeftArm' and 'Computational predicate'. The ontology is, however, way less structured than CORA and is meant to be used specifically with the rest of the framework. It does not follow any standardisation, which limits its extensibility.

The Perception and Manipulation Knowledge (PMK) [9] ontology is designed to supply robotic systems with knowledge and understanding of problems in human occupied environments. Just like CORA, it has been modelled under SUMO [16]. This ontology is discussed in more detail in the accompanying paper, just as the failure ontology [10] proposed by the same authors.

These ontologies are all fine candidates to be expanded to the needs for this work, but for all of them holds that the representation of objects and sub-environments needs more detail to truly capture the relevant aspects. These aspects are discussed in Section 3.2, together with the introduction of the newly designed ontology. When applying the solution presented in this work to a more general system, the new ontology could be integrated with one of the ontologies mentioned above. Such potential integration is discussed in more detail in Section 3.2.3.

2.2.2. Related frameworks

In literature multiple frameworks can be found that apply ontologies and symbolic reasoning for knowledge processing in robotics.

The Knowrob [3] framework, also discussed in the accompanying paper, has formed the basis for many projects coming from the Institute for Artificial Intelligence (IAI) at the University of Bremen [1]. This includes RoboSherlock [2], a perception system aimed at recognising and reasoning about every day objects, and several task and action planning solutions [4, 25, 26]. However, Knowrob does not follow documented standardisation in the concept definitions of its ontology. This limits its potential for integration in larger systems.

Knowrob has been utilised by Winkler et al. [28] in a retail environment. They use knowledge about the shelves and objects to aid perception and plan manipulation of partially occluded objects. They show their knowledge based perception system performs well with clutter and partial occlusions. They also show their manipulation plans include correctly removing obstructing objects before attempting to grasp an occluded goal object. For planning they use an adapted A* planner with several different predefined actions. They show the planner most of the time finds a plan in less than 10 seconds, but in cases of multiple obstructing or irregular objects, it can take up to 2 minutes. Their proposed system is only suitable for situations with partial occlusions, still allowing for full observability of the scene. Fully occluded objects are not considered.

An interesting framework worth mentioning is the Perception and Manipulation Knowledge (PMK) framework is presented by Diab et al. [9]. This framework has also been extensively discussed in the accompanying paper.

All the systems mentioned so far strongly rely on observations and do not consider objects that are not directly detected. Therefore, they would fail in environments where not all objects are observable.

3

The SyRePOE system

This chapter explains the SyRePOE system and its components. It starts with an explanation of the general pipeline for the system is given in Section 3.1. Next the SyRePOE ontology is presented in Section 3.2, detailing its content, the rationale behind the design choices, how to extend the ontology, and a setup for integration of the SyRePOE ontology in other ontology based systems. Section 3.3 explains more about the reasoning capabilities of SyRePOE and the logic used for them. Section 3.4 shows how SyRePOE handles task planning. This chapter is closed by Section 3.5, which evaluates SyRePOE on the scales of autonomy recently proposed by Olivares-Alarcos et al. [19].

3.1. Concept

An overview of the taken approach is given in Figure 3.1, also shown in Figure 2 of the accompanying paper. This figure shows the components involved in many ontology based systems. In the bottom the physical world is shown, which is acted upon by the robot and measured by sensors. The sensor data is filtered and preprocessed by the perception module. The incoming information is interpreted by the reasoner, using contextual information from the ontology. The ontology is then updated based on the drawn conclusions. Next the planning module picks the next action and its parameters from a predefined list of parameterised actions based on the drawn conclusions from the reasoner and/or contextual information from the ontology. Lastly, the robot performs that action on the physical world.

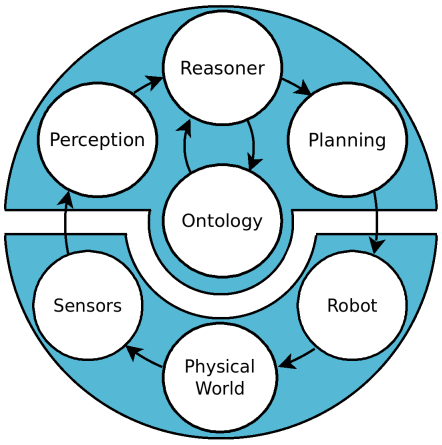


Figure 3.1: Diagram showing the ontology and reasoner approach used in SyRePOE. The top section represents software, the bottom section hardware.

SyRePOE is implemented using the Robot Operating System (ROS). The ontology is implemented in OWL [14], making use of Protégé ontology editor [15]. The reasoner is implemented using SWI-Prolog [27]. The communication between the ontology, Prolog and ROS is realised utilising the ROSProlog package from the Knowrob framework [3]. Action planning is done by passing a Planning Domain

Definition Language (PDDL) [13] description of the world model to the Metric Fast-Forward planner [12] through the ROSPlan framework [5].

3.1.1. Pipeline diagram

A more detailed pipeline of SyRePOE is shown in Figure 3.2. Here the same pipeline is shown in three different reference frameworks: a classic control loop, Sense-Plan-Act and MAKE-K. It is important to note that there is only a single architecture, but Figure 3.2 expresses it in three different reference frameworks to make it easier to understand for readers of different backgrounds.

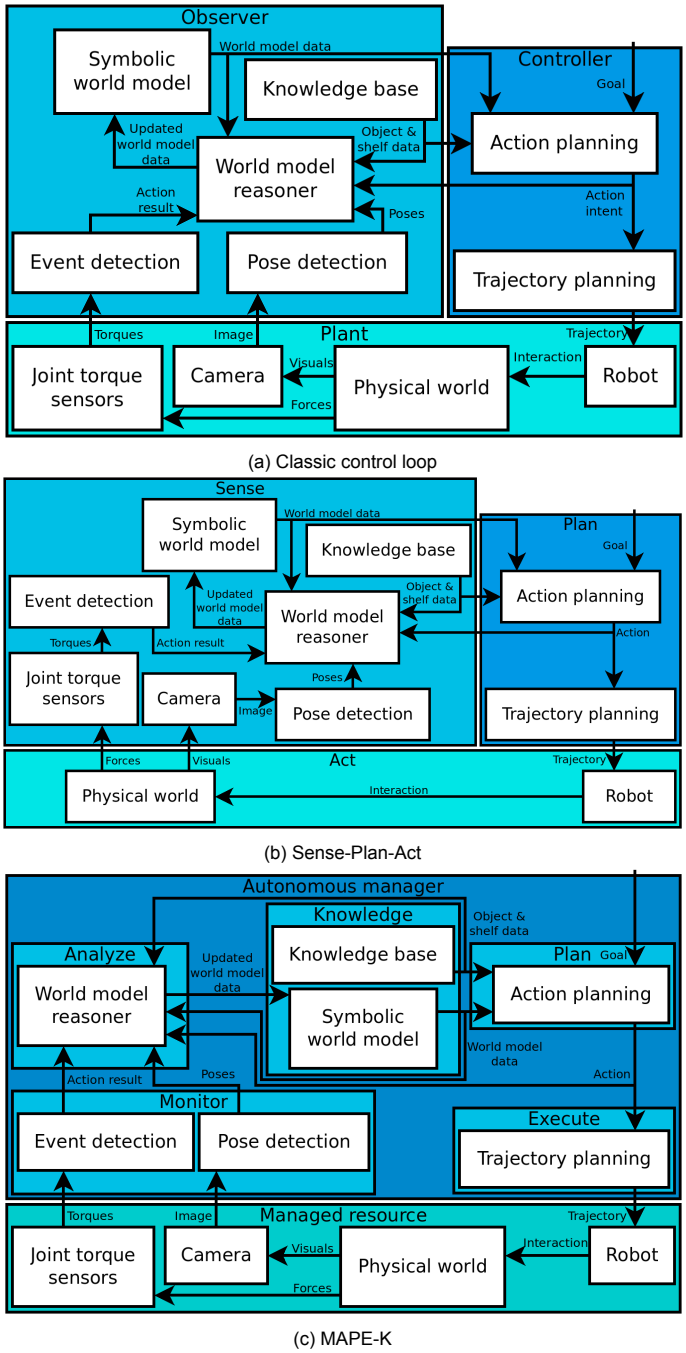


Figure 3.2: Schematic representation of the SyRePOE pipeline in different frameworks.

