Automated retail store restocking using PDDL and ROSPlan

RO47014 Knowledge representation and symbolic reasoning
Henk Jekel, 5609593

***Abstract*—The report describes a task planning solution utilizing PDDL and ROSPlan for robot restocking in a retail store. The intial PDDL knowledge base contains the world model. ROSPlan is used to extend the knowledge base based on a product ontology. The ontology uses store rules: explicit regulations that determine on which table products should be placed based on product characteristics. A plan is produced based on the extended knowledge base. The proposed KRR system has significant implications for the automation of stock management processes in retail stores.**

I. INTRODUCTION

The global rise of the elderly population has resulted in a scarcity of labor, particularly in developed nations with aging societies [1]. In contrast, the ever-expanding retail sector necessitates a greater workforce to maintain its operations and bring its products closer to consumers. To meet this demand, service robots can be deployed to relieve humans of repetitive duties, reduce cost for retailers while simultaneously ensuring the maintenance of service quality, supporting the growth of the retail industry [2]. A core component of such a service robot is task planning: *Organising actions to achieve goals, before starting to execute them* [3].

The project aims to implement a solution where a robot can pick up a set of products and place them in the correct locations within a simulated store environment. The used locations will be a stock table, a refrigerated table, a bin, and a non-refrigerated table. Products that have been damaged on the way to the store should not be stocked but discarded. A damaged product has lost part of its contents and has a reduced weight. The robot needs to acquire information about the properties of each product that imply whether it needs to be refrigerated or not, and whether the product has been damaged based on its mass. The robot will have to use this information and reason on where to place each product based on a set of fixed store rules.

This report presents a solution to a task planning problem for a simulated retail store restocking process. The solution uses a knowledge representation and symbolic reasoning (KRR) approach, figure 1. It enables the robot to reason about the best course of action based on its understanding of the environment and the objects within it, described in the initial knowledge base. The Planning Domain Definition Language (PDDL) [4] framework and ROSPlan [5] are employed to achieve the reasoning and planning required for the task. These components allow for informed decision-making based on a set of store rules and knowledge about the environment and the objects within it.

The solution incorporates a Python-based ontology that defines the product classes. By leveraging this ontology, the robot is able to perform task planning of a specific product instance by adding the appropriate predicates and goal state based on the ontology for the specific product instance to the initial knowledge base. The program adds a planning goal for each product instance based on its class. The generality of the ontology allows the robot to have a minimal amount of information on each product and still find a plan that is efficient and robust, even when faced with an unknown product.

The building blocks of the solution allow for handling of more complex scenarios if required. The goal of the report is to demonstrate a working solution that showcases the benefits of using an ontology for product generality. This allows the robot to have a compact knowledge base for task planning. By using this approach, the project aims to achieve greater efficiency, accuracy, and adaptability in decision-making in complex retail store environments.

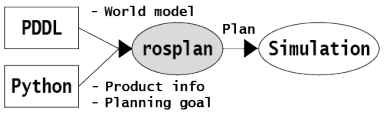

Fig. 1: Blockdiagram of the task planning

II. WORLD MODEL AND KNOWLEDGE BASE

Figure 2 illustrates the solution to the task planning problem, which utilizes a PDDL domain file containing durative actions for transferring product instances from the stock table to their respective goal table. An initial problem file is also provided, specifying the initial environment state and robot state. Section II-A elaborates on the creation of this initial knowledge base within the PDDL files. To enable the planner interface, explained in sectionIII), to perform task planning based on the complete knowledge base, a python script in the form of a problem interface is employed. This script reads a list of product instances and feeds it to a product ontology, which provides the initial product information and goal

state. The script then incorporates the product information and respective goal state into the knowledge base for each product instance, as discussed in section II-B. The knowledge base now forms a complete abstraction of the simulated world illustrated in figure 3, allowing the planner interface to solve the task planning problem.

