TIAGo's Riddle in a Post Office

Group 06 - Edoardo Panichi 5630444, Gioele Buriani 5629888, Severin Woernle 5629438

***Abstract*— This paper discusses how to implement a simulated post office warehouse where a TIAGo robot manages some deliveries based on only partial information regarding the addressees. The idea behind the store rules of this project is based on Einstein's riddle, which makes a robot much more efficient than a human at this job. The robot must understand where each object needs to be delivered and move it accordingly in the room. To achieve this result we used Prolog and PDDL 2.1 for the reasoning part and ROS to make everything work together.**

I. Introduction

"It is comparatively easy to make computers exhibit adult-level performance in solving problems on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility." [1]. Even though formulated in 1988, the Moravec's paradox perfectly explains the huge gap between robots' brain skills and motor skills that is well present even nowadays. It is by keeping this concept in mind that we tried to think of an application where our TIAGo robot would not only be able to replicate what a human being can do, but where it can also prove to be much more efficient.

One of the fields where robots can easily outperform humans is logical thinking: robots can make hundreds of logical inferences in a fraction of a second while a human being would take hours to solve the same problem. For this reason, in order to show the potentiality of our robot, we decided to create a logical problem that has to be solved by the operator in order to understand the action that needs to be taken. It will be clear that, while a robot only takes a few seconds to understand what to do, for an average human operator several hours would be necessary.

We decided to structure our problem as a form of Einstein's riddle [2]. This famous logical problem is said to be invented by Albert Einstein when he was a kid and that only 2 percent of the population would be able to solve it. The main idea is that there are five people of five different nationalities, living in houses of five different colours, owning five different types of pets, smoking five different brands of cigarettes and drinking five different types of beverages. The player does not know the relation between all these elements (e.g. who owns which pet) and has to figure out all this information with just a series of 15 hints such as '*The Brit lives in the house with red walls*' or '*The man who smokes Blends has a next-door neighbor who drinks water*' (Figure 1).

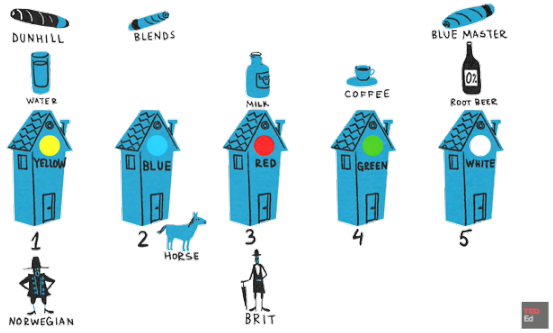

Fig. 1: A TED-Ed animation that shows the process of solving the riddle

We decided to apply this idea to a post office warehouse, where the operator only receives partial information about who should receive each object and therefore has to infer it by progressively building its own knowledge base. After understanding to which city a certain object has to be delivered, the operator will then need to pick the object up from its location and place it on the shelf that corresponds to the correct city.

In particular, for the riddle part of our problem we decided to use Prolog [3]. We felt like it was the ideal program for what we needed: we have to input logical information for the system to automatically build a knowledge base by inferring new information. Finally, it has to answer a simple query regarding where should a certain object be delivered.

After the logical reasoning part is finished and the robot has determined that a specific object should be delivered in a specific place, it has to move the object in the room accordingly. To manage this part we decided to use PDDL 2.1 [4] since it has already proven to work extremely well with these kinds of tasks involving durative actions. The goal to be fed to the PDDL

reasoner will obviously depend on the output of Prolog.

Finally, to make the two programs work together we use ROS that will also manage running the simulated environment.

II. World Model and Knowledge Base

We will now discuss more specifically how we implemented our robotic task and how we modelled our world. We will start by defining exactly the riddle part and then we will show how we dealt with the simulation environment.