In these diagrams the blocks named ‘Camera’, ‘Physical world’ and ‘Robot’ are quite self explanatory. The ‘Pose detection’ takes an image from the camera and finds the poses of visible objects. This

module is part of the perception module in Figure 3.1. The ‘Joint torque sensors’ measure the torques applied to each joint of the robot. The ‘Event detection’ then analyses these torques to detect when the opposing force to the robot is too big. How exactly this is done is explained in Sections 4.3.2 and 4.4.1 for the simulated and real environments respectively. The ‘World model reasoner’ takes the data supplied by the pose- and event detection and unifies it with the currently believed world model, which is represented by the ‘Symbolic world model’ and maintained in the ontology. The world model, together with some static knowledge about the objects and the environment from the ‘Knowledge base’, is passed to ‘Action planning’. Here an action to be executed is selected. For the case-study discussed later in Chapter 4 it can choose between placing a new object in the front position of any of the rows or pushing an object in the front position of any of the rows backwards by a distance equal to the depth of the object. How the action selection works in detail is discussed in Section 3.4. The selected action is passed to ‘Trajectory planning’, where it is translated to joint commands. After the action is executed, the action is passed to the ‘World model reasoner’, which reasons about the expected action effect as explained in Section III-B of the accompanying paper, and updates the world model accordingly. By integrating the action effect separately from the observations, the system can differentiate between the effect of actions it performed itself and the effect of actions other agents might have performed in between action execution and observation. This makes for a more robust system that is less easily fooled into drawing wrong conclusions about action results, action effects, and the world model.

3.2. The SyRePOE ontology

The SyRePOE ontology is meant to maintain the world model of objects in a limited sub-environment in a logical and comprehensive manner. All classes and properties unique to this ontology are shown in Table 3.1. Since shelves are prime examples of sub-environments which are only observable and accessible from a single side, from here on shelves are the only sub-environments considered. The class ‘Shelf’ can be considered a subclass of sub-environments. The approach is generalisable to other sub-environments as well. How this ontology is used to represent objects is explained in Section III-A of the accompanying paper.

Table 3.1: Ontology content

Classes	Object properties	Data properties
• Shelf • Object • ObjectType • Pose • Manipulability • Detection • DetectionMethod • TimePoint	• onShelf • containsType • hasType • hasPose • hasManipulability • hasDetection • hasDetectionMethod • hasTimePoint	• hasDepth • hasWidth • hasCapacity • hasInterRowDistance • isCurrentlyWorked • hasBackManipulability • hasXCoordinate • hasYCoordinate • hasZCoordinate • hasThetaCoordinate • hasUnixTime

3.2.1. Ontology design

The ontology was developed using the Ontology 101 methodology [18]. Following this methodology, first the competency questions for the ontology were constructed. These questions are listed below.

- What is the pose of object X?
- What is the pose of shelf X?
- What shelf does object X belong to?
- What are the dimensions of object X?
- What are the dimensions of shelf X?
- How many objects should fit on shelf X?
- Can object X be moved further back?

Next, a few existing ontologies, namely PMK [9], Knowrob [3] and CORA [11], were checked to see which of the competency questions can already be answered by them. These ontologies have been discussed in more detail in Section 2.2.1. Concepts like objects, sub-environments, poses and time points can certainly be expressed in these ontologies. However, these ontologies lack the detailed expressivity needed to answer all competency questions listed above. The definitions of most existing classes are not specific enough. Most needed classes would have to be newly defined subclasses of existing classes in these ontologies. This would mean that to adapt these ontologies such that they can answer all competency questions would be at least equally complicated as designing a new ontology. This is why a new ontology was designed and attention was paid to potential integration of this new ontology into existing ontologies, as further discussed in Section 3.2.3.

The most important concepts the new ontology has to represent are shelves and objects. Since every shelf is unique, but they are all shelves, it is an easy choice to make ‘Shelf’ a class, allowing individual shelves to be individuals of the class ‘Shelf’. It is known that there can be multiple objects on a single shelf, which each are their own individual. It therefore makes sense to create a class ‘Object’ and have each object on a shelf be an individual of that class. Individuals of the class ‘Object’ can be connected to individuals of class ‘Shelf’ using the ‘onShelf’ property to signify that this individual object is on that specific shelf.

Many objects are similarly shaped (they are for example all cartons of milk), but they differ completely from other groups of objects (which might for example be tubes of toothpaste). In order to not have to link spatial dimensions to every individual of the class ‘Object’, it would be useful to have something that can represent a type of object, so to show that an object can be a carton of milk and all cartons of milk have the same dimensions. (Of course larger and smaller cartons of milk exist, but a type can be created for each variation.) To this cause a class ‘ObjectType’ was made. In this class individuals can be made, like ‘milk’, with data properties giving it a width, a depth and a height. Each individual of class ‘Object’ can be linked to an individual of class ‘ObjectType’ through the ‘hasType’ property. Additionally, when a certain shelf contains a certain type of object, the individual of class ‘Shelf’ can be linked to an individual of class ‘ObjectType’ using the ‘containsType’ property.

Alternatively object types could have been represented as subclasses of class ‘Object’. Then each such subclass would have class restrictions to say that all members of this subclass necessarily have a specific width, depth and height. This implementation was rejected, because retrieving properties from the ontology using Prolog is more straight forward than retrieving class restrictions.

3.2.2. Adding new shelves and object types to the ontology

To deploy SyRePOE in a real environment, like a retail store, the ontology needs to be initialised with all sub-environments and object types it needs to interact with. In the example of a retail store, each store would have their own ontology describing the types of products on offer and the physical layout of the store. This will differ per store and will change over time. It is therefore important to know how to update this information.

When a new product becomes available in a store, its object type needs to be added to the ontology. This object type needs to be given a name, a depth, a width and a height. This new product also needs a shelf to be presented on, so a new shelf must be added to the ontology. This shelf needs to be given a name, a location in the store (X-, Y-, Z- and Θ-coordinates), a width, a depth, a capacity, an inter-row distance, and the object type that belongs to the shelf.

To add this information by hand for a single new object type or shelf is not much of an effort, but creating an ontology from scratch for an entire store would be too much to ask. Luckily many stores have digital databases that store the information mentioned above for the entire store already. Since both such a database and the ontology are highly structured, a simple program could be written to extract the necessary information from the database and generate the ontology automatically. This program could output an XML style OWL file directly, or it could use Prolog predicates to assert new instances of classes ‘ObjectType’ and ‘Shelf’, together with their properties, to a knowledge base and then export the knowledge base as an ontology.

In large stores with a lot of shelves, the knowledge base will contain thousands of object types and thousands of shelves. Reasoning over so many facts, could potentially take a long time. Luckily, the store is compartmentalised, and so can be the reasoning. When the system has finished stocking a shelf, it should assume that customers will interact with the objects on the shelf before the system interacts with this same shelf again. Therefore the information the system has about the objects on the

shelf will be outdated. Additionally, when stocking one shelf, the information about objects on another shelf is not needed for reasoning over the objects on this shelf. Therefore the information the system has about the objects on other shelves is irrelevant. Since there is no need to remember information that is both outdated and irrelevant, the system only needs to remember information about objects on the shelf it is currently working on. When the system would return to a previously visited shelf, it will simply restart without assumptions about the amount or location of objects on the shelf. The number of objects on a single shelf will only in very rare cases become large, for example over one hundred objects. This shows that the total number of facts the system has to reason about on each occasion will be limited.

3.2.3. Ontology integration

The SyRePOE ontology developed for this work is suitable to be integrated into a higher level ontology. This section goes deeper into how such integration could be done, what this would mean for the system presented here and what possibilities this creates for the combined system.