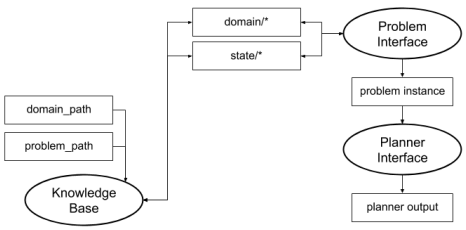

Fig. 2: The flow of information from KB to planner [5]

A. Initial knowledge base

The simulated world used in the Gazebo simulator is illustrated in Figure 3. As mentioned in the introduction, the world model has four tables, one for each product class, and they have been labeled as follows:

- *wp_table_1*: Stock
- *wp_table_2*: Refrigerated
- *wp_table_3*: Bin
- *wp_table_4*: Shelf

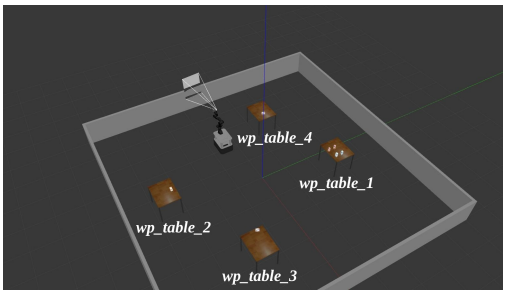

Fig. 3: The simulated world

The PDDL files use first-order logic to describe relevant aspects of the world in a structured and formal language. Specifically, predicates of arity 1 are employed to verify whether the robot is at a table waypoint. These predicates take the current waypoint of the robot as an input argument and compare it to the constant table waypoint to check for a match. As such, the predicates are denoted as *table_name/1*. The model includes a single robot with one gripper, which can move from waypoint to waypoint (*robot-at/2*). The waypoints are situated adjacent to the tables (*wp_table_x* where x denotes the table number), with a starting waypoint *wp0* where the robot base begins when the simulation initiates. The *waypoints.yaml* file represents the six degrees of freedom (DOF) coordinates that describe the position and orientation of the waypoints within the gazebo world frame. Additionally, the *objects.yaml* file contains the six DOF coordinates at which products must be picked up, while the *place.yaml* file contains the six DOF coordinates at which products must be placed.

The structured and formal language used to represent the state of the robot gripper in the PDDL files employs two predicates, *free/1* and *not_free/1*, which are mutually exclusive. In addition, the gripper can hold an object, which is captured using the *is_holding/2* predicate. The products that require stocking are classified as objects in the PDDL types, and their location is tracked through the use of the *object-at/2* predicate, which verifies whether an object is present at a specific waypoint. The initial position of all products is the stock table at waypoint *wp_table_1*. To indicate whether a product has been damaged on the way to the store and should be discarded, two mutually exclusive predicates, *is_full/1* and *is_not_full/1*, are defined. Similarly, to represent the need for refrigeration, the predicates *is_refrigerated/1* and *is_not_refrigerated/1* are implemented. When an object is successfully stored at its intended destination, it is assigned the *stored/1* predicate.

The mutually exclusive predicates discussed are negated versions of each other. Although PDDL provides the option to negate existing predicates by adding *not*, this feature is not supported by the POPF planner employed in the solution represented in this report. Therefore, the design choice was made to define the negated version of the predicates. This workaround is further discussed in Section III.

B. Problem interface

Upon running the *rosplan_place.launch* file, the ROSPlan system is initiated. This loads the PDDL domain and initial problem into the ROSPlan knowledge base. Additionally, the *load_product_info* python program is executed. This file transforms a product with its name, type and weight into a ROS service that adds the instance of the product as an object to the knowledge base. The program then adds the right predicates to the object into the knowledge base and finally adds a goal table on which the object should be stored into the knowledge base. The python program requires the product list, which contains the apriltag name, type of product (currently hagelslag and yoghurt), and the product weight. The *load_product_info* program utilizes a product ontology and is aware of the weight of full products and whether they require refrigeration. Then, depending on the product type, the knowledge base is extended with the appropriate predicates for its weight and refrigeration requirements. Once the necessary information of a product is added to the ROSPlan knowledge base, a concatenation of durative

actions, comprising the move, pick, and place actions, is executed to pick and place the product object. There are three durative place actions defined, one for each table. Allocation of the appropriate place action for a give product instance is based upon the governing store rules. These rules are initially presented in natural language, followed by their formal representation in first-order logic:

1) A full and perishable product must be placed on the refrigerated table.

$$\forall x\ object(x) \wedge\ is\_refrigerated(x) \wedge\ is\_full(x) \Rightarrow store\_in(x, refrigerated\_table)$$