First of all, we had to reformulate Einstein's riddle so that it was applicable to our case. In fact, we assumed we only have the people's names and the description of where they live. In particular: There are five people called: *Jan*, *Karin*, *Monique*, *Peter* and *Rob*.
They live in five different streets: *Bankastraat*, *Bloemstraat*, *Dorpstraat*, *Hoogstraat* and *Kerkstraat*.
They live in five different cities: *Amsterdam*, *Den Haag*, *Eindhoven*, *Rotterdam* and *Utrecht*.
Their houses are of five different types: *apartment*, *cottage*, *palace*, *trailer* and *villa*.
Finally, they have to receive five different objects from the post office: *bracelet*, *phone*, *ring*, *tablet* and *watch*.
Additionally, we assumed all the deliveries have to be done in a specific temporal order.

The hints and the question for our own version of the riddle therefore are:

- *Peter lives in a house on Dorpstraat*
- *The bracelet has to be delivered to a house on Kerkstraat*
- *Bloemstraat is a street in Rotterdam*
- *Karin should receive her delivery right before Rob*
- *Karin lives in Utrecht*
- *The ring has to be delivered to a villa*
- *Jan lives in an apartment*
- *The third delivery should be in Den Haag*
- *The first item should be delivered to a house on Bankastraat*
- *The watch should be delivered right before or after visiting a cottage*
- *The apartment should be visited right before or after delivering the phone*
- *In Eindhoven there is a palace*
- *There is a trailer on Hoogstraat*
- *Monique should receive her object right before or after the delivery in Bankastraat*
- *Amsterdam should be visited right before or after delivering to a cottage*

Question: *In which city should the tablet be delivered?*

This is just a sample question that follows the idea behind the riddle. From a practical point of view, in order to answer this question, the system has to infer all the relations and can therefore be asked any other question, such as *Who should receive the phone?* or *On which street should the ring be delivered?*.

In order to implement this riddle on Prolog, we decided to use a list called `Houses` that reflected the structure of the problem (five people, each living in a type of house, on a street, in a city and each due an object) where all elements are variables. Then we translated all the hints into members for the structure. Then, since we are interested specifically in the cities in which the object has to be delivered, we added them as variables in the list (e.g. `City_tablet`). Finally, we defined a complex term called `find_cities`, which is implied by the list and its members and returns the value for the five cities variables. We also added another predicate called `find_city` which can be queried singularly to find the destination of a single object. By simply querying, for example, `?- find_city(X, ring).` We directly obtain the name of the city in which the ring should be delivered. This implementation was inspired by a solution to Einstein's riddle that we found online [5].

As explained above, through the usage of Prolog we are able to infer the addressee of each package in the warehouse. Once this piece of information is known, TIAGo needs to arrange the shipping orders among the available shelves. We decided to reserve a shelf per city (or per person in our specific case).

To plan a sequence of actions able to obtain the desired result, we employed a PDDL 2.1 program [6]. In the *domain.pddl* file four durative-action are implemented: *move, pick_right, pick_left* and *place*. In addition, we added a normal action named *full_shelf* which solves the conflict in reasoning when a shelf is full. None of these actions can start if another one is already active. To achieve this, we defined a predicate named `(no_actions_in_exec ?v - robot)`, which is true if no other actions are currently executed. The design decision of avoiding parallel execution of the actions has been made because with parallel actions the simulation does not work properly. The reasoning about where to place an object on the shelves is implemented in the durative-action *place*.

III. Reasoning and planning

We will now see how the previous elements got combined in order to make TIAGo pick the correct

objects and place them in the right location in the simulated environment.

First of all, as explained in the previous section, we implemented the riddle in a Prolog file in which in the end two types of queries could be made:
`find_cities(City_tablet, City_watch, City_phone, City_ring, City_bracelet)`
and
e.g. `find_city(X, tablet)`
For the final application, we decided to use only the first one that returns the list of all the city names corresponding to the respective objects.