The SyRePOE ontology has been modelled under PMK [9]. This means that all classes in the SyRePOE ontology can be considered subclasses of classes in the PMK ontology. Table 3.2 shows the classes of the SyRePOE ontology with a possible interpretation of it in PMK, including the accompanying description logic definition. Since PMK has been modelled under SUMO [16], SyRePOE is indirectly modelled under SUMO as well. This allows for the integration with any other ontology modelled under SUMO, quickly expanding the system's knowledge to other fields, like linguistics [8, 17, 23], scientific experimentation [24] and system failure analysis [10].

Table 3.2: SyRePOE ontology classes modelled under PMK [9]. DL stands for Description Logic.

Class	PMK description	DL description
Object	An Artifact that has a Manipulability and a Pose	$Object \subseteq Artifact \wedge$ $Object \subseteq \{x \mid \exists m \in Manipulability, (x,m) : hasManipulability\} \wedge$ $Object \subseteq \{x \mid \exists p \in Pose, (x,p) : hasPose\}$
ObjectType	An Artifact that has spatial dimensions related to it	$ObjectType \subseteq Artifact \wedge$ $ObjectType \subseteq \{x \mid (\exists h \in \mathbb{R})$ $(\exists d \in \mathbb{R})(\exists w \in \mathbb{R})$ $[(x,h) : hasHeight \wedge (x,d) : hasDepth \wedge (x,w) : hasWidth]\}$
Manipulability	A Quantity	$Manipulability \subseteq Quantity$
Pose	A Quantity-Aggregation	$Pose \subseteq QuantityAggregation$
Detection	An Attribute that has a DetectionMethod and a TimePoint	$Detection \subseteq Attribute \wedge$ $Detection \subseteq \{x \mid \exists m \in DetectionMethod, (x,m) : hasDetectionMethod\} \wedge$ $Detection \subseteq \{x \mid \exists t \in TimePoint, (x,t) : hasTimePoint\}$
DetectionMethod	A DeviceGroup	$DetectionMethod \subseteq DeviceGroup$
TimePoint	A Quantity	$TimePoint \subseteq Quantity$
Shelf	A Region that has an ObjectType and a Pose	$Shelf \subseteq Region \wedge$ $Shelf \subseteq \{x \mid \exists y \in ObjectType, (x,y) : containsType\} \wedge$ $Shelf \subseteq \{x \mid \exists p \in Pose, (x,p) : hasPose\}$

By integrating SyRePOE into PMK, the strengths of the two frameworks can be combined. This would allow SyRePOE to benefit from the strong integration with Task And Motion Planning (TAMP), offered by PMK, as well as the specific specification of individual sensors and signal processing algorithms in the ontology, which allows the system to answer queries like *What sensors are available to detect the shape of this object?* At the same time PMK would gain the ability to utilise haptic feedback for registering manipulability of objects. This registration allows the reasoner to draw conclusions about

the presence of objects which it cannot directly see, enhancing the world model in visually occluded areas.

As an alternative to the integration with PMK, SyRePOE could also be integrated in Knowrob [3]. This would give SyRePOE access to Knowrob's functionalities, like episodic memories, learning and physics engine based simulation. Knowrob, however, does not follow any standardisation of terms, like PMK does. This means integrating with the Knowrob ontology would make it more difficult to integrate with SUMO.

3.3. The SyRePOE reasoner

The SyRePOE reasoner is responsible for reasoning over the current world model and concluding as much additional information as possible. Its main functions and the logic used to accomplish these are explained in Section III-B of the accompanying paper.

A Prolog module for spatial relations was created to allow the reasoner to check relative positions of objects compared to each other and to the sides and back of their sub-environment. This module forms the basis needed to propagate manipulability through the objects.

To deal with measurement errors of the poses of the perceived objects, a pose error margin is built into the reasoner. When an object is detected in a pose within that margin of an already known object, the newly detected object is considered to be the same as the known objects. This margin is set to an empirically determined ratio of the object dimensions. For example, in simulation objects were allowed to deviate by 20% the object width sideways and 20% the object depth backwards.

Besides the features explained in the accompanying paper, the reasoner also uses a combination of two cases to decide if it is done with the sub-environment it is currently working on. The logic used for this is show in Eq. (3.1). Here c represents the number of objects the shelf can maximally contain and the $in\_front(O)$ statement checks if the Y-coordinate of object O is smaller than one object depth; in that case the object is considered to be in the front of the sub-environment. The first of these cases checks whether the goal in the sub-environment is accomplished. In the task of stocking a retail shelf, this checks if the number of objects on the shelf according to the world model matches the capacity of the shelf. The second predicate checks if the system still has actions available capable of changing the current state of the world. In the task of stocking a retail shelf this predicate checks if the front of the shelf is completely full and all objects there are registered as not manipulable. Ideally this situation should only occur if the shelf is full. This predicate is build as a fail safe, since in case this predicate succeeds, the system cannot place a new object, nor can it move a object back to make space for a new object.

$$|Object| = c \vee (\forall O \in Object)\big(\neg in\_front(O) \vee \neg manipulable(O)\big) \rightarrow shelf\_done \tag{3.1}$$

3.4. Planning

To perform a task, SyRePOE needs to select actions to perform based on the world model. For the task of stocking a retail shelf, planning is separated into two parts. The first part is performed by the reasoner. The reasoner can suggest correctional actions to arrange objects in a sub-environment into lines behind each other. The logic used for this rule is shown in Eq. (3.2). In this equation w_O is the width of an object and d_i is the distance between rows of objects. This rule checks for each known object whether it is in such a line. If an object is not, a sideways push is proposed to move the object from its current X-coordinate X_s to the closest X-coordinate where it would be in a row X_e. If all objects are in a row, the problem translates to a simpler problem, such that a graph-search-based planner can more quickly and efficiently decide on the next action to take.

$$\begin{aligned}(\exists O \in Object)(\exists X_s \in \mathbb{R})(\exists Y_s \in \mathbb{R})\Big(&obj\_at(O, X_s, Y_s) \wedge \\ &(\forall N \in \mathbb{N}_0)(N + 0.5) \cdot (w_O + d_i) \neq X_s \wedge \\ &(\exists X_e \in \mathbb{R})\big((\exists M \in \mathbb{N}_0)(M + 0.5) \cdot (w_O + d_i) = X_e \wedge \\ &X_s - X_e \leq 0.5 \cdot (w_O + d_i)\big)\Big) \rightarrow \\ &sideways\_push(O, X_e - X_s)\end{aligned} \tag{3.2}$$

Even though the reasoner has been shown to correctly suggest such sideways pushes, this feature has not been tested in the case studies discussed in Chapter 4. The design of the gripper made it difficult to successfully apply sideways pushes without affecting other objects on the shelf. Another gripper or a differently designed motion of the robot might resolve this problem, but is considered outside the scope of this thesis.

For further planning, the world model is translated to a simpler description in Planning Domain Definition Language (PDDL) [13]. The PDDL problem is supplied with a static list of actions to choose from. This list consists of two actions. The first, a place action, picks a new object form a fixed pose on the base of the robot, representing the supply, and places it at the front of a row on the shelf. The second action, a push action, pushes the object in the front of a row backwards by one object depth. These two actions can be applied to any row. The logic used for the definition of the place and push actions in the PDDL problem is presented in Eq. (3.3) and Eq. (3.4) respectively, where r is a parameter representing the specific row the action would be applied to and r_{max} is a number representing the maximum number of objects a row may contain. In this logic the $place(r)$ statement is true if the place action is applied to row r, and similarly for $push(r)$ and the push action. The other statements on the left side of the equations are preconditions to the action. All statements on the right side of the equations are postconditions. It is important to note that these rules indicate the expected result of the actions assuming they are successfully applied. After the action is executed and the action result (whether the action succeeded or failed) is known, the expected action effect and the action result are passed to the reasoner and processed as described in Section III-B of the accompanying paper. The goal state for the action planner is to have each row of objects on the shelf filled to capacity. The ROSPlan framework [5] is used to pass this problem description to the Metric-FF planner [12]. The Metric-FF planner is used because it is available is ROSPlan and capable of using numeric fluents.