2) A full and non-perishable product must be placed on the non-refrigerated table

$$\forall x\ object(x) \wedge\ is\_not\_refrigerated(x) \wedge\ is\_full(x) \Rightarrow store\_in(x, non\_refrigerated\_table)$$

3) An empty or partially full product must be placed in the bin

$$\forall x\ object(x) \wedge\ is\_not\_full(x) \Rightarrow store\_in(x, bin)$$

The presented task planning solution requires the following assumptions considering the world model to hold true:

1) The initial location of the store products are known.
2) The waypoints the robot base must move to are known.
3) All restocking locations have sufficient space to hold the products.
4) All products are within the mass limits of the gripper, for the picking and holding capabilities.
5) All products have an april_tag on them, in order for them to be perceived within the simulation.

III. REASONING AND PLANNING

Once the ROSPlan knowledge base has been updated with the relevant predicates for the product instances and their corresponding goal states, the planner interface, figure 2, is used to search for a viable plan to achieve these goal states. The objective of the plan is to store the following items:

TABLE I: Instances of product objects

Name	**Type**	**Weight**
april_tag_cube_8	”hagelslag”	0.5
april_tag_cube_23	”hagelslag”	0.1
april_tag_cube_15	”yoghurt”	1.0
april_tag_cube_24	”yoghurt”	0.5

The POPF (Partially Observable Planning with Fairness) planner is used to perform reasoning based on the complete knowledge base provided by the problem interface. It provides a plan given one exists. Subsection III-A elaborates on the design choices for the durative actions and how the definitions allow the POPF planner to find a viable plan.

A. Durative actions

As mentioned, the robot performs durative actions to reach the goal state. These durative actions, defined in the PDDL 2.1 framework [6], include move, pick, place, place1, and place2. These place actions represent placing on the non-refrigerated table, the bin table and the refrigerated table respectively.

The move action enables the robot to navigate between waypoints using the *robot-at/2* predicate that specifies a from and a to waypoint variable. Only one move action is needed between a pick and a place action. To avoid redundant moves, a new predicate called *can_move/1* was introduced. The predicate is a condition for the move action and is set to false as an effect of the move action. An effect of the pick and place actions is that *can_move/1* is set to true again, allowing the robot to move to the desired table after picking or placing the object.

The pick action instructs the robot to pick up an object from the stock table's adjacent waypoint, provided that the robot satisfies the conditions of being located at that waypoint and having a free gripper. The effect of the action is that the robot is holding the object, while the object is no longer on the stock table. Additionally, the action enables the robot to perform one move action after the pick action to navigate to the destination waypoint to place the object.

To circumvent the limitations of the POPF planner's lack of support for disjunctive-preconditions and conditional-effects, the original place action is split into three separate versions, each with distinct condition and effect elements. The place action is utilized for placing full and non-refrigerated objects onto the shelf table, while the place1 action is used for placing non-full objects onto the bin table. Finally, the place2 action is designed for placing full and refrigerated objects onto the refrigerator table.

The place actions have conditions that require the robot to be at the designated waypoint and holding the object. Upon execution of these actions, the effects include that the object is placed at its designated location, that the robot is no longer holding the object, and that the gripper is empty. Another effect is that the robot can perform one move action after a place action. Additionally, the object is assigned the stored predicate, indicating that the goal for this object has been achieved, and the plan proceeds to the next object on the list. Once all objects on the list have been processed, the plan is complete.

IV. RESULTS

Given the knowledge base, initial state, and goals, the planner produced a plan as shown in Figure 4. In accordance with the design of the task planning system,

the planner plans the following actions for each product instance: the robot has to move to the stock table to pick up the product, it then has to move to the appropriate table to place the object. As the figure indicates, the plan correctly identifies the tables at which the object instances should be placed.