We then created a ROS node in Python from which we used rosprolog to create a query and solve it using the Prolog file. The result is then stored in the form of a dictionary and its data can be therefore used from the Python file. For this implementation we took inspiration from our professor's example [7] and ROSplan tutorials [8].

Now, in order to use this data in the simulation we had to transpose the objects and cities of the "real" world into the elements of the simulation. In particular, for the objects:
bracelet → `aruco_cube_111`
phone → `aruco_cube_222`
ring → `aruco_cube_333`
tablet → `aruco_cube_444`
watch → `aruco_cube_582`
While for the cities:
amsterdam → `shelf_1`
denHaag → `shelf_2`
eindhoven → `shelf_3`
rotterdam → `shelf_4`
utrecht → `shelf_5`.

Once done that, we generate the problem thanks to a dedicated service in ROSplan. Subsequently, for each object present, we generate a new goal with the following structure: `new_goal = KnowledgeItem()`, where in particular, the destination shelf is chosen according to the city found solving the riddle with ROSprolog. To add the goals to the ROSplan problem, we call another service that handles the update procedure.

Since the node has to be executed only once, the process is brutally interrupted with `sys.exit(1)`.

After assigning the desired goals to the ROSplan problem, the PDDL reasoner is used to plan the right order of actions to solve the problem. The *move* action keeps track of the robot's position and updates its position.

For the *pick* action we defined two actions, one for picking the cube with the right gripper and one for picking the cube with the left gripper. Thereby we ensure that TIAGo can reach every cube on the table. To give TIAGo the knowledge of which gripper to use, we assigned to each cube the knowledge directly. In reality, it would be much more convenient if TIAGo automatically detects whether the cube is to its left or right side. But because we do not want to modify the simulation itself, we simplified the problem. The picking action has the following effects on the knowledge base. First, after the action, TIAGo holds the picked object in the corresponding gripper. Second, the location where the object was placed is no longer occupied and free now. And third, it is updated that the object is no longer stored at the previous location.

In the *place* action the reasoning about where to place the objects on the shelf is implemented. Figure 2 shows the configuration of the shelves.

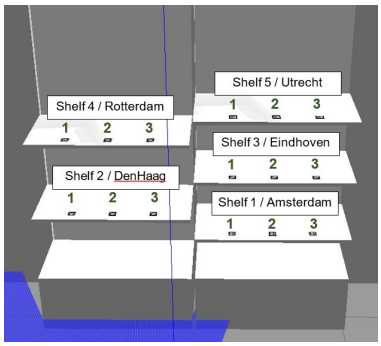

Fig. 2: Configuration of the shelves and the corresponding locations

Each shelf has three hard-coded locations assigned to it. The predicate '`(previous ?loc1 ?loc2 - location)`' is used to define the order in which the cubes should be stored. The cube is stored at location loc2 if the location is free and if location loc1 is occupied. To be able to use the previous predicate for the first location we added an imaginary prefixed location that is already occupied at the start. Thanks to the `previous` predicate, first an object is placed at the first location, then at the second and so on. To ensure that TIAGo places the objects in the right order, we assume that all locations are free, or filled up starting from the first location. In order to be able to store an object at all locations with both grippers, we added new waypoints to the cabinets. Finally, we assigned a specific waypoint to each location depending on the gripper. The effects of the *place* action are the opposite

of the *pick* action. First, TIAGo no longer holds the object in the corresponding gripper. Second, the location where the object was placed is now occupied and no longer free. And third, the object is now stored at the corresponding location.

The last action *full_shelf* is needed to ensure that the reasoner does not fail once a shelf is full. In that case, TIAGo will not pick the cube and leave it on the table.

Our original PDDL version failed with the simulation for some unknown reasons, the left gripper did not find a path. That is why we changed the corresponding CPP files back to their default status and created a second version of the PDDL file. [9] The second version has additional waypoints so TIAGo can grab all cubes with the right gripper.

IV. Results

Talking about the results, overall the reasoning part works extremely well: the Prolog code works perfectly and always updates the PDDL problem with the correct goals.