$$\begin{aligned}&\big(place(r) \wedge \neg front\_occupied(r) \wedge (\exists N \in \mathbb{N}_0) number\_of\_objects(r, N)\big) \rightarrow \\ &\big(front\_occupied(r) \wedge number\_of\_objects(r, N+1)\big)\end{aligned} \tag{3.3}$$

$$\begin{aligned}&\Big(push(r) \wedge front\_occupied(r) \wedge \neg front\_blocked(r) \wedge \\ &\big(\exists N \in \mathbb{N}_0\big)(number\_of\_objects(r, N) \wedge N < r_{max})\Big) \rightarrow \\ &\neg front\_occupied(r)\end{aligned} \tag{3.4}$$

Combined this means that the planning is partially handled by the PDDL problem and partially by the SyRePOE reasoner. This is because the reasoner is capable of efficiently suggesting an action that could be performed based on the poses of each individual object. However, it cannot make a goal oriented plan of actions. By asking the reasoner for an action that fixes something that would be a problem for the PDDL solver to handle (like an object between rows), the reasoner can get the system closer to a state from which the PDDL solver can handle it. The Metric-FF planner builds a complete plan of actions that is guaranteed to get the system to its goal state. Implementing each individual object in the sub-environment as an individual in PDDL, however, would increase planning times to enormous amounts compared to only keeping track of the number of objects in each row in the sub-environment. Therefore planning is shared between the reasoner and a PDDL solver to ensure the goal state is approached, while still having reasonable planning time.

While planning, PDDL uses the closed world assumption, meaning that any fact it has not been told that is true, it considers to be false. This contradicts the open world assumption made in the ontology. However, for planning a single action, the closed world assumption is acceptable to be made. Let's say there is a fact F and an action A and to achieve the goal (amongst other facts) F needs to be true. Let's say that if F is false, A brings the system closer to the goal and therefore to making F true. If F is true, A does not do anything. An example of such an action is applying a push action to an object of which it is unknown if it is blocked. This means that if F is false, A gets the system closer to its goal, but if F is true, A does not change the state of the world. Now, if F is unknown, A is a valuable action, because it either gets the system to a state closer to F being true, or it teaches the system that F was already true. Either way, the system gained something. Therefore, if F is unknown, it can be treated as false during planning. This is only valid when replanning after performing one action, since the remainder of

the plan might not be valid anymore, depending on the outcome of the performed action. Alternatives to PDDL, like RDDL [22], have been proposed that do not use the closed world assumption. These alternatives are, however, less commonly used than PDDL, making them less suitable to be readily integrated in other systems.

To integrate a PDDL planner in SyRePOE, the problem needs to be translated into two PDDL files: a PDDL domain file and a PDDL problem file. The PDDL domain file is constructed to contain as few entities as possible. This is because the number of possible actions to apply rises exponentially with the number of entities known to the planner. This means that the fewer entities are represented in the PDDL files, the faster planning is. The only entities represented by the domain are rows of the sub-environment, which is why objects should not be be in between rows when solving the problem using the PDDL solver. Each row keeps track of the number of objects it contains, whether or not its front spot is occupied, and whether or not the object in its front spot is manipulable. Additionally the maximum number of objects per row and an accumulative action cost are registered. The actions that are available to the planner are a push action that clears the front spot of the row it is applied to, and a place action that occupies the front space of the row it is applied to and increases the number of objects in that row by one. The logic used for these actions is presented in Equations (3.3) and (3.4). All facts and numbers needed for PDDL to assess the preconditions are supplied by the ontology. Both actions increase the accumulative action cost by one.

The PDDL problem file is generated every iteration of the operation cycle of the system. It initialises with the accumulative action cost at zero and registers the number of objects each row can maximally contain. It introduces a row entity for every row the sub-environment is wide. For each row the number of objects in that row, the occupancy state of the front spot of that row, and the manipulability of the object in that front spot are queried from the reasoner. The goal is always the same: for every row, either have the number of objects in the row equal to the row capacity or have the front spot both occupied and blocked. This again reflects the two situations in which the system is done, as shown in Eq. (3.1); either the shelf is full or there are no more actions to perform. The metric used to find the optimal plan is also always the same: minimise the accumulative action cost. Since all actions have a cost of 1, this amounts to the least amount of actions to perform.

3.5. SyRePOE on autonomy scales

Section III-C of the accompanying paper discusses the expressivity and capabilities of SyRePOE on scales introduced by Olivares-Alarcos et al. [19]. Below two tables are given for a complete overview of the aspects of these scales and whether or not they are covered by SyRePOE. The ontology concepts considered in [19] are shown in Table 3.3. The reasoning skills considered in [19] are shown in Table 3.4. Further details of the terms used in Tables 3.3 and 3.4 can be found in [19].

Table 3.3: Overview of relevant concepts for autonomous robotics defined in the SyRePOE ontology. List from [19].

Concept	Defined
Objects	**Yes**
Environment map	**Yes**
Affordance	No
Action	No
Task	No
Activity	No
Behavior	No
Function	No
Plan	No
Method	No
Capability	No
Skill	No
Hardware	No
Software	No
Interaction	No
Communication	No

Table 3.4: Overview of the reasoning skills for autonomous robotics that SyRePOE supports. List from [19].

Reasoning skill	Supported
Recognition and categorisation	No
Decision making and choice	No
Perception and situation assessment	No
Prediction and monitoring	No
Problem solving and planning	No
Reasoning and belief maintenance	**Yes**
Execution and action	**Yes**
Interaction and communication	No
Remembering, reflection, and learning	No

4

Case-Study

This chapter goes over the three stages of validation performed on the SyRePOE system. All utilise the retail shelf stocking scenario. The first, explained in Section 4.2, is a conceptual discretised world. It is solely used to show the reasoning capabilities the system should theoretically have are working correctly. Then, Section 4.3 goes into the tests performed with a simulated robot in a more realistic virtual environment. Lastly, Section 4.4 explains the tests done with a real robot in a mock-up store. Before all that, the assumptions made in the scenario are explained in Section 4.1.

4.1. Assumptions

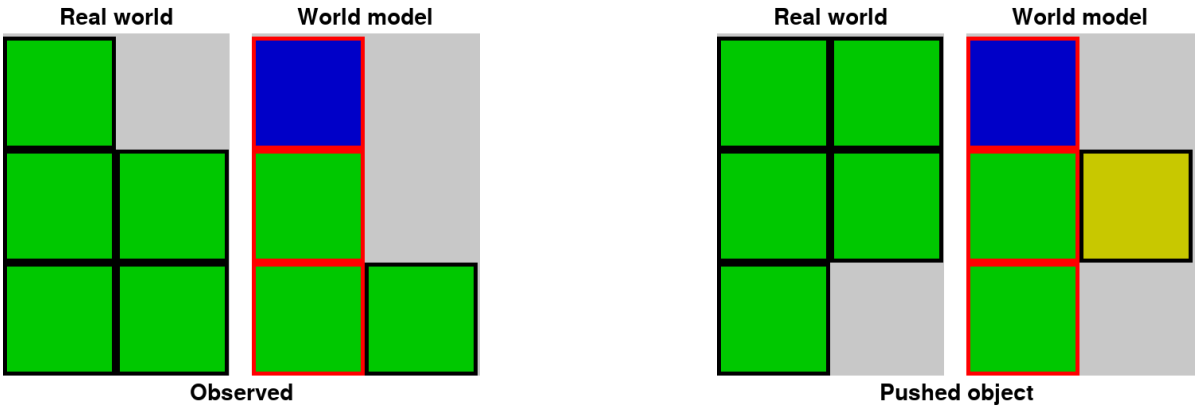
For the case study scenario a few more assumptions have been made on top of those presented in Section 1.4. Whereas the previously presented assumptions are inherent to the entire SyRePOE system, the following assumptions are specific to the conditions of the case study scenario and can more easily be changed. Any problem that would arise from not making the assumptions mentioned below is considered outside the scope of this work.