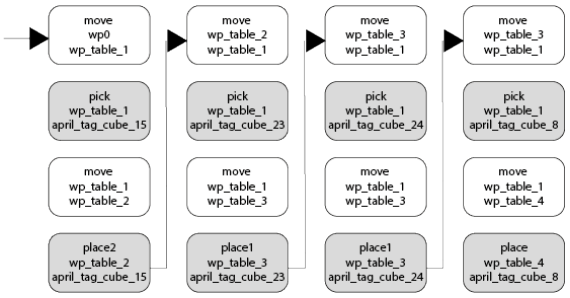

Fig. 4: Generated plan

TABLE II: Product predicates and end locations

Name	Full	Refrigerated	End location
april_tag_cube_15	True	True	Refrigerator table
april_tag_cube_23	Not	Not	Bin table
april_tag_cube_24	Not	True	Bin table
april_tag_cube_8	True	Not	Shelf table

The plan's correctness has been verified; however, several issues arose when attempting to simulate the plan in a customized world. A plethora of ROS errors were encountered when deploying the task planning system in the custom world depicted in Figure 3 to execute the plan presented in Figure 4. One consequence of the errors was that Gazebo was unable to render the robot in the custom world. A logical debug step was to attempt to plan in the course provided world. However, when introducing additional products into this standard world via April tags, the robot failed to recognize these tags, even when using the base world provided in the course. The coordinates of the additional objects were stored in the objects.yaml file, as described in Section II-A. It is possible that an additional step is necessary for proper functionality or that the April tags are not working correctly. In an attempt to provide an even simpler combination of the task planning solution with the simulated world, the durative actions where tested in the base world provided in the course. When testing the durative actions defined using the base world, the robot could move and occasionally pick up products, but it never succeeded in placing them. Even in a simplified plan that only involved move, pick, move and place of a single product, the robot failed to place the item. A ROS warning message indicated that the location to place the item was "out of reach," despite attempts to adjust the waypoints. A video illustrating the outcome of this task is available in the GitLab repository at the following link: test_video.com. This behavior implies that there may be an error in the ROSPlan action interface responsible for executing the placing action. Further investigation is required to resolve this specific issue. However, due to time constraints and the fact that the problem did not relate to the primary learning objectives of this course, it was not given priority and remained unresolved. Another issue was that the robot arm spawns in a singularity, hindering the proper functioning of the pick action. In conclusion, the plan presented in Figure 4 is correct based on our design choices and store rules, but the simulation did not visualize the plan's execution.

V. DISCUSSION

This report has demonstrated the effectiveness of the designed task planning system in the context of automated retail store restocking. To improve the current design, further research could explore the use of an alternative planner such as SMTplan+ [7]. As described in section II-A, the current design uses a workaround where negated versions of predicates are introduced to enable the POPF planner to solve the task planning problem. The POPF planner also forced the solution to include separate place actions for each table, each with distinct condition and effect elements. While this strategy proved effective for the current problem, it will become tedious when dealing with a larger amount of tables at which products need to be placed. Therefore, future work should investigate alternative planning approaches to address these challenges.

To achieve a functional simulation, it is important to prioritize resolving the issue with the placing task. Investigation of the place action interface could potentially address the problem, or alternatively, tuning the location definition using quaternions via a place_client could be a potential solution. ROS provided warnings that the item placement location was "out of reach," indicating that the location definition could be causing the issue. Another crucial step would be to address the issue where the robot arm spawns in a singularity, hindering the proper functioning of the pick action. A possible solution for this issue is to remove the "panda" tags from the world file. Additionally, reading issues related to the april_tags must be resolved. The simulated world could also be further customized by adding more products and tables, and incorporating an alternative planner to prevent the need for tedious redefining of actions.

REFERENCES

[1] N. Kishida and H. Nishiura, "Demographic supply-demand imbalance in industrial structure in the super-aged nation japan," *Theoretical Biology and Medical Modelling*, vol. 15, pp. 1–10, 2018.

[2] P. K. Donepudi, "Robots in retail marketing: A timely opportunity," *Global Disclosure of Economics and Business*, vol. 9, no. 2, pp. 97–106, 2020.

[3] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, vol. 247, pp. 10–44, 2017.
[4] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, "Discrete and deterministic planning," in *An Introduction to the Planning Domain Definition Language*. Springer, 2019, pp. 13–61.
[5] "Rosplan." [Online]. Available: http://kcl-planning.github.io/ROSPlan/
[6] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *CoRR*, vol. abs/1106.4561, 2011. [Online]. Available: http://arxiv.org/abs/1106.4561
[7] M. Cashmore, M. Fox, D. Long, and D. Magazzeni, "A compilation of the full pddl+ language into smt," in *Proceedings of the international conference on automated planning and scheduling*, vol. 26, 2016, pp. 79–87.