As for the PDDL part, we had some minor problems with the reasoner since sometimes it produces a sub-optimal solution. However, it always found a solution that satisfies the problem goals.

The main problems were encountered in the simulation part: since the movement of the robot is calculated with an RRT algorithm, its randomizing the results would often make the robot not able to find a valid path and therefore fail the simulation. In order to face this problem, we implemented a pre-position for the *pick* action and both a pre-position and post-position for the *place* action. This helped guide the arm correctly and therefore reduced the occurrence of faults.

One of the results we obtained with the simulation can be seen through this link:
`https://youtu.be/XmrigG12jSs`

As can be seen in the video, the robot had an initially sub-optimal solution where it placed the first object on the wrong shelf. Later it however corrects itself placing it in the right one. After placing the first two objects the robot had some problems related to the simulation and started failing by dropping the third and fourth objects.

Overall, the *pick* action worked in all cases, while the *place* action had some problems probably related to the narrow space between the shelves.

V. Discussion

We are now going to share some specific considerations regarding our solution and possible points for improvement.

The first point we would like to make is related to the Prolog reasoning. Unfortunately, the solution implemented at the moment is strictly based on the structure of the original riddle. We chose it more to show the potentiality of the robotic reasoning, rather than because it could resemble a real-life scenario. As it is now the problem is hardly scalable, but in a more realistic scenario where there might be many more products, but maybe only little information is missing from each address, this system would work without any problem.

Another interesting point of discussion regards the solution found by the PDDL solver (https://popf-cloud-solver.herokuapp.com) we picked. This solver is not optimal, so sometimes it may happen that the sequence of actions done by TIAGo leads to the goal, but some actions taken are unnecessary. For example, it may go to a table where it does not pick anything, or it might place momentarily an object in an temporary position. The problem can be solved by adopting an optimal solver, keeping into consideration that it might be slower.

As for the possible future developments and points of improvement, one of the first operations that could be done is changing the querying system of the robot. Right now the robot makes one general query at the beginning of the execution where it asks for the location of all the possible objects using the predicate `find_cities`. A possible alternative is for the robot to first reach an object, detect what type of object it is, and then query singularly for that object with the predicate `find_city`. Then after it placed the object in the right place it starts looking for the next one. This looks like a more realistic and scalable solution.

Another point of improvement is related to the simulation. Right now we have an effective solution, not particularly elegant, that involves associating each object with a specific aruco cube and each city with a shelf from 1 to 5. While this is not actually a big problem, from a simulation point of view it would be better to at least modify the names of the elements inside the simulation to directly resemble the ”real” world. A further development could involve also changing the image and the shape of the objects to make the simulation even more realistic.

References

[1] Hans Moravec. *Mind children: The future of robot and human intelligence*. Harvard University Press, 1988.

[2] Jeremy Stangroom. *Einstein's Riddle: 50 Riddles, Puzzles, and Conundrums to Stretch Your Mind*. A&C Black, 2009.

[3] Patrick Blackburn, Johannes Bos, and Kristina Striegnitz. *Learn prolog now!* Vol. 2012. College Publications Londres, 2006.

[4] Patrik Haslum et al. "An introduction to the planning domain definition language". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 13.2 (2019), pp. 1–187.

[5] *A Prolog solution to Einstein's riddle*. URL: https://gist.github.com/jgilchrist/1b85b04f4395057972dd.

[6] *PDDL file used to control TIAGo*. URL: http://editor.planning.domains/#edit_session=b0SVKwClX0DU0wg.

[7] *Example: Using Prolog in from Python code*. URL: https://github.com/chcorbato/rosprolog_examples.

[8] *ROSplan Tutorial*. URL: http://kcl-planning.github.io/ROSPlan//tutorials/tutorial_08.

[9] *PDDL file without left gripper*. URL: http://editor.planning.domains/#edit_session=uJ7tuItfDpe0hGD.