- The available supply is sufficiently large to fully stock the shelf.
- The supply consists solely of objects belonging on this shelf.
- Goal configuration is a grid.
- Objects that are initially on the shelf are close to a grid pose, deviating no more than one third the object dimension in each direction.
- All objects of the same object type are equivalent.
- The shelf does not contain any objects of types that are not supposed to be there.

4.2. Proof of concept

To provide a proof of the feasibility of the idea, a simple scenario was created. This scenario is visualised in Figure 4.1. In the scenario there is a shelf of two by three objects, divided into a grid. On each spot in the grid there could either be an object present or not. Objects cannot stand in between spots. The system has 4 possible actions to perform: Place an object at the front left, place an object at the front right, push the object on the front left back one spot, and push the object on the front right back one spot. It is always allowed to push a line of multiple objects at once. The system can only see the frontmost objects of each line. A certain number of objects (maximally 6) is on the shelf at the start of the scenario. The system needs to add objects to the shelf until 6 objects are present on the shelf. The system needs to realise when it is done itself. It needs to succeed in this independently of the starting configuration of the objects on the shelf.

In order to accomplish this, the reasoner will have to reason about the transitive property of manipulability (if A is directly in front of B and B cannot be pushed backwards, then A cannot be pushed backwards), the presence of occluded objects, and how many objects are on the shelf, in order to know when the system is done.



(a) Before push action

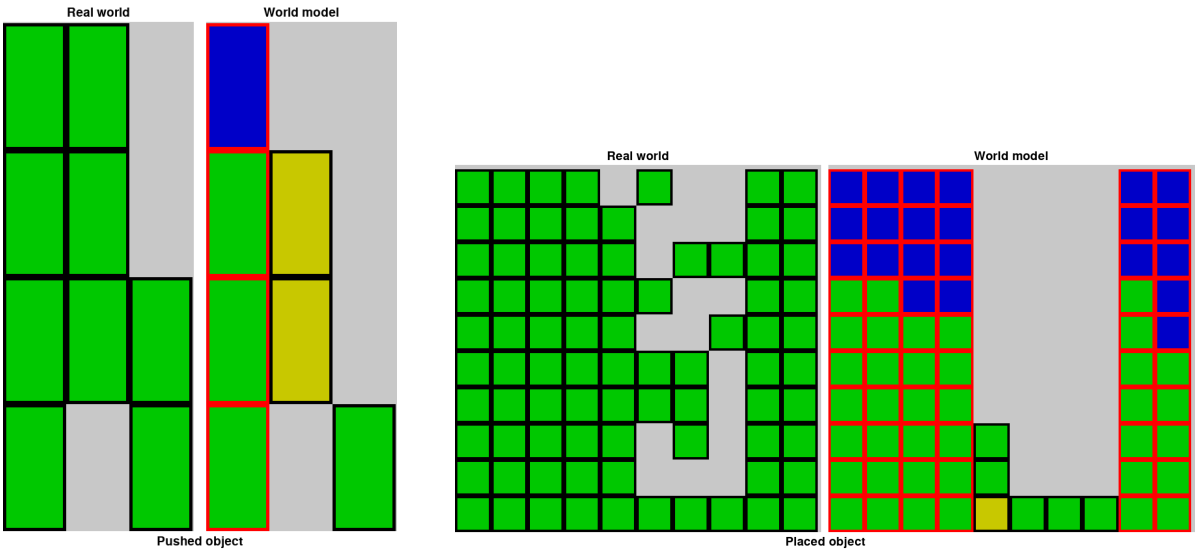
(b) After push action

Figure 4.1: Screenshots of the visualisation tool used in the proof of concept of before (a) and after (b) a push action was performed. The real world is shown on the left, the world model on the right.

4.2.1. Visualisation

A custom visualisation tool was built for quick insight into the current state of the world model and the real world. A screenshot of this visualisation is shown in Figure 4.1. It shows a top view of the real situation on the left, while on the right the world model is shown in a similar manner. The bottom of the figure represents the front of the shelf. In the real world a green, black-bordered square means the presence of an object. Grey means no object is present. In the world model an object is filled green if it has been observed, blue if it has only been inferred to be present and yellow if the object is believed to have moved to its current location due to a performed action of which the result has not yet been observed. Objects are black-bordered by default, but get red borders if the object is registered as not manipulable. Grey mean no object is known to be present. Figure 4.1 shows in the world model that the right most object was pushed last.

An altered version of this visualisation tool was later used for visualisation of the world model when deploying the reasoner in simulation and on the real robot. The same colour conventions as explained above are used throughout the rest of this work.



(a) Three by four rectangular objects

(b) Ten by ten square objects

Figure 4.2: Screenshots of the visualisation tool used in the proof of concept on shelves and objects of various dimensions.

4.2.2. First results

The system has been tested on the proposed scenario. Its reasoning capabilities have been tested and shown to work as expected. After that, the shelf and object dimensions were varied to check that the system could also handle rectangular object types and deeper or wider shelves. This was done to make sure the reasoning works correctly, independent of object or shelf dimensions. Figure 4.2a shows a case using rectangular objects and Figure 4.2b a case with an extraordinarily large shelf. Once these scenarios were all accomplished, SyRePOE was ready for validation in a more realistic case.

4.3. Simulation

The scenario as described in Section IV-A of the accompanying paper was set up in Gazebo and performed as explained in Section IV-B. Figure 4.3 shows a screenshot of the simulation, containing the robot, a set of empty shelves and a model object on top of the robot's base. The objects used for the simulation are modelled after packages of Albert Heijn 400g chocolate sprinkles. The rotational inertia of the objects around the vertical axis is higher than natural to limit accidental rotations and the centre of mass is lowered to decrease the chance of objects toppling. The pose of object visible in Figure 4.3 is considered the supply pose. Whenever the robot needs an object, one will spawn in exactly this pose. The shelf height is made to be just higher than the object is tall, like one would find in a retail store.

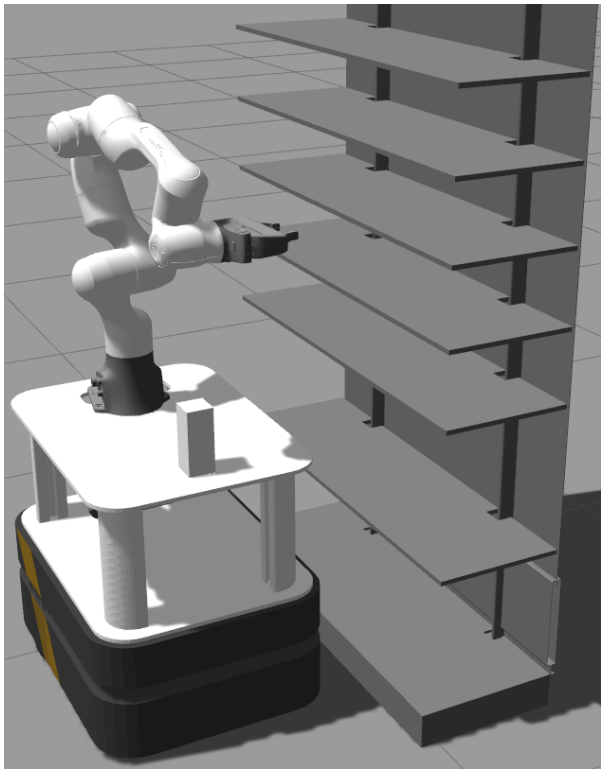


Figure 4.3: Screenshot of the Gazebo simulation of the AIRLab robot with a set of empty shelves and a model object.

An adapted version of the visualisation tool described in Section 4.2.1 is linked to this simulation as well. This way one can see what the robot believes to be going on on the shelf. Figure 4 of the accompanying paper shows an impression of this visualisation. The main differences between the visualisation used here and for the proof of concept are that here only the world model is visualised, not the real world case, and that the entire true size of the shelf is shown, not just a compactified grid of objects. The latter means that space in between objects on the shelf is shown to scale with the object size.

The rest of this section dives further into how the simulation is coupled to the planner and reasoner.

4.3.1. Action primitives

After the action planner has decided what action to perform, this high level action description needs to be translated to robot commands. This is done using the action primitives built for this simulation. For the push and place actions a sequence of waypoints is designed. The execution trajectories are then calculated by MoveIt [6].

The push action is designed to always push the object at the front of the shelf at a specified sideways position back by a distance equal to the depth of the object. The place action is designed to first grasp a new object from the supply and then place this object at the front of the shelf at a specified sideways position.

Due to the size of the gripper on this robot, very specific and dexterous grasping and placing tactics will have to be employed in order to place objects right next to each other and be able to push one object backwards without displacing the neighbouring objects with the gripper. Since dexterous manipulation is out of the scope of this work, this problem is worked around by increasing the inter-row distance of the objects on the shelf to dimensions that allow the gripper to freely place and push objects of one row without affecting other rows. Additionally, objects cannot be too wide or they don't fit the gripper. This is why, before deployment in a real retail store, the system should be tested on different robots. Using a suction gripper could be a first step towards resolving such issues.

4.3.2. Measurements

The measurements of the environment the reasoner needs consist of two channels. The first is visual, the second is haptic. In simulation the visual measurements are done simply by asking Gazebo for the pose of all objects and selecting those objects for which line of sight exists from the front side of the sub-environment to at least half of the frontal surface of the object. This represents that a perception module can only recognise it as an object if enough of the object is visible. The poses of these objects are passed to the reasoner.

The haptic feedback is realised by taking the squared sum of the efforts exerted by each joint. This signal is passed through a low-pass filter with a cutoff frequency of 4.0 Hz. A threshold is set at 2.0 Nm above the average filtered total effort over the last 10 samples. If the current sample of the filtered total effort exceeds this threshold, the action is aborted and considered to have failed. The cutoff frequency, threshold and number of samples mentioned above have been empirically determined to prevent the robot from crushing objects, while still not being so sensitive it aborts its movement when displacing the weight of multiple objects at once.

4.3.3. Semi-random configuration generator

To simulate the robot arriving at a shelf, where it does not know the configuration of objects on the shelf, a semi-random configuration generator is build. This generator spawns objects on the shelf in a semi-random pose. The poses of the objects are not completely random, because the majority of the objects on the majority of the shelves in a retail store will, at any given moment, at least approximately be in a grid configuration. Completely randomly placed objects on a shelf is a type of mess one does not often find in practice.

The semi-random configuration generator takes the grid as a starting point. It firstly gives every position in the grid a 50% chance of containing an object. It then gives every object a random sideways deviation of maximally one tenth the object width, picked from a uniform distribution. Lastly, all objects that are not at the back of the shelf nor have an object standing behind are given an 80% chance of being moved backwards. This displacement is of a random amount of maximally one object depth, picked from a uniform distribution.

4.3.4. Results

Several tests have been performed in the simulator with various initial configurations of objects. It has been shown SyRePOE is able to correctly infer presence of occluded objects and the system can reliably stock the shelf. An example execution is shown in Figure 4 of the accompanying paper.

4.4. Real Robot

To show SyRePOE is applicable in the real world, experiments have been performed with a real robot. For this a similar setup to the simulations is used, like explained in Sections IV-A and IV-C of the

accompanying paper. The main differences are the need for a perception module and the change in trajectory planning module. A general impression of the setup is given in Figure 4.4.

Figure 4.4: Impression image of SyRePOE stocking a shelf. On the left the Franka Emika Panda can be seen, together with objects and shelf marked with Aruco markers. On the top right the view of the shelf used for pose detection is visible. On the bottom right a visualisation of the world model is shown.

4.4.1. Trajectory planning and execution

In simulation motion trajectory planning and execution was realised by MoveIt [6]. MoveIt, however, did not always return satisfactory trajectories. Often trajectories that should have been straight lines had large joint reconfigurations halfway through the motion, sometimes leading to collisions with shelves or objects. In simulation, this is acceptable, but in reality it is unsafe. Therefore a different inverse kinematics (IK) algorithm was used here.

For picking up a new object from the robot base, three standard poses were predefined in joint space. These are a home pose, a pre-grasp pose and a grasp pose, all shown in Figure 4.5. By predefining these poses in joint space the grasp action could be guaranteed to be quick and the same on every execution.

(a) Home pose (b) Pregrasp pose (c) Grasp pose

Figure 4.5: Predefined joint space poses for grasping an object from the supply.

Initially the IK was used online for performing actions. Certain waypoints were passed to the IK to retrieve joint angles for each waypoint. Then for each joint the difference between the proposed joint angle for the next waypoint and the current joint angle were compared. If the difference exceeded an empirically determined threshold, the IK would be asked for another solution. This to prevent large motions of the arm when the end effector only has to move slightly. The online use of an IK algorithm retains the flexibility SyRePOE has towards shelf positions and dimensions. It was found, however, that execution became very slow (up to two minutes to place a single object), mostly because of the many tries the IK needed to find a satisfactory solution. To work around this, the IK was eventually used offline to find satisfactory solutions for the waypoints. These solutions were then saved and executed online. The waypoints used are a pre-place, place, pre-push and push pose for each row of objects on the shelf. This solution takes away the flexibility of the system towards shelf poses and dimensions, but significantly reduces execution time of actions.

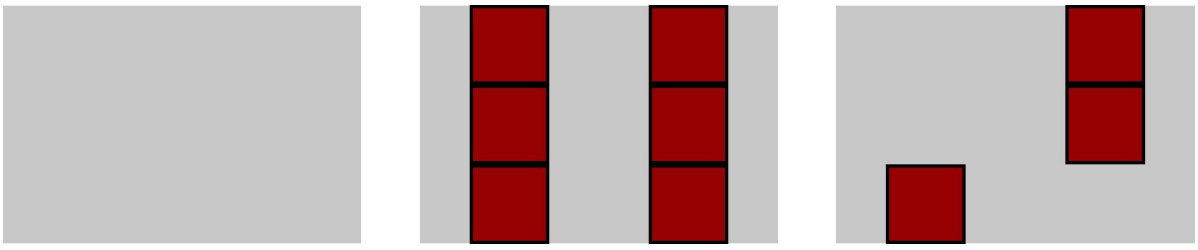
To track the generated trajectories an Active Inference Controller (AIC) [20] is deployed. The use of the AIC allows for compliance and easy access to opposing forces. Both of these are very useful when applying a push action to an object. The compliance prevents the robot from crushing objects, whereas the calculated opposing force is used for haptic feedback. More specifically, when the calculated opposing force exceeds an empirically determined threshold, the currently performed action is aborted and considered to have failed.

4.4.2. Perception

SyRePOE expects poses of the geometrical centres of the objects with respect to the front left corner of the sub-environment from the perception module. To detect these poses Aruco markers [21] were used. One marker is positioned at the front left corner of the shelf and each object has their own unique marker. By directly determining the relative pose of the object markers with respect to the shelf marker, the pose of the camera with respect to the shelf or robot becomes irrelevant. The camera, for which a Roboception rc_visard 160 color was used, was positioned on a tripod directly opposing the shelf for reliable marker detection. Even though each object has a unique marker, the markers are not used to identify the objects. This to reflect that it would be very hard for another perception module which does not use such markers to identify individual objects. Figure 4.4 shows an example view of the shelf and objects with highlighted markers as perceived by the camera in the top right.

4.4.3. Initial conditions

Three different initial conditions have been tested on the real robot: a completely empty shelf, a completely full shelf, and a partially full shelf. The approximate initial poses of the objects on the shelf for each scenario are shown in Figure 4.6. Three times the system was given the same task, to stock the shelf. The desired goal state is visualised in Figure 4.7. It is important to note that SyRePOE does not know the initial condition of the shelf beforehand.



(a) Empty shelf (b) Full shelf (c) Partially full shelf

Figure 4.6: Initial configurations for the three experiments performed with the real robot. The bottom of each figure represents the front side of the shelf.

In these experiments the poses of objects were allowed to deviate by a third of the object dimension in each direction to still be considered in the same pose. This pose tolerance helps deal with pose deviations caused by performed actions, and measurement errors.

The experiment starting with a full shelf was executed with the online use of the IK algorithm. The other two experiments use the IK offline.

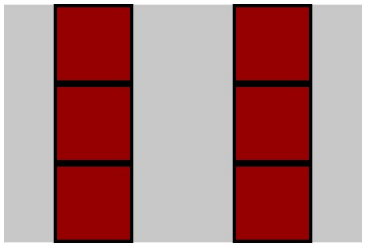


Figure 4.7: Goal configuration for the experiments with the real robot.

4.4.4. Results

In all three scenarios the system succeeded in stocking the shelf fully and having each object on the shelf registered in its world model. A video showing the execution and results of the experiments is available as supplementary material to this thesis. Figures 4.8, 4.9 and 4.10 show the results of the three scenarios. Figure 4.10 is also shown in Figure 5 in the accompanying paper, but is supplied here again for completeness.

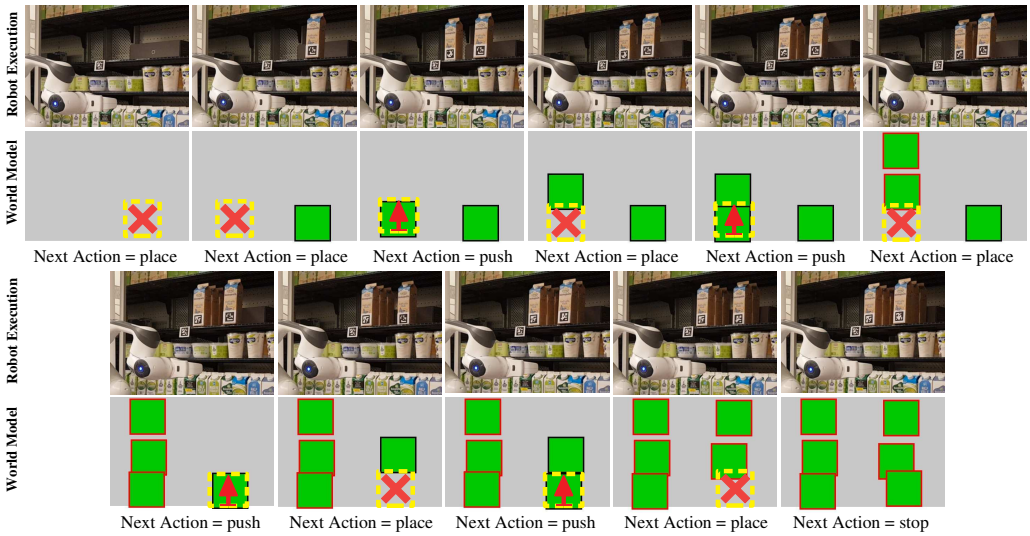


Figure 4.8: Snapshots from the real world experiments showing SyRePOE stocking an initially empty shelf. The environment is shown on top and the world model SyRePOE has at that time on the bottom, together with the selected action and action target in yellow dashed outline. The shelf is initialised with zero objects (top left), and is fully stocked in the end (bottom right).

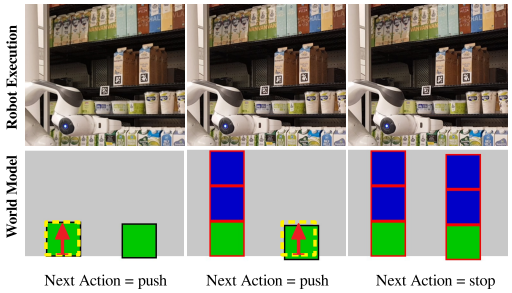


Figure 4.9: Snapshots from the real world experiments showing SyRePOE stocking an initially full shelf. The environment is shown on top and the world model SyRePOE has at that time on the bottom, together with the selected action and action target in yellow dashed outline. The shelf is initialised with six objects (left).

During execution the planning time of each iteration was measured. This planning time is defined as the time the system takes from having just completed the previous action to having decided which action to execute next. Over all experiments the mean planning time is 0.6176 seconds with the maximum at 1.165 seconds. For the empty shelf experiment the average planning time is 0.7666 seconds. For the

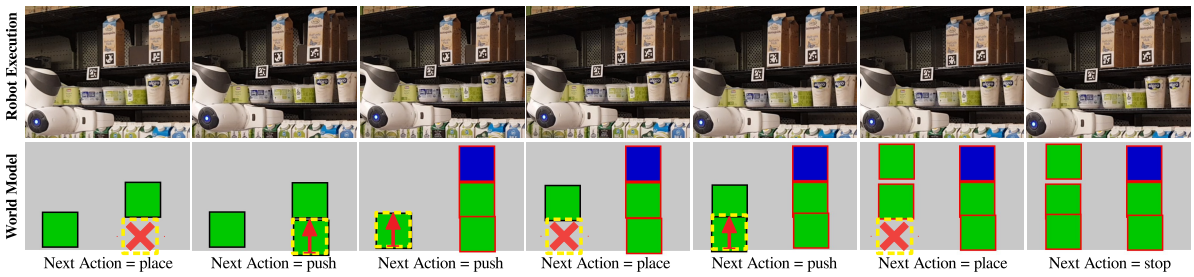


Figure 4.10: Snapshots from the real world experiments showing SyRePOE stocking an initially partially full shelf. The environment is shown on top and the world model SyRePOE has at that time on the bottom, together with the selected action and action target in yellow dashed outline. The shelf is initialised with three objects (left), and is fully stocked in the end (right).

full shelf experiment the average planning time is 0.5479 seconds. For the partially full shelf experiment the average planning time is 0.3924 seconds. These short planning times demonstrate reasoning in SyRePOE is quick relative to the action execution.

5

Future work

Besides integrating SyRePOE in existing ontology-based systems, like PMK [9] or Knowrob [3], as previously suggested, expanding SyRePOE itself could also improve its applicability. In this section, expansions are proposed in the areas of action selection, object representation and spatial reasoning.

5.1. Expanding action selection

The first suggestion for expanding the action selection process of SyRePOE is to incorporate actions, plans and planners in the ontology. The definition of available actions in the ontology would allow the reasoner to check the preconditions of such actions with the current state of the world model. It can then exclude irrelevant actions from the list of actions the planner can use, such to limit the search space and speed up planning. Planning could be sped up even more by saving generated plans in the ontology. The reasoner can then compare the current state of the world model with intermediate states of the world in those plans. This way, when the system is faced with a situation it has faced before, it does not need to replan its actions, but can instead follow a previously generated plan. In case no plan is available that contains the current state of the world model, the reasoner could suggest to use a specific planner, provided that planners are defined in the ontology as well. By defining different planners and their merits in the ontology, the reasoner would be able to suggest the most appropriate planner for the current situation. This is expected to generalise the applicability of SyRePOE and improve planning speed.

Secondly it is suggested to incorporate confidence in being able to successfully apply certain actions. In other words, how confident is the robot that when it plans this action, the result of the action will be exactly as expected? This confidence could be based on predefined values, past experience form this or other robots, the current state of the world model, and other knowledge about the environment. This way the system can select actions which are more reliable and predictable. If the confidence level of an action is too low, the action could be excluded from the actions available to solve the problem, since this essentially means that the expected outcome of such an action is unknown. This suggestion is expected to work especially well in combination with the expansion suggested above.

Lastly, action selection could be based on the amount of information a certain action can supply about still unknown areas. Currently actions are only selected based on whether or not they are expected to get the environment closer to the goal state. Gaining knowledge about the current state of the environment could be specified explicitly as an objective of the task.

5.2. Expanding object representation

Object representation could be expanded by incorporating object orientations. Currently the system assumes all objects are always facing forward. By including orientations of objects in the ontology the reasoner could suggest actions to rotate objects which are not facing in the right direction. Including orientations will make it significantly more difficult to reason about the effects of pushing one object with another. Such reasoning could in that case, for example, be outsources to a physics simulator.

Representation of pose uncertainty can also help improve object representation. Currently, when an object is perceived very close to where an object was already believed to be, these two objects are

assumed to be the same and the pose is updated to the last perceived pose. Instead, for example, a Bayesian filter could be used with the current pose from the ontology as prior and the newly perceived pose as a new measurement. Objects which are observed multiple times can be localised better and pose uncertainty can be estimated. Knowrob 2.0 [3] does something similar by saving all measurements of poses and calculating the relevant results, like expectation and uncertainty of the pose, on demand.

Thirdly, object shapes could be more explicitly specified in the ontology. SyRePOE currently assumes all objects are cuboid with a specified width and depth. With this comes the assumption that pushing one object with another is reliable. In practice, when an object is circular, like most cans, jars and bottles, such pushes are generally not reliable. If object shape is explicitly specified, the reasoner can conclude itself when such pushes are reliable and when they are not. This could tie in with the representation of confidence in successfully applying actions as suggested in Section 5.1.

Lastly it is suggested to incorporate left, right and front manipulability. Together with a parameter specifying if the sub-environment has solid walls on the sides of the sub-environment, left and right manipulability allows the system to better plan sideways push actions, increasing the solution space available to the system. Additionally, pose uncertainty after push actions could be reduced by pushing objects to the side of the sub-environment or against objects which cannot move to the left or right. This is particularly interesting since push actions are generally much less accurate than place actions. Adding front manipulability for completeness allows for manipulating objects in the same workspace from different directions. In retail stores this could be used in shelves on aisle-ends, which are accessible from the sides.

5.3. Expanding spatial reasoning

Spatial reasoning of SyRePOE could be improved by incorporating the third spatial dimension in pose definitions of objects. This allows for planning of stacking of objects and dealing with toppled objects. For example, when the system detects a non-manipulable object in a sub-environment with an unexpected block-size behind it, the reasoner might consider a toppled object lying there.

As a last suggestion, SyRePOE could be extended to incorporate reasoning over free space. Currently there is no registered difference between empty space and space of which it is unknown whether it is occupied or empty. Such reasoning could be incorporated, for example, by making an expectation of the observation that is about to be made. Any object that is indeed observed, would be confirmed to still be there. Any object that is observed but was not expected to be observed, would be added to the ontology. Any object that is not observed but was expected to be observed is checked for line of sight, considering the newly observed objects. If line of sight should exist, the object would be removed from the ontology. If line of sight does not exist, the object would be kept. Such reasoning would improve the tolerance towards external interference with the objects in the sub-environment, for example people taking objects away or placing new objects.

6

Conclusion

To come back to the design criteria mentioned in Section 1.3, the designed system can indeed stock retail shelves independent of the initial configuration of objects on the shelf, provided that these objects are approximately in a row of the grid configuration. This has been shown in simulation and with real world experiments. Three executions are shown in Figures 4.8, 4.9 and 4.10. SyRePOE can also suggest actions to move objects outside of a row into the nearest row, but this capability has not been tested due to physical limitations of the robot. The variety of initial conditions the system can handle could be further increased by incorporating orientations of objects in the world model. In all executions, both simulated and real, the experiment only ended when SyRePOE reported to be finished. This shows the system can indeed conclude itself when its task is accomplished. In the proof of concept phase (Section 4.2) the system has been tested on a variety of sub-environment sizes and object sizes, so to ensure its reasoning is independent of such factors. Section 3.2.2 has detailed how to add knowledge of new sub-environments and object types to the ontology and has shown this to be a simple procedure, which in some cases could even be automated.

Section 1.3 also posed a research question: Does reasoning over a symbolic world model allow for high situational awareness in robotic task planning with fully occluded objects? In this thesis SyRePOE has shown to be able to conclude the presence and poses of fully occluded objects by reasoning over knowledge of its environment, the current state of its symbolic world model, and feedback from multiple sensors, and to use this information for further task planning. This has been shown through the execution of a retail shelf stocking task, firstly in an abstract digital environment, then in a more realistic simulation, and finally on a real robot. Since the poses of fully occluded objects could by definition not be detected when only using conventional visual feedback, SyRePOE has shown to have higher situational awareness than systems relying on such visual feedback only.

Bibliography

[1] Michael Beetz. The institute for artificial intelligence! `https://ai.uni-bremen.de/`, 2020. Accessed: 2020-11-02.

[2] Michael Beetz, Ferenc Bálint-Benczédi, Nico Blodow, Daniel Nyga, Thiemo Wiedemeyer, and Zoltán-Csaba Marton. Robosherlock: Unstructured information processing for robot perception. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1549–1556. IEEE, 2015.

[3] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoğlu, and Georg Bartels. Knowrob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2018.

[4] Daniel Beßler, Mihai Pomarlan, and Michael Beetz. Owl-enabled assembly planning for robotic agents. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1684–1692, 2018.

[5] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[6] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[7] Michele Colledanchise and Petter Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics*, 33(2):372–389, 2016.

[8] Gerard De Melo, Fabian Suchanek, and Adam Pease. Integrating yago into the suggested upper merged ontology. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 190–193. IEEE, 2008.

[9] Mohammed Diab, Aliakbar Akbari, Muhayy Ud Din, and Jan Rosell. PMK—a knowledge processing framework for autonomous robotics perception and manipulation. *Sensors*, 19(5):1166, 2019.

[10] Mohammed Diab, Mihai Pomarlan, Daniel Beßler, Aliakbar Akbari, Jan Rosell, John Bateman, and Michael Beetz. An ontology for failure interpretation in automated planning and execution. In *Iberian Robotics conference*, pages 381–390. Springer, 2019.

[11] Sandro Rama Fiorini, Joel Luis Carbonera, Paulo Gonçalves, Vitor AM Jorge, Vítor Fortes Rey, Tamás Haidegger, Mara Abel, Signe A Redfield, Stephen Balakirsky, Veera Ragavan, et al. Extensions to the core ontology for robotics and automation. *Robotics and Computer-Integrated Manufacturing*, 33:3–11, 2015.

[12] Jörg Hoffmann. The metric-ff planning system: Translating“ignoring delete lists”to numeric state variables. *Journal of artificial intelligence research*, 20:291–341, 2003.

[13] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL-the planning domain definition language, 1998.

[14] Deborah L McGuinness, Frank Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[15] Mark A Musen. The protégé project: a look back and a look forward. *AI matters*, 1(4):4–12, 2015.

[16] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9, 2001.

[17] Ian Niles and Adam Pease. Linking lixicons and ontologies: Mapping wordnet to the suggested upper merged ontology. In *Ike*, pages 412–416, 2003.

[18] Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

[19] Alberto Olivares-Alarcos, Daniel Beßler, Alaa Khamis, Paulo Goncalves, Maki K Habib, Julita Bermejo-Alonso, Marcos Barreto, Mohammed Diab, Jan Rosell, Joao Quintas, et al. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*, 34, 2019.

[20] Corrado Pezzato, Riccardo Ferrari, and Carlos Hernández Corbato. A novel adaptive controller for robot manipulators based on active inference. *IEEE Robotics and Automation Letters*, 5(2): 2973–2980, 2020.

[21] Francisco J Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and vision Computing*, 76:38–47, 2018.

[22] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32:27, 2010.

[23] Jan Scheffczyk, Adam Pease, and Michael Ellsworth. Linking framenet to the suggested upper merged ontology. *Frontiers in artificial intelligence and applications*, 150:289, 2006.

[24] Larisa N Soldatova and Ross D King. An ontology of scientific experiments. *Journal of the Royal Society Interface*, 3(11):795–803, 2006.

[25] Moritz Tenorth and Michael Beetz. A unified representation for reasoning about robot actions, processes, and their effects on objects. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1351–1358. IEEE, 2012.

[26] Moritz Tenorth, Georg Bartels, and Michael Beetz. Knowledge-based specification of robot motions. In *ECAI*, pages 873–878, 2014.

[27] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN 1471-0684.

[28] Jan Winkler, Ferenc Bálint-Benczédi, Thiemo Wiedemeyer, Michael Beetz, Narunas Vaskevicius, Christian A Mueller, Tobias Fromm, and Andreas Birk. Knowledge-enabled robotic agents for shelf replenishment in cluttered retail environments. *arXiv preprint arXiv:1605.04177*, 2016.